SIAM J. COMPUT.
Vol. 36, No. 1, pp. 1–15

SCHEDULING SPLIT INTERVALS*

R. BAR-YEHUDA†, M. M. HALLDÓRSSON‡, J. (S.) NAOR†, H. SHACHNAI†, AND I. SHAPIRA†

Abstract. We consider the problem of scheduling jobs that are given as *groups* of nonintersecting segments on the real line. Each job J_j is associated with an interval, I_j, which consists of up to t segments, for some $t \geq 1$, and a weight (profit), w_j; two jobs are in conflict if their intervals intersect. Such jobs show up in a wide range of applications, including the transmission of continuous-media data, allocation of linear resources (e.g., bandwidth in linear processor arrays), and computational biology/geometry. The objective is to schedule a subset of nonconflicting jobs of maximum total weight.

Our problem can be formulated as the problem of finding a *maximum weight independent set* in a *t-interval* graph (the special case of $t = 1$ is an ordinary interval graph). We show that, for $t \geq 2$, this problem is APX-hard, even for highly restricted instances. Our main result is a $2t$-approximation algorithm for general instances. This is based on a novel *fractional* version of the Local Ratio technique. One implication of this result is the first constant factor approximation for nonoverlapping alignment of genomic sequences. We also derive a bicriteria *polynomial time approximation scheme* for a restricted subclass of t-interval graphs.

Key words. interval graph, independent set, scheduling, approximation algorithm

AMS subject classifications. 68Q25, 68W25, 90C59

DOI. 10.1137/S0097539703437843

1. Introduction.

We consider the problem of scheduling jobs that are given as *groups* of nonintersecting segments on the real line. Each job J_j is associated with a *t-interval*, I_j, which consists of up to t disjoint segments, for some $t \geq 1$, and a weight (profit), w_j; two jobs are in conflict if any of their segments intersect. The objective is to schedule on a single machine a subset of nonconflicting jobs whose total weight is maximum.

An instance of our problem can be modeled as the intersection graph of t-intervals, known as a *t-interval graph*. Each vertex in the graph corresponds to an interval that has been *"split"* into t parts, or segments, such that two vertices u and v are adjacent if and only if some segment in the interval corresponding to u intersects with some segment in the interval corresponding to v (see Figure 1). In the special case where intersections can occur only between the ith segments of two intervals, $1 \leq i \leq t$, we get the subclass of *t-union graphs*. (A precise definition is given in section 2.1.) Note that 1-interval graphs are precisely interval graphs. Our problem can be viewed as the *maximum weight independent set (MWIS) problem* restricted to a weighted t-interval graph $G(V, E)$, where we seek a subset of nonadjacent vertices $U \subseteq V$, such that the weight of U is maximized.

*Received by the editors November 25, 2003; accepted for publication (in revised form) August 7, 2004; published electronically April 21, 2006. A preliminary version appeared in the *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2002, pp. 732–741.

http://www.siam.org/journals/sicomp/36-1/43784.html

†Computer Science Department, Technion, Haifa 32000, Israel (reuven@cs.technion.ac.il, naor@cs.technion.ac.il, hadas@cs.technion.ac.il, csira@cs.technion.ac.il). The research of the first author was supported by the fund for the promotion of research at the Technion. The research of the third author was supported in part by US-Israel BSF grant 2002276 and by EU contract IST-1999-14084 (APPOL II).

‡Department of Computer Science, University of Iceland, IS-107 Reykjavik, Iceland (mmh@hi.is).

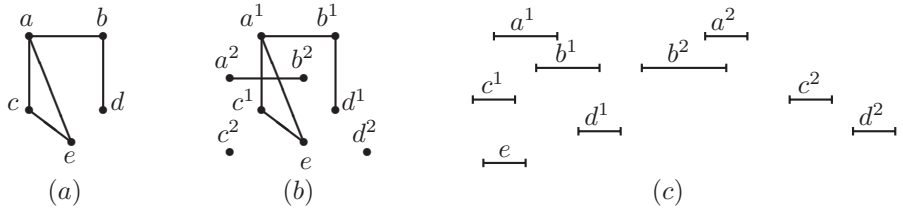


FIG. 1. *A* 2*-interval graph* (a), *corresponding interval (segment intersection) graph* (b), *and interval system* (c).

We describe below several practical scenarios involving t-interval graphs.

Transmission of continuous-media data. Traditional multimedia servers transmit data to the clients by *broadcasting* video programs at prespecified times. Modern systems allow us to replace broadcasts with the allocation of video data streams to individual clients *upon request* for some time interval (see, e.g., [34, 6]). In this operation mode, a client may wish to take a break and resume viewing the program at some later time. This scenario is natural in, for example, video programs that are used in remote education [25].

Suppose that a client starts viewing a program at time t_0. At time t_1 the client takes a break and resumes viewing the program at t_2, until the end of the program (at t_3). This scenario can be described by a *split interval*, I, that consists of two segments: $I^1 = (t_0, t_1)$ and $I^2 = (t_2, t_3)$.

The scheduler may get many requests formed as split intervals; each request is associated with a profit which is gained by the system only if *all* of the segments corresponding to the request are scheduled. The goal is to schedule a subset of nonoverlapping requests that maximizes the total profit, i.e., find an MWIS in the intersection graph of the split intervals.

Most of the previous work in this area describes analytic models (e.g., [31]) or experimental studies in which VCR-like operations can be used by the clients (see [6, 12, 34, 45]); however, these studies focus on the efficient use of system resources while supporting such operations rather than on the scheduling problem.

Linear resource allocation. Another application is allocation of multiple linear resources [22]. Requests for a linear resource can be modeled as intervals on a line; two requests for a resource can be scheduled together unless their intervals overlap. A disk drive is a linear resource when requests are for contiguous blocks [38]. A linear array network is a linear resource, since a request for bandwidth between processors i and j requires that bandwidth be allocated on *all* intervening edges. Consider a computer system that consists of a linear array network and a large disk, shared by a set of processors. A scheduler must decide when to schedule requests, where each request may comprise of distinct requests to these two linear resources, e.g., "a certain amount of bandwidth between processors 4 and 7, and a lock on blocks 1000-1200 of the disk." Two requests are in conflict if they overlap on the disk or in their bandwidth requirements. Thus, when the goal is to maximize the number of requests satisfied by the system, we get an instance of the MWIS problem on the subclass of 2-*union* graphs. Indeed, each segment in a 2-interval represents an allocation of one of the resources (e.g., the first segment is bandwidth allocation, and the second segment is the allocation of blocks on the disk to a given request). In general, with t different resources we get an instance of the MWIS on a t-union graph.

Genomic sequence similarity. One of the more fundamental problems in computational biology is to determine the similarity of substructures. We consider here genomic sequences (DNA, protein) and define the substructures to be contiguous subsequences. The similarity score of a substructure is generally related to the local alignment, or editing distance, between the two subsequences.

When considering the total similarity of two whole sequences, we can view this as being made up by the combination of individual substructures. Due to genomic rearrangements, the order of the subsequences need not be preserved between the two genomes. The nonoverlapping local alignment problem seeks a collection of substructures, each corresponding to pairs (S_i, T_i) of subsequences of the genomes S and T, where none of the subsequences overlap (neither in S nor in T). The objective is to maximize the sum of the similarity scores of the substructures.

As an example, suppose $S = xxxAxxBC$ and $T = C'yyA'zzB'$, where A, B, and C are sequences with similarity scores of 15, 20, and 11 to sequences A', B', and C', respectively. Then, the total similarity of S and T would be 46.

We may assume that the input sequences have been preprocessed to give subsequence pairs with nonzero similarity. Each such substructure (S_i, T_i) corresponds to a 2-interval, formed by the interval that S_i forms with S one on hand, and the interval that T_i forms with T on the other hand. The nonoverlapping restriction of the problem implies that the set of 2-intervals that we find needs to be mutually independent. Hence, the nonoverlapping local alignment problem corresponds to the MWIS in 2-union graphs.

The common total similarity of t sequences simultaneously can similarly be modeled as the MWIS problem in t-union graphs. Previously, the problem was considered only in the case where the projections of input boxes did not contain one another, i.e., the case of proper t-union graphs. While making the problem easier, this restriction is not intrinsic to the biological problem.

Computational geometry. The problem of finding an independent set among a set of multidimensional axis-parallel boxes is of independent interest in computational geometry. It corresponds to the MWIS problem in t-union graphs.

1.1. Our results. We provide a comprehensive study of the MWIS problem in t-interval graphs. In section 2, we show that the problem is APX-hard even on highly restricted instances, namely, on $(2, 2)$-union graphs (defined in section 2.1). In section 3 we discuss some structural properties of t-interval graphs. In particular, we derive a bound on the inductiveness of a t-interval graph. As a corollary, we extend the best bound known on the chromatic number of t-interval graphs of Gyárfás [18]. We show this bound to be asymptotically optimal.

In section 3.2, we study the MWIS problem on 2-interval graphs. We show that a simple greedy algorithm achieves the factor $O(\min\{\log R, \log n\})$, where R is the ratio between the longest and shortest segments in the instance.

Our main result (in section 4) is a $2t$-approximation algorithm for the MWIS in any t-interval graph, for $t \geq 2$, which is based on a novel *fractional* version of the Local Ratio technique. (The Local Ratio technique was first developed in [5] and later extended by [2, 4].) We use the fractional Local Ratio technique to round a fractional solution obtained from a linear programming relaxation of our problem. We expect that our nonstandard use of the Local Ratio technique will find more applications. Indeed, recently, this technique was used for obtaining improved bounds for the MWIS in the intersection graph of axis-parallel rectangles in the plane [32].

As we shall see, the MWIS in t-interval graphs properly includes the k-dimensional

matching problem. For this unweighted problem the best approximation factor known is $k/2+\epsilon$, for any $\epsilon > 0$ [26]. Hazan, Safra, and Schwartz [24] have recently shown that it is hard to approximate the k-dimensional matching problem within an $O(k/\log k)$ factor unless P = NP. Thus, our results are close to best possible.

For the class of t-union graphs, we develop (in section 5) a $(1, 1+\epsilon)$-approximation scheme with respect to the optimal profit and the latest completion time of any interval in some optimal solution. In particular, our scheme gets as parameter $T_{\mathcal{O}}$ the latest completion time of an interval in some optimal schedule and outputs a subset of intervals of optimal profit, in which the latest completion time of any interval is at most $T_{\mathcal{O}}(1+\epsilon)$.

1.2. Related work. We mention below several works that are related to ours.

Split interval graphs. Many NP-hard problems, including the MWIS problem [15, 16], can be solved efficiently in interval graphs. Split interval graphs have a long history in graph theory [43, 17, 39, 44], and more recently, union graphs have been studied under the name of *multitrack* interval graphs [30, 19, 29]. We mention some of the main results. For any fixed $t \geq 2$, determining whether a given graph is a t-interval (t-union) graph is NP-complete [44] ([19], respectively). 2-union graphs contain trees [43, 30] and more generally all outerplanar graphs [29], while 3-interval graphs contain the class of planar graphs [39]. Graphs of maximum degree Δ are $\lceil \frac{1}{2}(\Delta+1) \rceil$-interval graphs [17]. The complete bipartite graph, $K_{m,n}$, is a t-interval and t-union graph for $t = \lceil (mn+1)/(m+n) \rceil$ [43, 19].

Union graphs, which constitute a subfamily of split interval graphs, were also considered in several papers. Bafna, Narayanan, and Ravi [3] considered the problem of finding a weighted independent set in t-union graphs in the context of an application coming from computational biology. The union graphs considered in [3] are *proper*; i.e., there is no containment between segments. For the weighted independent set problem in proper t-union graphs, the paper [3] shows that the problem is NP-hard and gives a $(2^t - 1 + 1/2^t)$-approximation algorithm. This is obtained by mapping the problem to the MWIS in (2^t+1)-claw free graphs, noting that proper t-union graphs are (2^t+1)-claw free. Recently, Chlebík and Chlebíkova [11] showed that proper t-union graphs are $(2t+1)$-claw free. Using an algorithm of Berman [7] this gives a $(t+1/2)$-approximation of the MWIS in proper t-union graphs. Berman, DasGupta, and Muthukrishnan showed in [8] that a simple $O(n \log n)$ algorithm (based on the Local Ratio technique) yields a factor of 3 for proper 2-union graphs.

Coupled-task and flow shop scheduling. The problem of scheduling 2-intervals (known as *coupled-task scheduling*) was considered in the area of machine scheduling, with the objective of minimizing the overall completion time, or *makespan* (see, e.g., [35, 41]). Relaxed versions of the problem that require only a lower bound on the time that elapses between the schedules of the two tasks of each job (also called *time-lag problems*) were studied, e.g., in [37, 13, 10].

An instance of our problem can be viewed as an instance of the *flow shop* problem, in which the segments and break times are represented by *tasks* that need to be processed on a set of $m = 2t+1$ machines. (The precise transformation is given in section 5.) In general, the flow shop problem, where the objective is to minimize the makespan, is NP-complete even on three machines [14]. The best result known is the $O(\log^2(m\mu)/\log\log(m\mu))$-approximation algorithm, where μ is the maximum number of operations per job, and m is the number of machines [40, 42]. Hall [20] gave a polynomial time approximation scheme (PTAS) for this problem in the case where m is fixed (but arbitrary).

2. Preliminaries.

2.1. Definitions and notation. Given a t-interval graph, $G = (V, E)$, we assume that each vertex $v \in V$ is mapped to a set of at most t segments, and we call v a *split interval*. Suppose that segment I is one of the segments that vertex v is mapped to; then we say that I belongs to v and denote it by (v, I). We denote by $\mathcal{I}(\mathcal{G})$ the collection of segments (or intervals) on the real line, partitioned into disjoint *groups*, where each group is associated with a split interval. A t-interval graph is *proper* if no segment properly contains another segment.

In the subfamily of t-union graphs, the segments associated with each vertex can be labeled in such a way that for any two vertices u and v, the ith segment of u and the ℓth segment of v never intersect for $1 \leq i, \ell \leq t$, and $i \neq \ell$. Union graphs correspond also to certain geometric intersection graphs. The t segments are viewed as intervals on orthogonal axes, corresponding to a t-dimensional box; two boxes intersect if their projections on any of the t axes do. We further define subclasses of union graphs, where coordinates are all integral and segments are half-open. In (a, b)-union graphs, a subclass of 2-union graphs, all x-segments are of length a and all y-segments are of length b.

Given a graph $G = (V, E)$, we denote by $N(v)$ the set of neighbors of $v \in V$, and by $N[v]$ the closed neighborhood of v, $\{v\} \cup N(v)$. A $(k+1)$-*claw* is a graph consisting of a center vertex adjacent to $k+1$ mutually nonadjacent vertices. A graph is called $(k+1)$-*claw free* if it contains no k-claw as an induced subgraph.

Finally, we define our performance measures. Denote by OPT an optimal algorithm. The *approximation factor* of an algorithm $\mathcal{A}$ is r if for every finite input instance I, $\mathcal{A}(I)/OPT(I) \geq 1/r$, where $\mathcal{A}(I)$ and $OPT(I)$ are the values of $\mathcal{A}$ and OPT on I. A *polynomial time approximation scheme (PTAS)* is an algorithm which takes as input both the instance I and an error bound ϵ, has performance guarantee $R(I, \epsilon) \leq (1 + \epsilon)$, and runs in time polynomial in $|I|$. A (β, ϵ) *bicriteria PTAS* is a PTAS which is a β-approximation in one optimization criterion and a $(1 + \epsilon)$-approximation in the other criterion.

2.2. Hardness results. The independent set problem in interval graphs is easy to solve exactly, since interval graphs always contain a *simplicial* vertex, i.e., a vertex whose neighborhood is a clique. In fact, most approximation algorithms for independent sets on geometric intersection graphs are based on a related relaxed property: there always exists a vertex whose neighborhood does not contain a large independent set. We first show that for general t-interval graphs this property does not hold.

Observation 2.1. For any $n \geq 2$, there exists a 2-interval graph G on n vertices, in which every vertex has $\Omega(\sqrt{n})$ independent neighbors.

Proof. We show how to construct a 2-interval graph, in which every vertex has k independent neighbors. We construct the graph from $(k+1)$ sets of intervals; each set consists of k intervals, and each interval is composed of two segments: a *short* segment and a *long* segment. All the short segments in the input are of the same length, and likewise for all the long segments.

The graph is constructed as follows. For $\ell = 1, \ldots, k$, all the short segments of the intervals in the ℓth set, $I^\ell_1, \ldots, I^\ell_k$, intersect the long segments of the ℓth intervals in the sets $1, \ldots, k+1$, i.e., $I^1_\ell, \ldots, I^{k+1}_\ell$, excluding I^ℓ_ℓ. Finally, all the short segments of the intervals in the $(k+1)$th set intersect the long segment of the interval I^ℓ_ℓ, $1 \leq \ell \leq k$ (see Figure 2). Thus, we get that any interval I^ℓ_j with $\ell \neq j$ intersects k nonintersecting intervals in the jth set, and I^ℓ_ℓ intersects k nonintersecting intervals

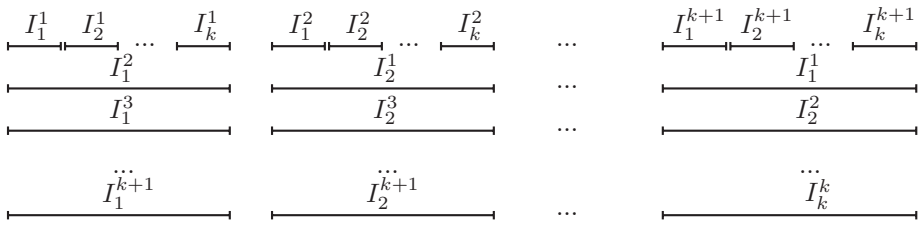


FIG. 2. *A 2-interval graph in which every vertex has k independent neighbors.*

in the $(k+1)$th set.

Note that since $k(k+1) \leq n$, we may have some remaining intervals, which are not contained in any set. We can place the segments of each such interval on the line such that its long segment intersects all the intervals in the ℓth set, for some $1 \leq \ell \leq k+1$, providing that interval with k independent neighbors. □

We note that the above construction can be modified to hold for 2-union graphs. We now give a structural result that implies hardness of approximation for a highly restricted class of proper 2-union graphs. A *degree-3 graph* is one of maximum degree 3.

THEOREM 2.2. *The class of* $(2,2)$*-union graphs includes the class of degree* 3.

Proof. A *linear forest* is a collection of disjoint paths. A path can be represented as a collection of length-2 half-closed intervals between integral endpoints, e.g., $[0,2), [1,3), [2,4)$, etc. Thus, a union of a pair of linear forests can be represented as a $(2,2)$-union graph. Akiyama, Exoo, and Harary [1] showed that degree-3 graphs can be represented as a union of two linear forests. Namely, they showed that for a degree-3 graph G, we have that $la(G) = 2$, where $la(G)$ denotes the *linear arboricity* of G or the minimum number of classes in a partition of $E(G)$ such that each class induces a linear forest. □

It follows that the MWIS problem is APX-hard on unweighted $(2,2)$-union graphs, since the (unweighted) MIS problem is APX-hard on degree-3 graphs (see [9, 23]). It also implies equivalent hardness results for other optimization problems that are hard to approximate on degree-3 graphs.

COROLLARY 2.3. *The MWIS problem is APX-hard on unweighted* $(2,2)$*-union graphs.*

Segments of unit size, whose start points are integral, are called *unit segments*. t-interval graphs of unit segments can be characterized precisely.

For some $k > 1$, let $S = \{1, 2, \ldots, n\}$, and let C be a collection of subsets of S, where each subset is of size at most k. The *k-set packing* problem is that of finding a maximum cardinality subcollection $C' \subseteq C$, such that the intersection of any two sets in C' is empty. In the *weighted* version, each subset has a weight, and we seek a subcollection C' of maximum weight.

LEMMA 2.4. *The k-set packing problem is equivalent to the MWIS in the special class of k-interval graphs of unit segments.*

Proof. There is a bijective mapping between unit segments and the set S, where $[i, i+1)$ maps to i for all values of i. Thus, there is a bijective mapping between sets of up to k elements from S and sets of up to k unit segments. □

A special case of k-set packing is the *k-dimensional matching* problem. Here, S is partitioned into subsets $S_1, S_2, \ldots, S_k$, and each set in C contains exactly one element from each S_i. The k-dimensional matching problem is similarly equivalent to

the MWIS in the special class of k-union graphs of unit segments. The former problem is NP-hard to approximate within factor $O(k/\log k)$ [24], while the best factor known is $k/2+\epsilon$ for any $\epsilon > 0$ [26]. We note that the 2-set packing problem is equivalent to the (polynomially solvable) edge cover problem, while 3-dimensional matching is APX-hard [36].

COROLLARY 2.5. *The MWIS problem in* $(1,1)$*-interval graphs is polynomial solvable. The MWIS problem in* $(1,1,1)$*-union graphs is APX-hard.*

The correspondence between $(1,1)$-union graphs to line graphs of bipartite graphs, and the resulting polynomial solvability of the MWIS problem, was shown by Halldórsson et al. [22].

3. Greedy algorithms.

3.1. Coloring t-interval graphs. For a t-interval graph G (see in Figure 1(a) for $t=2$), let G^* denote the graph formed by the intersection of the segments of the intervals (Figure 1(b)). The clique number, $\omega(G^*)$, denotes the maximum number of segments crossing a point on the real line.

THEOREM 3.1. *For any t-interval graph G, there is a vertex v in G such that*

$$d(v) \leq 2t(\omega(G^*)-1)-1.$$

Proof. Since each vertex in G corresponds to up to t vertices of G^*, $|V(G^*)| \leq t\cdot|V(G)|$, and since each edge in G corresponds to one or more edges in G^*, $|E(G)| \leq |E(G^*)|$. Since G^* is an interval graph, there is a simplicial ordering of the graph so that each vertex v_i has at most $\omega(G^*)-1$ neighbors among the vertices $v_{i+1},\ldots$. Thus, the number of edges in G^* is at most $(\omega(G^*)-1)|V(G^*)|$; in fact, it must be strictly less, since the last vertex has later neighbors. It follows that the average degree of G is bounded by

$$\overline{d}(G) = \frac{2|E(G)|}{|V(G)|} \leq 2t\frac{|E(G^*)|}{|V(G^*)|} < 2t(\omega(G^*)-1).$$

Hence, the minimum degree of G is at most $2t(\omega(G^*)-1)-1$. ∎

This leads to a simple coloring algorithm: find a vertex v satisfying the lemma, color the remaining graph $G\setminus v$, and finally color v with the smallest color not used by previously colored neighbors. This results in a $2t(\omega(G^*)-1)$-coloring.

The above gives a $2t$-approximation for coloring t-interval graphs via a greedy algorithm. Gyárfás [18] showed that the chromatic number of a t-interval graph G is at most $2t(\omega(G)-1)$, where $\omega(G)$ is the clique number of the graph.

COROLLARY 3.2. *A greedy algorithm colors G using $2t(\omega(G^*)-1)$ colors.*

Observe that this bound is obtained without knowledge of the underlying interval representation of G^*; this is important since deducing the representation is known to be NP-hard [44]. We show that this is about the best bound on $\chi(G)$ one can obtain in terms of $\omega(G^*)$, within a constant factor.

LEMMA 3.3. *For infinitely many t, there is a proper t-interval graph G such that $\omega(G) = (t-1)\omega(G^*)$.*

Proof. Let p be any prime number and $\mathbb{Z}_p$ be the finite field over $\{0,1,\ldots,p-1\}$. Let $t=p+1$. Let $C_{i,j}$ and D_i, $i,j\in\mathbb{Z}_p$, be any disjoint unit segments. We shall construct a system of t-intervals $I_{x,y}$, $x,y\in\mathbb{Z}_p$, and show that the t-intervals are pairwise overlapping, i.e., that any pair contains a common segment.

Let $I_{x,y} = \{C_{i,ix+y \bmod p} : i\in\mathbb{Z}_p\}\cup\{D_x\}$ for each $x,y\in\mathbb{Z}_p$. Clearly, the sets contain t segments each. Consider a pair of t-intervals $I_{x,y}$, $I_{x',y'}$. If $x=x'$, then both

t-intervals contain the segment $D_x = D_{x'}$. Otherwise, there exists an $i \in \mathbb{Z}_p$ that is a solution to the modular equation $i(x - x') \equiv (y' - y) (\mathrm{mod} p)$. Then, both t-intervals $I_{x,y}$ and $I_{x',y'}$ contain the segment $C_{i,ix+y \bmod p} = C_{i,ix'+y' \bmod p}$.

It follows that the intersection graph on these t-intervals is a clique on $p^2 = (t-1)^2$ vertices. On the other hand, each segment $C_{i,j}$ is contained in exactly $(t-1)$ t-intervals $I_{x,y}$ (namely, those for which $j = ix+y \bmod p$) and the same holds for each D_i. Thus, the clique number of G^* is $t - 1$. □

3.2. Greedy independent set algorithms. We study here a greedy algorithm for the special case where $t = 2$, in order to motivate the use of more complicated techniques in later sections.

Recall from Observation 2.1 that, in a 2-interval graph, the neighborhood of every vertex may include many independent vertices. Thus, purely greedy methods are bound to fail. Consider, for instance, the optimal greedy algorithm for independent sets in interval graphs that iteratively adds the interval with the leftmost right endpoint. An analogous method for 2-interval graphs could be to iteratively select the 2-interval with the leftmost right endpoint of the *first segment*, among all 2-intervals that do not intersect previously chosen 2-intervals. This algorithm, which we call Sort-and-Select, cannot be expected to perform well on all 2-interval graphs. However, it performs well under certain circumstances, which allows us to partition the instance into well-solvable subcases.

THEOREM 3.4. *Let G be a 2-interval graph where*

- *the first segment is no shorter than the second, and*
- *the ratio between the length of the shortest and longest second segment is at most* 4.

Then, the approximation factor of Sort-and-Select is 6.

Proof. Let I be the interval chosen first by Sort-and-Select. We claim that I intersects at most six independent intervals. Namely, the second segment of I is at most four times the length of the shortest segment in the graph; as a result, it intersects at most five independent segments/vertices. Also, since the first segment is furthest to the left of all segments in the graph, it does not intersect two independent vertices. Thus, among the intervals eliminated by the addition of I to the solution, the optimal solution can contain at most six. By induction, the algorithm then achieves an approximation factor of 6. □

Using the Local Ratio technique, which is discussed in depth in the next section, one can obtain the same factor for the weighted case. Also, by a similar argument, one can argue a factor of 3 for the case of proper 2-interval graphs.

Given a general 2-interval graph, we first divide the intervals into those where the first segment is shorter than the second segment and those where the first segment is at least as long as the second. This gives us two instances, which can be viewed as symmetric by reversing the direction of the real line. Thus, by increasing the approximation factor by a factor of 2, we can assume that in our instance the first segments are no shorter than the second segments.

We can partition the instance into $(\lg R)/2$ subinstances, or *buckets*, where R is the ratio between the longest to shortest (S) second segment. The bucket G_i consists of intervals with second segments in the range $[4^{i-1}S, 4^i S)$ for $i = 1, 2, \ldots, \lceil (\lg R)/2 \rceil$. Each bucket satisfies the conditions of Theorem 3.4; thus, the largest of the independent sets found in each bucket by Sort-and-Select is a $6 \lg R$ approximation.

Note that we can represent the n second segments in the input by $2n$ endpoints

on the line, and we define the length of each segment as the number of endpoints that lie between its left and right endpoints plus one. Then, the maximal possible length of a segment is $2n-1$, and the number of buckets is $B = \min\{\lg R, \lg(2n-1)\}$. Hence, we obtain the following result.

THEOREM 3.5. *There is a greedy partitioning algorithm for the maximum independent set in 2-interval graphs achieving a factor of* $O(\min\{\log R, \log 2n\})$.

4. A $2t$-approximation algorithm. We describe here a $2t$-approximation algorithm for the MWIS problem in a t-interval graph $G = (V, E)$. The algorithm is based on rounding a fractional solution derived from a linear programming relaxation of the problem. The standard linear programming relaxation of the MWIS problem is the following. For each $v \in V$, let $x(v)$ be the linear relaxation of the indicator variable for v, i.e., whether v belongs to the independent set. Let $\mathbf{w}, \mathbf{x} \in \mathbb{R}^{|V|}$ be a weight vector and a relaxed indicator vector, respectively.

maximize	$\mathbf{w} \cdot \mathbf{x}$	*subject to:*
for each clique $\mathcal{C} \in G$:	$\sum_{v \in \mathcal{C}} x(v) \leq 1$	

A feasible solution for the above linear program, whose value is an upper bound on the MWIS problem in the graph, can be obtained from the Lovász ϑ-function [33]. However, as we shall see, it is not necessary to optimize over all cliques in the case of t-interval graphs. We say that a clique $\mathcal{C}$ in the graph is an *interval clique* if for every vertex $v \in \mathcal{C}$, there is a segment $I \in v$ such that the intersection of $((v, I)|v \in \mathcal{C})$ is nonempty. It is easy to see that the interval cliques are defined by the set of right endpoints of the segments in $\mathcal{I}(\mathcal{G})$ as follows. Each right endpoint z corresponds to the clique defined by the vertices containing z. (See Figure 3 for an example.) Therefore, the number of interval cliques in a t-interval graph is linear in the number of segments.

We now further relax the MWIS problem and consider only interval cliques. In the integral case, let $x(v)$ denote the indicator variable of vertex v, and for each $I \in v$, $x(v, I) = x(v)$. In the linear relaxation (P), for each interval clique $\mathcal{C}$, we require that the sum of the variables $(v, I) \in \mathcal{C}$ is at most 1. It suffices to require, for each vertex v and $I \in v$, $x(v, I) \geq x(v)$, since only $x(v)$ participates in the objective function, and therefore in an optimal solution, without loss of generality, $x(v, I) = x(v)$.

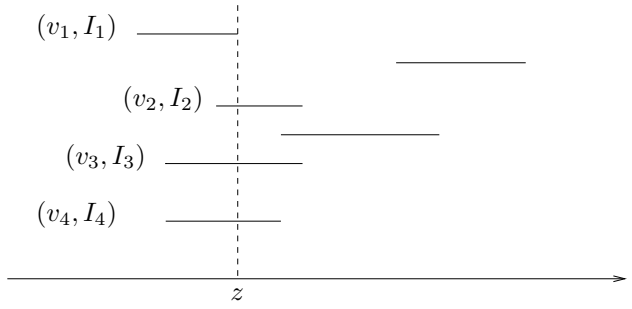


FIG. 3. *The interval clique* (v_1, v_2, v_3, v_4) *is defined by* z*, the right endpoint of* (v_1, I_1).

(P) maximize $\mathbf{w} \cdot \mathbf{x}$ *subject to:*			
for each interval clique $\mathcal{C}$:	$\sum_{(v,I)\in\mathcal{C}} x(v,I)$	$\leq$	1
for each $v \in V$ and $I \in v$:	$x(v,I) - x(v)$	$\geq$	0
for each $v \in V$ and $I \in v$:	$x(v), x(v,I)$	$\geq$	0

Since the number of interval cliques in a t-interval graph is linear in the number of segments, an optimal solution to (P) can be computed in polynomial time.

The heart of our rounding algorithm is the following lemma. It can be viewed as a fractional analogue of Theorem 3.1.

LEMMA 4.1. *Let* $\mathbf{x}$ *be a feasible solution to* (P). *Then, there exists a vertex* $v \in V$ *satisfying*

$$\sum_{u\in N[v]} x(u) \leq 2t.$$

Proof. For two adjacent vertices u and v, define $y(u,v) = x(v) \cdot x(u)$. Define $y(u,u) = x(u)^2$. For a segment I, let $R(I)$ be the interval clique defined by the right endpoint of I ($I \in R(I)$). We prove the claim using a *weighted* averaging argument, where the weights are the values $y(u,v)$ for all pairs of adjacent vertices, u and v.

Consider the sum $\sum_{v\in V}\sum_{u\in N[v]} y(u,v)$. An upper bound on this sum can be obtained as follows. For each $v \in V$, consider all segments $I \in v$, and for each (v,I), add up $y(u,v)$ for all (u,J) that intersect with (v,I) (including (v,I)). In fact, it suffices to add up $y(u,v)$ only for segments (u,J) such that $(u,J) \in R(I)$, and then multiply the total sum by 2. This suffices because of the following: (a) If, for segments (v,I) and (u,J), the right endpoint of I precedes the right endpoint of J, then (v,I) "sees" (u,J) and vice-versa. Since $y(u,v) = y(v,u)$, each of them contributes the same value to the other. (b) For segments (v,I) and (u,J), the constraints of (P) imply that $x(v,I) = x(v)$ and $x(u,J) = x(u)$. Hence, it follows from (a) and (b) that the mutual contribution of two segments (u,J) and (v,I) that intersect depends only on u and v; i.e., it is $y(u,v)$. Thus,

$$\sum_{v\in V}\sum_{u\in N[v]} y(u,v) \leq 2 \cdot \sum_{v\in V}\sum_{I\in v}\sum_{(u,J)\in R(I)} y(u,v).$$

Since

$$\sum_{(u,J)\in R(I)} y(u,v) \leq \; x(v) \cdot \sum_{(u,J)\in R(I)} x(u) \leq x(v),$$

we get that

$$\sum_{v\in V}\sum_{u\in N[v]} y(u,v) \leq 2t \cdot \sum_{v\in V} x(v).$$

Hence, there exists a vertex v satisfying

$$\sum_{u\in N[v]} y(u,v) \leq 2t \cdot x(v). \tag{1}$$

By factoring out $x(v)$ from both sides of inequality (1), the statement of the lemma is obtained. □

We now define a fractional version of the Local Ratio technique. The proof of the next lemma is immediate.

LEMMA 4.2. *Let* $\mathbf{x}$ *be a feasible solution to* (P). *Let* $\mathbf{w_1}$ *and* $\mathbf{w_2}$ *be a decomposition of the weight vector* $\mathbf{w}$ *such that* $\mathbf{w} = \mathbf{w_1} + \mathbf{w_2}$. *Let* $r > 0$. *Suppose that* $\mathbf{y}$ *is a feasible integral solution vector to* (P) *satisfying* $\mathbf{w_1} \cdot \mathbf{y} \geq r(\mathbf{w_1} \cdot \mathbf{x})$ *and* $\mathbf{w_2} \cdot \mathbf{y} \geq r(\mathbf{w_2} \cdot \mathbf{x})$. *Then,*

$$\mathbf{w} \cdot \mathbf{y} \geq r(\mathbf{w} \cdot \mathbf{x}).$$

The rounding algorithm will apply a Local Ratio decomposition of the weight vector $\mathbf{w}$ with respect to an optimal solution $\mathbf{x}$ to linear program (P). The algorithm proceeds as follows.

1. If no vertices remain, return the empty set. Otherwise, proceed to the next step.
2. Define $V_0 = \{v \in V | w(v) < 0\}$. If V_0 is nonempty, return $\mathcal{I}$, the recursive solution for $V \setminus V_0$. Otherwise, proceed to the next step.
3. Let $v' \in V$ be a vertex satisfying $\sum_{u \in N[v']} x(u) \leq 2t$. Decompose $\mathbf{w}$ by $\mathbf{w} = \mathbf{w_1} + \mathbf{w_2}$ as follows:

$$w_1(u) = \begin{cases} w(v') & \text{if } u \in N[v'], \\ 0 & \text{otherwise.} \end{cases}$$

   (In the decomposition, the component $\mathbf{w_2}$ may be nonpositive.)
4. Solve the problem recursively using $\mathbf{w_2}$ as the weight vector. Let $\mathcal{I}'$ be the independent set returned.
5. If $\mathcal{I}' \cup \{v'\}$ is an independent set, return $\mathcal{I} = \mathcal{I}' \cup \{v'\}$. Otherwise, return $\mathcal{I} = \mathcal{I}'$.

Clearly, the set $\mathcal{I}$ is an independent set. We now analyze the quality of the solution produced by the algorithm.

THEOREM 4.3. *Let* $\mathbf{x}$ *be an optimal solution to linear program* (P). *Then, it holds for the independent set* $\mathcal{I}$ *computed by the algorithm that* $w(\mathcal{I}) \geq \frac{1}{2t} \cdot \mathbf{w} \cdot \mathbf{x}$.

Proof. The proof is by induction on the number of vertices having positive weight. Any vertex deleted by the algorithm is considered to have zero weight. At the basis of the induction (Step 1), the inductive hypothesis holds, since the weight vector is considered to be zero. We now prove the inductive step.

In Step 2, if V_0 is nonempty, by the inductive hypothesis, $w(\mathcal{I}) \geq \frac{1}{2t} \cdot \mathbf{w} \cdot \mathbf{x}$. Extending $\mathbf{w}$ to include the nonpositive components that were deleted in Step 2 can only decrease the right-hand side, and therefore the inequality still holds.

In Steps 3–5, let $\mathbf{y}$ and $\mathbf{y}'$ be the indicator vectors of the sets $\mathcal{I}$ and $\mathcal{I}'$, respectively. By the decomposition in Step 3, weight vector $\mathbf{w_2}$ has fewer positive weight vertices than $\mathbf{w}$. Therefore, by the inductive hypothesis, $\mathbf{w_2} \cdot \mathbf{y}' \geq (1/2t) \cdot \mathbf{w_2} \cdot \mathbf{x}$. Since $w_2(v') = 0$, it also holds that $\mathbf{w_2} \cdot \mathbf{y} \geq (1/2t) \cdot \mathbf{w_2} \cdot \mathbf{x}$. From Step 5 of the algorithm it follows that at least one vertex from $N[v']$ belongs to $\mathcal{I}$. Hence, $\mathbf{w_1} \cdot \mathbf{y} \geq (1/2t) \cdot \mathbf{w_1} \cdot \mathbf{x}$. Thus, by Lemma 4.2, it follows that

$$\mathbf{w} \cdot \mathbf{y} \geq \frac{1}{2t} \cdot \mathbf{w} \cdot \mathbf{x}.$$

We have thus proved that $\mathcal{I}$ is a $2t$-approximate solution to the MWIS problem. □

We now outline an alternative way of using Lemma 4.1 to obtain the same approximation factor. Given an optimal solution $\mathbf{x}$ to linear program (P), a *multicoloring* of V by a set X is a mapping $\psi : V \to 2^X$ such that $|\psi(v)| = x(v)$ for each vertex v, and $\psi(v) \cap \psi(u) = \emptyset$ for each edge $(u, v) \in E(G)$. Since $\mathbf{x}$ is a feasible solution to (P), a repeated application of Lemma 4.1 results in a multicoloring with values in the closed interval $[0, 2t]$.

To view this as a multicoloring, it may be easier to discretize the instance within any desired precision by multiplying the $x(v)$'s by a sufficiently large integer L. Then the values assigned are positive integers in the range $1, \ldots, 2tL$. A continuous viewpoint is to assign each vertex a collection of contiguous segments; if we use Lemma 4.1 to assign the values one by one, we can always guarantee that a vertex v can be mapped to segments from $[0, 2t]$ of combined length $x(v)$ without overlapping any of the segments to which its neighbors are mapped to. In fact, by always mapping a vertex to the smallest available values, we need never use more than n disjoint segments for any vertex.

Let $0 = z_0 < z_1 < \cdots < z_{k-1}$ denote the values where the multicoloring changes, and let $z_k = 2t$. Thus, the coloring remains unchanged in the segment $[z_i, z_{i+1})$, $i = 0, \ldots, k-1$. Consider the sets $S_i = \{v \in V : x_i \in \psi(v)\}$ for $i = 0, \ldots, k-1$. Since ψ is a multicoloring, the S_i's are independent sets in G. Let $\mathcal{I}$ be the set S_i of maximum weight, $\sum_{v \in S_i} w(v)$.

THEOREM 4.4. *$w(\mathcal{I})$ is a $2t$-approximate independent set.*

Proof. Observe that the number of color values to which vertex v is mapped is $x(v)$, and we can represent them by $\sum_{S_i \ni v}(z_i - z_{i+1}) = x(v)$. We have that

$$\sum_{v \in V} w(v)x(v) = \sum_{v \in V} w(v) \sum_{S_i \ni v} (z_{i+1} - z_i) = \sum_{S_i} (z_{i+1} - z_i) \sum_{v \in S_i} w(v)$$
$$= \sum_{i=0}^{k-1} (z_{i+1} - z_i) w(S_i) \leq \sum_{i=0}^{k-1} (z_{i+1} - z_i) w(\mathcal{I}) = 2t w(\mathcal{I}). \qquad \square$$

5. A bicriteria approximation scheme for union graphs. Recall that the MWIS problem is APX-hard already on $(2, 2)$-union graphs. We consider below the larger subclass of t-union graphs in which the possible number of segment lengths is bounded by some constant. For this subclass we develop a *bicriteria* PTAS, which finds an MWIS by allowing some delays in the schedule.

Let c_i denote the number of distinct lengths of the ith segment, $1 \leq i \leq t$, where t is some constant. Recall that, in the flow shop problem, we are given a set of n jobs, $J_1, \ldots, J_n$, that need to be processed on m machines, $M_1, \ldots, M_m$; each job, J_j, consists of m operations, $O_{j,1}, \ldots, O_{j,m}$, where $O_{j,i}$ must be processed without interruptions on the machine M_i for $p_{j,i}$ time units. Any machine, M_i, can process either a *single* operation at a time or an *unbounded* number of operations; in the latter case we call M_i a *nonbottleneck* machine. Each job may be processed by at most one machine at any time. For a given schedule, let C_j be the completion time of J_j. The objective is to minimize the *maximum completion time* (or makespan), given by $C_{max} = \max_j C_j$. Denote by C^*_{max} the optimal makespan.

An instance of our problem can be transformed to an instance of the flow shop problem, where each job has $2t + 1$ operations, and the machines M_{2i+1}, $0 \leq i \leq t - 1$, are nonbottleneck machines. In our transformation, we apply some ideas from [27, 21, 28]. We represent each t-interval, I_j, as a job J_j, where each segment is associated with an "operation" of the job. In addition, we simulate the breaks with

operations of the same lengths that need to be processed on nonbottleneck machines. Similarly, to include the release time r_j of I_j, we add to J_j the operation $O_{j,1}$, whose length is equal to r_j; the machine M_1 is a nonbottleneck machine. Thus, if I_j has t segments, J_j has $2t$ operations.

Recall that, in a union graph, each interval has a due date, d_j, that is equal to its release time plus the sum of its processing times and break times. To simulate these due dates we define a *delivery time*, q_j, for each job, J_j. Let $q_j = -d_j$. We add to J_j the operation $O_{j,(2t+1)}$, where $p_{j,(2t+1)} = q_j$, and M_{2t+1} is a nonbottleneck machine. Our objective then is to minimize the maximum *delivery completion time*, given by $\max_j\{C_j + q_j\} = \max_j\{C_j - d_j\}$. This is equivalent to minimizing the maximum *lateness* of any job, given by $L_j = C_j - d_j$. Hence, our objective can be viewed as minimization of $L_{max} = \max_j L_j$.

Denote by $T_{\mathcal{O}}$ the maximum completion time of an optimal solution for the MWIS instance. Since we look for an MWIS that can be scheduled with maximum lateness at most $\epsilon T_{\mathcal{O}}$, we slightly modify the definition of *lateness*. Let $\tilde{d}_j = d_j - T_{\mathcal{O}}$; then, for any j, $\tilde{d}_j \leq 0$. By setting $q_j = -\tilde{d}_j$, we get that all the delivery times are positive. The maximum lateness is now given by $L_{max} = \max_j\{C_j - d_j + T_{\mathcal{O}}\}$. Indeed, for any job J_j, $C_j \geq d_j$; therefore, $L_{max} \geq T_{\mathcal{O}}$, and since in any optimal schedule there are no "late" jobs, the minimal lateness is $L^*_{max} = T_{\mathcal{O}}$.

Our scheme uses as procedure a PTAS for finding a $(1+\epsilon)$-approximation for the flow shop makespan problem with a fixed number of machines (see, e.g., [20]). We represent a t-interval I_j by a $(2t+1)$-vector $(p_{j,1}, \ldots, p_{j,2t+1})$, where $p_{j,1}$ is the release time, $p_{j,2i}$ ($p_{j,2i+1}$), is the length of the ith segment (break), $1 \leq i < t$, and $p_{j,2t+1}$ $(= q_j)$ is the delivery time of the corresponding job, J_j.

Below we summarize the steps of our scheme, which gets as parameters the value of $T_{\mathcal{O}}$ and some $\epsilon > 0$.

1. We scale the parameter values for J_j; that is, we divide the processing and release times by $T_{\mathcal{O}}$ and round each release time down and each break time up to the nearest multiple of ϵ.
2. We guess $\mathcal{O}$, the number of intervals scheduled by OPT.
3. We guess the subset $S_{\mathcal{O}}$ of $\mathcal{O}$ intervals of maximal weight, scheduled by OPT. This is done by guessing the set of vectors representing $S_{\mathcal{O}}$, among which we choose the subset of intervals of maximum weight.
4. Using a PTAS for minimizing the makespan in the flow shop instance of $S_{\mathcal{O}}$, we find a schedule of $S_{\mathcal{O}}$ for which $L_{max} \leq (1+\epsilon)L^*_{max}$.

Note that due to the above rounding, we need to add ϵ to the release times; also, each break time may delay the optimal completion time by ϵ. Therefore, by taking $\epsilon' = \epsilon/2t$ we guarantee that the delay of each interval is at most $(1+\epsilon)$ times $T_{\mathcal{O}}$. Finally, we set $L_j = L_j - T_{\mathcal{O}}$; thus, the maximum lateness of any job in our schedule is equal to at most $\epsilon T_{\mathcal{O}} = \epsilon C^*_{max}$.

For the complexity of the scheme, note that Steps 1 and 2 take linear time, and since the possible number of vectors $(p_{j,1}, \ldots p_{j,2t+1})$ is $(2t/\epsilon)^t \prod_{i=1}^{t} c_i$, we can guess $S_{\mathcal{O}}$ in $O(n^{(t^t \prod_{i=1}^{t} c_i)/\epsilon^t})$ steps. This is multiplied by the complexity of the PTAS for the flow shop.

THEOREM 5.1. *Let $t \geq 1$ be some fixed constant. Given a t-union graph with a constant number of distinct segment lengths, let $\mathcal{W}$ be the weight of an optimal MWIS, whose latest completion time is $T_{\mathcal{O}}$. Then, for any $\epsilon > 0$, there is a PTAS that schedules an independent set of weight at least $\mathcal{W}$, such that* any *interval is late by at most $\epsilon T_{\mathcal{O}}$.*

Acknowledgments. We thank Yossi Azar for many helpful comments on this paper. We also thank two anonymous referees for insightful comments and suggestions.

REFERENCES

[1] J. Akiyama, G. Exoo, and F. Harary, *Covering and packing in graphs* III: *Cyclic and acyclic invariants*, Math. Slovaca, 30 (1980), pp. 405–417.

[2] V. Bafna, P. Berman, and T. Fujito, *A* 2*-approximation algorithm for the undirected feedback vertex set problem*, SIAM J. Discrete Math., 12 (1999), pp. 289–297.

[3] V. Bafna, B. Narayanan, and R. Ravi, *Nonoverlapping local alignments (weighted independent sets of axis parallel rectangles)*, Discrete Appl. Math., 71 (1996), pp. 41–53.

[4] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B.Schieber, *A unified approach to approximating resource allocation and scheduling*, J. ACM, 48 (2001), pp. 1069–1090.

[5] R. Bar-Yehuda and S. Even, *A local ratio theorem for approximating the weighted vertex cover problem*, Ann. Discrete Math., 25 (1985), pp. 27–46.

[6] P. Basu, A. Narayanan, R. Krishnan, and T. D. C. Little, *An implementation of dynamic service aggregation for interactive video delivery*, in Proceedings of SPIE—Multimedia Computing and Networking, San Jose, CA, 1998, pp. 110–112.

[7] P. Berman, *A* $d/2$*-approximation for maximum weight independent set in* d*-claw free graphs*, Nordic J. Comput., 7 (2000), pp. 178–184.

[8] P. Berman, B. DasGupta, and S. Muthukrishnan, *Simple approximation algorithm for nonoverlapping local alignments*, in Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2002, pp. 677–678.

[9] P. Berman and T. Fujito, *Approximating independent sets in degree* 3 *graphs*, in Proceedings of the 4th Workshop on Algorithms and Data Structures (WADS'95), Lecture Notes in Comput. Sci. 955, Springer-Verlag, New York, 1995, pp. 449–460.

[10] P. Brucker, T. Hilbig, and J. Hurink, *A branch and bound algorithm for a single-machine scheduling problem with positive and negative time-lags*, Discrete Appl. Math., 94 (1999), pp. 77–99.

[11] M. Chlebík and J. Chlebíkova, *Approximation hardness of optimization problems in intersection graphs of* d*-dimensional boxes*, in Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2005, pp. 267–276.

[12] A. Dan, P. Shahabuddin, and D. Sitaram, *Channel Allocation under Batching and VCR Control in Movie-On-Demand Servers*, IBM Research Report RC19588, IBM, Yorktown Heights, NY, 1994.

[13] M. Dell'Amico, *Shop problems with two machines and time lags*, Oper. Res., 44 (1996), pp. 777–787.

[14] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA, 1979.

[15] F. Gavril, *Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph*, SIAM J. Comput., 1 (1972), pp. 180–187.

[16] M. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[17] J. R. Griggs and D. B. West, *Extremal values of the interval number of a graph*, SIAM J. Algebraic Discrete Methods, 1 (1980), pp. 1–7.

[18] A. Gyárfás, *On the chromatic number of multiple interval graphs and overlap graphs*, Discrete Math., 55 (1985), pp. 161–166.

[19] A. Gyárfás and D. B. West, *Multitrack interval graphs*, Congr. Numer., 109 (1995), pp. 109–116.

[20] L. A. Hall, *Approximability of flow shop scheduling*, Math. Programming, 82 (1998), pp. 175–190.

[21] L. A. Hall and D. B. Shmoys, *Approximation algorithms for constrained scheduling problems*, in Proceedings of the IEEE 30th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1989, pp. 134–139.

[22] M. M. Halldórsson, S. Rajagopalan, H. Shachnai, and A. Tomkins, *Scheduling Multiple Resources*, manuscript, 1999.

[23] M. M. Halldórsson and K. Yoshihara, *Approximation algorithms for maximum independent set problem on cubic graphs*, in Proceedings of International Symposium on Algorithms and Computation (ISAAC '95), Lecture Notes in Comput. Sci. 1004, Springer-Verlag, New York, 1995, pp. 152–161.

[24] E. Hazan, S. Safra, and O. Schwartz, *On the hardness of approximating k-dimensional matching*, in Proceedings of the 6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, Princeton University, Princeton, NJ, 2003, pp. 83–97.

[25] G. Hoyle, Distance Learning on the Net, http://www.hoyle.com.

[26] C. A. J. Hurkens and A. Schrijver, *On the size of systems of sets every t of which have an SDR, with an application to the worst-case ratio of heuristics for packing problems*, SIAM J. Discrete Math., 2 (1989), pp. 68–72.

[27] K. Jansen, R. Solis-Oba, and M. Sviridenko, *Makespan minimization in job shops: A polynomial time approximation scheme*, in Proceedings of the 31th Annual ACM Symposium on Theory of Computing, ACM, New York, 1999, pp. 394–399.

[28] D. Karger, C. Stein, and J. Wein, *Scheduling algorithms*, in Algorithms and Theory of Computation Handbook, CRC Press, Boca Raton, FL, 1998, Chapter 35.

[29] A. V. Kostochka and D. B. West, *Every outerplanar graph is the union of two interval graphs*, Congr. Numer., 139 (1999), pp. 5–8.

[30] N. Kumar and N. Deo, *Multidimensional interval graphs*, Congr. Numer., 102 (1994), pp. 45–56.

[31] M. Y. Y. Leung, C. S. Lui, and L. Golubchik, *Use of analytical performance models for system sizing and resource allocation in interactive video-on-demand systems employing data sharing techniques*, IEEE Trans. Knowl. Data Eng., 14 (2002), pp. 615–637.

[32] L. Lewin-Eytan, J. Naor, and A. Orda, *Routing and admission control in networks with advance reservations*, in Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX), Rome, Italy, 2002, pp. 215–228.

[33] L. Lovász, *On the Shannon capacity of a graph*, IEEE Trans. Inform. Theory, 25 (1979), pp. 1–7.

[34] C. Martin, P. S. Narayanan, B. Ozden, R. Rastogi, and A. Silberschatz, *The Fellini multimedia storage server*, in Multimedia Information Storage and Management, Kluwer Academic Publishers, Dordrecht, The Netharlands, 1996.

[35] A. J. Orman and C. N. Potts, *On the complexity of coupled-task scheduling*, Discrete Appl. Math., 72 (1997), pp. 141–154.

[36] C. H. Papadimitriou and M. Yannakakis, *Optimization, approximation, and complexity classes*, J. Comput. System Sci., 43 (1991), pp. 425–440.

[37] A. H. G. Rinnooy Kan, *Scheduling Problems*, Martinus Nijhoff, The Hague, 1976.

[38] D. Rotem, *Analysis of disk arm movement for large sequential reads*, in Proceedings of Principles of Database Systems (PODS), ACM, New York, 1992, pp. 47–54.

[39] E. R. Scheinerman and D. B. West, *The interval number of a planar graph—three intervals suffice*, J. Combin. Theory Ser. B, 35 (1983), pp. 224–239.

[40] J. P. Schmidt, A. Siegel, and A. Srinivasan, *Chernoff–Hoeffding bounds for applications with limited independence*, SIAM J. Discrete Math., 8 (1995), pp. 223–250.

[41] R. D. Shapiro, *Scheduling coupled tasks*, Naval Research Logistics Quarterly, 27 (1980), pp. 489–498.

[42] D. B. Shmoys, C. Stein, and J. Wein, *Improved approximation algorithms for shop scheduling problems*, SIAM J. Comput., 23 (1994), pp. 617–632.

[43] W. T. Trotter, Jr., and F. Harary, *On double and multiple interval graphs*, J. Graph Theory, 3 (1979), pp. 205–211.

[44] D. B. West and D. B. Shmoys, *Recognizing graphs with fixed interval number is NP-complete*, Discrete Applied Mathematics, 8 (1984), pp. 295–305.

[45] P. S. Yu, J. L. Wolf, and H. Shachnai, *Design and analysis of a look-ahead scheduling scheme to support pause-resume video-on-demand applications*, ACM Multimedia Systems Journal, 3 (1995), pp. 137–149.

SIAM J. COMPUT.
Vol. 36, No. 1, pp. 16–27

A SIMPLE ALGORITHM FOR MAL'TSEV CONSTRAINTS*

ANDREI BULATOV† AND VÍCTOR DALMAU‡

Abstract. A Mal'tsev operation is a ternary operation φ that satisfies the identities $\varphi(x,y,y) = \varphi(y,y,x) = x$. Constraint satisfaction problems involving constraints invariant under a Mal'tsev operation constitute an important class of constraint satisfaction problems, which includes the affine satisfiability problem, subgroup and near subgroup constraints, and many others. It is also known that any tractable case of the counting constraint satisfaction problem involves only Mal'tsev constraints.

The first algorithm solving the arbitrary constraint satisfaction problem with Mal'tsev constraints has been given by Bulatov. However, this algorithm is very sophisticated and relies heavily on advanced algebraic machinery. In this paper, we give a different and much simpler algorithm for this type of constraint.

Key words. constraint satisfaction, Mal'tsev

AMS subject classifications. 08A70, 68Q25, 69T99

DOI. 10.1137/050628957

1. Introduction. Constraint satisfaction problems arise in a wide variety of areas, such as combinatorics, logic, algebra, and artificial intelligence. An instance of the constraint satisfaction problem (CSP) consists of a set of variables, a set of values (which can be taken by the variables) that is called a domain, and a set of constraints (where a constraint is a pair given by a list of variables, called the constraint scope, and a relation indicating the valid combinations of values for the variables in the scope, called the constraint relation). The goal in a CSP is to decide whether or not there is an assignment of values to the variables satisfying all of the constraints. It is sometimes customary to cast the CSP as a relational homomorphism problem [15], namely, the problem of deciding, given a pair $(\mathbf{A}, \mathbf{B})$ of relational structures, whether or not there is a homomorphism from $\mathbf{A}$ to $\mathbf{B}$. In this formalization, each relation of $\mathbf{A}$ contains tuples of variables that are constrained together, and the corresponding relation of $\mathbf{B}$ contains the allowed tuples of values that the variable tuples may take.

The CSP is NP-complete in general, motivating the search for polynomial-time tractable cases of the CSP. A particularly useful way to restrict the CSP in order to obtain tractable cases is to restrict the types of constraints that may be expressed, by requiring the relations appearing in a constraint to belong to a given fixed set Γ. Such a restricted version of the CSP is denoted by CSP(Γ). This form of restriction can capture and place into a unified framework many particular cases of the CSP that have been independently investigated, for instance, the HORN SATISFIABILITY, 2-SATISFIABILITY, and GRAPH H-COLORABILITY problems. Schaefer was the first to consider the class of problems CSP(Γ); he proved a now famous dichotomy theorem,

*Received by the editors April 11, 2005; accepted for publication (in revised form) November 29, 2005; published electronically April 21, 2006.
http://www.siam.org/journals/sicomp/36-1/62895.html

†School of Computing Science, Simon Fraser University, 8888 University Drive, Burnaby V5A 1S6, BC, Canada (abulatov@cs.sfu.ca).

‡Departament de Tecnologia, Universitat Pompeu Fabra Estació de França, Passeig de la circumval.lació, Barcelona 08003, Spain (victor.dalmau@upf.edu). This author's research was partially supported by the MCyT under grants TIC 2002-04470-C03, TIC 2002-04019-C03, and TIN 2004-04343, the EU PASCAL Network of Excellence, IST-2002-506778, and the MODNET Marie Curie Research Training Network, MRTN-CT-2004-512234.

showing that for every set Γ of relations over a two-element domain, CSP(Γ) is either solvable in polynomial time or NP-complete [23].

In recent years, much effort has been invested toward the program of isolating all sets Γ of relations (or *constraint languages*) over a finite domain that give rise to a class of instances of CSP, CSP(Γ), solvable in polynomial time. Impressive progress has been made along these lines leading to the identification of several broad conditions on Γ that guarantee tractability. It was initiated by [2, 17, 19], where it has been shown that the complexity of CSP(Γ) depends only on the *algebraic invariance properties* of relations from Γ.

Some important recent achievements in the field include a complete classification of CSPs over a three-element domain [4] and the conservative CSP [6]. Remarkably, one of the main ingredients in both results is the recent result due to the first author [5] stating that every set Γ of relations on a finite set invariant with respect to a Mal'tsev operation, that is, a ternary operation φ satisfying $\varphi(x, y, y) = \varphi(y, y, x) = x$ for all x, y, gives rise to a tractable problem class. This result also encompasses and generalizes several previously known tractable cases of the CSP, such as affine problems [19, 23], constraint satisfaction problems on finite groups with near subgroups and their cosets [14, 15], and paraprimal CSPs [11].

Another reason Mal'tsev constraints are so important is that, to date, almost all CSPs known to be solvable in polynomial time can be solved using *local propagation algorithms*. In this type of algorithm we identify forbidden combinations of values for sets of variables of fixed size and then propagate the original problem by imposing new constraints that use this information. The only known exception is CSPs involving constraints invariant under a Mal'tsev operation. Thus, Mal'tsev constraints constitute a conceptually important class of constraints that require a completely different approach in solution techniques. Feder and Vardi attempted to capture this distinction in [15], where they used Datalog programs to simulate local propagation algorithms and suggested the term *problems with ability to count* for those problems which cannot be solved using Datalog. We also note another appearance of Mal'tsev constraints [8]. In that paper, we show that invariance with respect to a Mal'tsev operation is a necessary condition for the tractability of the counting CSP.

It is fair to say that the original proof of the tractability of Mal'tsev constraints [5] is very complicated. Furthermore, it makes intensive use of advanced algebraic techniques and thus is hardly comprehensible for a nonalgebraist. In this paper we give a different proof of the tractability of Mal'tsev constraints. The proof presented in this paper is notably simpler than the original proof and does not require the use of any previous algebraic result; indeed the proof is completely self-contained.

2. Preliminaries.

2.1. Constraint satisfaction problem. Let A be a finite set and let n be a positive integer. An n-ary relation on A is any subset of A^n. In what follows, for every positive integer n, $[n]$ will denote the set $\{1, \dots, n\}$.

A constraint satisfaction problem is a natural way to express simultaneous requirements for values of variables. This is stated more precisely in the following definition.

DEFINITION 2.1. *An instance* $\mathcal{P} = (V; A; \{C_1, \dots, C_m\})$ *of a* constraint satisfaction problem *consists of*

- *a finite set of variables,* $V = \{v_1, \dots, v_n\}$*;*
- *a finite domain of values,* A*;*

- *a finite set of constraints,* $\{C_1, \dots, C_m\}$*; each constraint* C_l*,* $l \in [m]$*, is a pair* $((v_{i_1}, \dots, v_{i_{k_l}}), S_l)$*, where*
  - $(v_{i_1}, \dots, v_{i_{k_l}})$ *is a tuple of variables of length* k_l*, called the* constraint scope*, and*
  - S_l *is a* k_l*-ary relation on* A*, called the* constraint relation.

A solution *to a CSP instance is a mapping* $s : V \to A$ *such that for each constraint* C_l*,* $l \in [m]$*, we have that* $(s(v_{i_1}), \dots, s(v_{k_l})) \in S_l$*.*

The question in the CSP instance $\mathcal{P}$ *is to decide whether or not there exists a solution to* $\mathcal{P}$*.*

The CSP is NP-complete in general, even when the constraints are restricted to binary constraints [21] or when the domain of values has size 2 [10].

Example 1. An instance of the standard propositional 3-SATISFIABILITY problem [16, 22] is specified by giving a formula of propositional logic consisting of a conjunction of clauses, each of which contains exactly three literals, and asking whether there are values for the variables which make the formula true.

Suppose that $\Phi = F_1 \wedge \dots \wedge F_n$ is such a formula, where the F_i are clauses. The satisfiability question for Φ can be expressed as the CSP instance $(V, \{0,1\}, \mathcal{C})$, where V is the set of all variables appearing in the clauses F_i and $\mathcal{C}$ is the set of constraints $\{(s_1, R_1), \dots, (s_n, R_n)\}$, where each constraint (s_k, R_k) is constructed as follows:

- $s_k = (x_1^k, x_2^k, x_3^k)$, where x_1^k, x_2^k, x_3^k are the variables appearing in clause F_k;
- $R_k = \{0,1\}^3 \setminus \{(a_1, a_2, a_3)\}$, where $a_i = 1$ if x_i^k is negated in F_k and $a_i = 0$ otherwise (i.e., R_k contains exactly those 3-tuples that make F_k true).

The solutions of this instance are exactly the assignments which make the formula Φ true.

Example 2. An instance of the GRAPH k-COLORABILITY problem consists of a graph G. The question is whether the vertices of G can be labeled with k colors so that adjacent vertices are assigned different colors.

This problem can be expressed as a CSP instance as follows. Let $G = (V, E)$ be a graph. Then we treat V as the set of variables (thus interpreting vertices of G as variables). For the domain we use a k-element set A of colors. Finally, for each edge $(u, v) \in E$ we introduce a constraint $((u, v), \neq_A)$, where $\neq_A$ is the disequality relation on A, defined by

$$\neq_A \ = \{(a, b) \in A^2 \mid a \neq b\}.$$

In applications of the CSP, we normally need just some restricted versions of the problem. One of the most natural and useful ways to restrict the CSP is to impose restrictions on the allowed constraint relations.

DEFINITION 2.2. *For any set of relations* Γ*,* CSP(Γ) *is defined to be the class of decision problems with*

- *Instance: A constraint satisfaction problem instance* $\mathcal{P}$*, in which all constraint relations are elements of* Γ*.*
- *Question: Does* $\mathcal{P}$ *have a solution?*

Example 1 (continued). If we define $\Gamma_{\text{3-SAT}}$ to be the constraint language on $\{0,1\}$ consisting of all relations expressible by 3-clauses, then any instance of 3-SATISFIABILITY can be expressed as an instance of CSP($\Gamma_{\text{3-SAT}}$) and vice versa. In other words, 3-SATISFIABILITY is equivalent to CSP($\Gamma_{\text{3-SAT}}$).

Example 2 (continued). Similarly, the GRAPH k-COLORABILITY can be viewed as CSP($\{\neq_A\}$).

In the two examples we have given, even the restricted problems are NP-complete. However, in many cases restricting the allowed form of constraint relations, we arrive at a problem solvable in polynomial time (we refer to such problems as *tractable* problems).

Example 3. An instance of Graph Unreachability consists of a graph, $G = (V, E)$, and a pair of vertices, $v, w \in V$. The question is whether there is no path in G from v to w.

This can be expressed as the CSP instance $(V, \{0,1\}, \mathcal{C})$, where

$$\mathcal{C} = \{(e, \{=_{\{0,1\}}\}) \mid e \in E\} \cup \{((v), \{(0)\}), ((w), \{(1)\})\},$$

where $=_{\{0,1\}}$ denotes the equality relation on the set $\{0,1\}$.

If we define Γ_{Unreach} to be the constraint language on $\{0,1\}$ containing just the relations $=_{\{0,1\}}$, $\{(0)\}$, and $\{(1)\}$, then any instance of Graph Unreachability can be expressed as an instance of $\mathrm{CSP}(\Gamma_{\text{Unreach}})$ in this way.

The research project, which this paper is a part of, aims to distinguish those constraint languages Γ which give rise to a tractable problem $\mathrm{CSP}(\Gamma)$ from those which do not.

In the CSP literature, it is usual to assume constraint languages to be finite. This is motivated by certain difficulties of representing infinite collections of relations, by the fact that applications of the CSP in discrete mathematics normally require only some fixed variety of relations, and also by some traditions well established in the area. The algebraic approach we introduce in the next section makes it very natural to deal with infinite constraint languages. Thus, in this paper, the constraint languages we consider are allowed to be infinite. This implies, in particular, that the arity of relations is not necessarily bounded.

Example 4. Let A be a finite field, and let Γ_{LIN} be the constraint language consisting of all relations over A which consist of all solutions to some linear equation over A. Any relation from Γ_{LIN}, and therefore any instance of $\mathrm{CSP}(\Gamma_{\text{LIN}})$, can be represented by a system of linear equations over A (for more details, see [3]).

Clearly, Γ_{LIN} is infinite and yet every instance of $\mathrm{CSP}(\Gamma_{\text{LIN}})$ can be easily defined and it can be solved in polynomial time (e.g., by Gaussian elimination).

2.2. Polymorphisms and invariants. We briefly introduce the basics of the algebraic approach to the CSP. For more details the reader is referred to [2].

Definition 2.3. *Let $\varphi : A^m \to A$ be an m-ary operation on A and let R be an n-ary relation over A. We say that R is* invariant *under φ and φ is said to be a* polymorphism *of R if for all (not necessarily different) tuples $\mathbf{t_1} = (t_1^1, \ldots, t_n^1), \ldots, \mathbf{t_m} = (t_1^m, \ldots, t_n^m)$ in R, the tuple $\varphi(\mathbf{t_1}, \ldots, \mathbf{t_m})$ defined as*

$$(\varphi(t_1^1, \ldots, t_1^m), \ldots, \varphi(t_n^1, \ldots, t_n^m))$$

belongs to R.

Let C be a set of operations on A and let Γ be a constraint language. Then $\mathrm{Inv}(C)$ denotes the set of all relations invariant under each operation from C, and $\mathrm{Pol}(\Gamma)$ denotes the set of all operations which are polymorphisms of every relation from Γ.

Example 5. Let R be the solution space of a system of linear equations over a field F. Then the operation $\varphi(x, y, z) = x - y + z$ is a polymorphism of R. Indeed, let $A \cdot \mathbf{x} = \mathbf{b}$ be the system defining R, and suppose that $\mathbf{x}, \mathbf{y}, \mathbf{z} \in R$. Then

$$A \cdot \varphi(\mathbf{x}, \mathbf{y}, \mathbf{z}) = A \cdot (\mathbf{x} - \mathbf{y} + \mathbf{z}) = A \cdot \mathbf{x} - A \cdot \mathbf{y} + A \cdot \mathbf{z} = \mathbf{b} - \mathbf{b} + \mathbf{b} = \mathbf{b}.$$

In fact, the converse can also be shown: If R is invariant under φ, then it is the solution space of a certain system of linear equations.

The cornerstone theorem proved by Jeavons [17] and Jeavons, Cohen, and Gyssens [19] provides a link between the complexity of CSP(Γ) and the properties of the polymorphisms of Γ. It amounts to saying that, for every *finite* constraint language Γ, the complexity of CSP(Γ) depends solely on the set Pol(Γ). In other words, if Pol(Γ_1) = Pol(Γ_2), then CSP(Γ_1) and CSP(Γ_2) are polynomial-time reducible to each other (provided Γ_1 and Γ_2 are finite). This also motivates the prior study of constraint languages of the form Inv(C) for some set C of operations. In fact, most of the existing tractability results on constraint languages may be formulated as the tractability of problems CSP(Inv(C)), where C consists of a single operation! (See, e.g., [1, 9, 12, 13, 17, 18, 20].)

One such type of operation that guarantees the tractability of the corresponding CSP is Mal'tsev operations.

DEFINITION 2.4. *A ternary operation $\varphi : A^3 \to A$ on a finite set A is called Mal'tsev if it satisfies the following identities:*

$$\varphi(x, y, y) = \varphi(y, y, x) = x \quad \forall x, y \in A.$$

For instance, operation φ defined in Example 5 is Mal'tsev.

The following theorem was first proved in [5].

THEOREM 2.5. *Let φ be a Mal'tsev operation. Then* CSP(Inv(φ)) *is solvable in polynomial time.*

The proof given in [5] employs a sophisticated algorithm that relies heavily on algebraic techniques and can hardly be understood without extensive knowledge of universal algebra. In this paper, we give a much shorter and simpler proof that is completely self-contained. Although the two algorithms have been designed mostly independently, they have a similar structure. To complete this section we outline the structure of the two algorithms and describe the main differences between them.

Both algorithms use *compact representations* of relations invariant under a Mal'tsev operation. Clearly, the size of an n-ary relation can be exponential in n. Therefore, if n is, for example, the number of variables in a CSP instance, then to represent an n-ary relation we need something more sophisticated than just a list of tuples. One possible form of such representation is introduced in the next section; representations used in [5] are very similar, but have a much finer structure. Both representations exploit the *rectangularity* of relations invariant under a Mal'tsev operation (see the next section).

Given a CSP instance $\mathcal{P} = (V, A, \{C_1, \dots, C_k\})$, both algorithms progressively construct compact representations of the complete sets of solutions of the instances $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_k = \mathcal{P}$, where $\mathcal{P}_i = (V, A, \{C_1, \dots, C_i\})$. In [5], this is done by constructing, in a certain complicated way, and then solving a system of linear equations such that a representation of $\mathcal{P}_{i+1}$ can be found as a basis of its solution space. Later, when working on [7], the second author observed that the same task can be fulfilled in a much more straightforward and simple way. This observation is the core of the present paper.

3. Signatures and representations. Let A be a finite set, let n be a positive integer, let $\mathbf{t} = (t_1, \dots, t_n)$ be an n-ary tuple, and let $i_1, \dots, i_j$ be (not necessarily

different) elements from $[n]$. By $\mathrm{pr}_{i_1,\dots,i_j}\,\mathbf{t}$ we denote the tuple $(t_{i_1},\dots,t_{i_j})$. Similarly, for every n-ary relation R on A and for every $i_1,\dots,i_j \in [n]$, we denote by $\mathrm{pr}_{i_1,\dots,i_j} R$ the j-ary relation given by $\{\mathrm{pr}_{i_1,\dots,i_j}\,\mathbf{t} : \mathbf{t} \in R\}$.

Given a relation R and an operation φ, we denote by $\langle R\rangle_\varphi$ the smallest relation R' that contains R and is invariant under φ. Very often, the operation φ will be clear from the context and we will drop it, writing $\langle R\rangle$ instead of $\langle R\rangle_\varphi$.

Let n be a positive integer, let A be a finite set, let $\mathbf{t},\mathbf{t}'$ be n-ary tuples, and let (i,a,b) be any element in $[n]\times A^2$. We say that $(\mathbf{t},\mathbf{t}')$ *witnesses* (i,a,b) if $\mathrm{pr}_{1,\dots,i-1}\,\mathbf{t} = \mathrm{pr}_{1,\dots,i-1}\,\mathbf{t}'$, $\mathrm{pr}_i\,\mathbf{t} = a$, and $\mathrm{pr}_i\,\mathbf{t}' = b$. We also say that $\mathbf{t}$ and $\mathbf{t}'$ witness (i,a,b), meaning that $(\mathbf{t},\mathbf{t}')$ witnesses (i,a,b).

Let R be any n-ary relation on A. We define the *signature* of R, $\mathrm{Sig}_R \subseteq [n]\times A^2$, as the set containing all those $(i,a,b)\in[n]\times A^2$ witnessed by tuples in R; that is

$$\mathrm{Sig}_R = \{(i,a,b)\in[n]\times A^2 : \exists \mathbf{t},\mathbf{t}'\in R \text{ such that } (\mathbf{t},\mathbf{t}') \text{ witnesses } (i,a,b)\}.$$

A subset R' of R is called a *representation* of R if $\mathrm{Sig}_{R'} = \mathrm{Sig}_R$. If, furthermore, $|R'| \le 2|\,\mathrm{Sig}_R\,|$, then R' is called a *compact* representation of R. Observe that every relation R has compact representations. Indeed, in order to construct such a compact representation R' we need only select, for each (i,a,b) in Sig_R, a couple of tuples $\mathbf{t},\mathbf{t}'$ in R that witness (i,a,b) and include them in R'.

Example 6. Fix a set A, an element $d\in A$, and an integer n. For every $(i,a)\in [n]\times A$ we define the tuple $\mathbf{e}^d_{i,a}$ as the only tuple satisfying

$$\mathrm{pr}_j\,\mathbf{e}^d_{i,a} = \begin{cases} a & \text{if } i=j \\ d & \text{otherwise} \end{cases} \quad \text{for all } j\in[n].$$

It is easy to observe that for every $(i,a,b)\in[n]\times A^2$, $(\mathbf{e}^d_{i,a},\mathbf{e}^d_{i,b})$ witnesses (i,a,b). Consequently, for a fixed d, the set of tuples $\{\mathbf{e}^d_{i,a} : i\in[n],\, a\in A\}$ is a representation of the relation A^n. Notice also that it is indeed a compact representation.

The algorithm we propose relies on Lemma 3.1, which follows from the property of *rectangularity* of relations invariant under a Mal'tsev operation φ. Let R be an n-ary relation invariant under φ and let $\mathbf{t_1},\mathbf{t_2},\mathbf{t_3}\in R$ be such that $\mathrm{pr}_{1,\dots,n-1}\,\mathbf{t_2} = \mathrm{pr}_{1,\dots,n-1}\,\mathbf{t_3}$ and $\mathrm{pr}_n\,\mathbf{t_1} = \mathrm{pr}_n\,\mathbf{t_2}$. Then the tuple $\mathbf{t}$ with $\mathrm{pr}_{1,\dots,n-1}\,\mathbf{t} = \mathrm{pr}_{1,\dots,n-1}\,\mathbf{t_1}$ and $\mathrm{pr}_n\,\mathbf{t} = \mathrm{pr}_n\,\mathbf{t_3}$ belongs to R. Indeed, we can choose $\mathbf{t} = \varphi(\mathbf{t_1},\mathbf{t_2},\mathbf{t_3})\in R$. Let us see that $\mathbf{t}$ satisfies the required conditions. Since φ is Mal'tsev we can infer that

$$\mathrm{pr}_{1,\dots,n-1}\,\mathbf{t} = \mathrm{pr}_{1,\dots,n-1}\,\varphi(\mathbf{t_1},\mathbf{t_2},\mathbf{t_3}) = \mathrm{pr}_{1,\dots,n-1}\,\varphi(\mathbf{t_1},\mathbf{t_2},\mathbf{t_2}) = \mathrm{pr}_{1,\dots,n-1}\,\mathbf{t_1}.$$

Also, we have that

$$\mathrm{pr}_n\,\mathbf{t} = \mathrm{pr}_n\,\varphi(\mathbf{t_1},\mathbf{t_2},\mathbf{t_3}) = \mathrm{pr}_n\,\varphi(\mathbf{t_2},\mathbf{t_2},\mathbf{t_3}) = \mathrm{pr}_n\,\mathbf{t_3}.$$

LEMMA 3.1. *Let A be a finite set, let $\varphi : A^3\to A$ be a Mal'tsev operation, let R be a relation on A invariant under φ, and let R' be a representation of R. Then $\langle R'\rangle_\varphi = R$.*

Proof. Let n be the arity of R. By induction on i, we shall show that, for every $i\in[n]$, $\mathrm{pr}_{1,\dots,i}\langle R'\rangle = \mathrm{pr}_{1,\dots,i} R$. The case $i=1$ follows easily from the fact that for each $\mathbf{t}\in R$, $(1,\mathrm{pr}_1\,\mathbf{t},\mathrm{pr}_1\,\mathbf{t})$ is in Sig_R and hence in $\mathrm{Sig}_{R'}$.

So, let us assume that the claim holds for $i<n$ and let $\mathbf{t}$ be any tuple in R. We show that $\mathrm{pr}_{1,\dots,i+1}\,\mathbf{t}\in \mathrm{pr}_{1,\dots,i+1}\langle R'\rangle$. By induction hypothesis there exists a tuple $\mathbf{t_1}$ in $\langle R'\rangle$ such that $\mathrm{pr}_{1,\dots,i}\,\mathbf{t_1} = \mathrm{pr}_{1,\dots,i}\,\mathbf{t}$. We have that $(i+1,\mathrm{pr}_{i+1}\,\mathbf{t_1},\mathrm{pr}_{i+1}\,\mathbf{t})$

belongs to Sig_R, and therefore, there exist some tuples $\mathbf{t_2}$ and $\mathbf{t_3}$ in R' witnessing it. To complete the proof we just need to observe that since $\mathbf{t_2}$ and $\mathbf{t_3}$ witness $(i+1, \mathrm{pr}_{i+1}\mathbf{t}, \mathrm{pr}_{i+1}\mathbf{t_1})$, we have that $\mathrm{pr}_{1,\dots,i}\mathbf{t_2} = \mathrm{pr}_{1,\dots,i}\mathbf{t_3}$ and that $\mathrm{pr}_{i+1}\mathbf{t_1} = \mathrm{pr}_{i+1}\mathbf{t_2}$ and $\mathrm{pr}_{i+1}\mathbf{t} = \mathrm{pr}_{i+1}\mathbf{t_3}$. Then the equality $\mathrm{pr}_{1,\dots,i+1}\mathbf{t} = \mathrm{pr}_{1,\dots,i+1}\varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3})$ follows from the rectangularity of $\mathrm{pr}_{1,\dots,i+1}R$. □

4. Proof of Theorem 2.5. We prove Theorem 2.5 by giving a polynomial-time algorithm that correctly decides whether a CSP(Inv(φ)) instance has a solution.

Let $\mathcal{P} = (\{v_1, \dots, v_n\}, A, \{C_1, \dots, C_m\})$ be a CSP(Inv(φ)) instance which will be the input of the algorithm.

For each $l \in \{0, \dots, m\}$ we define $\mathcal{P}_l$ as the CSP instance that contains the first l constraints of $\mathcal{P}$, that is, $\mathcal{P}_l = (\{v_1, \dots, v_n\}, A, \{C_1, \dots, C_l\})$. Furthermore, we shall denote by R_l the n-ary relation on A defined as

$$R_l = \{(s(v_1), \dots, s(v_n)) : s \text{ is a solution of } \mathcal{P}_l\}.$$

As we have already mentioned, in a nutshell, the algorithm introduced in this section computes for each $l \in \{0, \dots, m\}$ a compact representation R'_l of R_l. In the initial case ($l = 0$), $\mathcal{P}_0$ does not have any constraint at all and, consequently, $R_0 = A^n$. Hence, a compact representation of R_0 can be easily obtained as in Example 6. Once a compact representation R'_0 of R_0 has been obtained, then the algorithm starts an iterative process in which a compact representation R'_{l+1} of R_{l+1} is obtained from R'_l and the constraint C_{l+1}. This is achieved by calling Procedure `Next`, which constitutes the core of the algorithm. The algorithm then goes as follows:

Algorithm `Solve`$((\{v_1, \dots, v_n\}, A, \{C_1, \dots, C_m\}))$
Step 1 **select** an arbitrary element d in A
Step 2 **set** $R'_0 := \{\mathbf{e}^d_{i,a} : (i, a) \in [n] \times A\}$
Step 3 **for each** $l \in \{0, \dots, m-1\}$ **do**
(let C_{l+1} be $((v_{i_1}, \dots, v_{i_{l+1}}), S_{l+1})$)
Step 3.1 **set** $R'_{l+1} :=$ `Next`$(R'_l, i_1, \dots, i_{l+1}, S_{l+1})$
endfor
Step 4 **if** $R'_m \neq \emptyset$ **return yes**
Step 5 **otherwise return no**

Observe that if we modify Step 4 so that the algorithm returns an arbitrary tuple in R'_m instead of "yes," then we have an algorithm that does not merely solve the decision question but actually provides a solution.

Correctness and polynomial-time complexity of the algorithm are direct consequences of the correctness and the running time of Procedure `Next`: As shown in section 4.3 (Lemma 4.1), at each iteration of Step 3.1, the output of the call `Next`$(R'_l, i_1, \dots, i_{l+1}, S_{l+1})$ is a compact representation of the relation $\{\mathbf{t} \in R_l : \mathrm{pr}_{i_1,\dots,i_{l+1}}\mathbf{t} \in S_{l+1}\}$, which is indeed R_{l+1}. Furthermore, the time complexity of `Solve` is obviously polynomial in the time complexity of `Next`. Thus to finish the proof of Theorem 2.5, we need to design `Next` and show that it is correct and has polynomial time complexity.

The remainder of the paper is devoted to defining and analyzing Procedure `Next`. In order to define Procedure `Next` it is convenient to introduce first two simple procedures, namely `Nonempty` and `Fix-values`, which will be intensively used by our Procedure `Next`.

4.1. Procedure `Nonempty`. This procedure receives as an input a compact representation R' of a relation R invariant under φ, a sequence $i_1, \dots, i_j$ of elements in

$[n]$, where n is the arity of R, and a j-ary relation S also invariant under φ. The output of the procedure is either an n-ary tuple $\mathbf{t} \in R$ such that $\mathrm{pr}_{i_1,\ldots,i_j}\,\mathbf{t} \in S$ or "no," meaning that such a tuple does not exist.

Procedure `Nonempty`$(R', i_1, \ldots, i_j, S)$
Step 1 **set** $U := R'$
Step 2 **while** $\exists \mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3} \in U$ such that $\mathrm{pr}_{i_1,\ldots,i_j}\,\varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}) \notin \mathrm{pr}_{i_1,\ldots,i_j}\,U$ **do**
Step 2.1 **set** $U := U \cup \{\varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3})\}$
endwhile
Step 3 **if** $\exists \mathbf{t}$ in U such that $\mathrm{pr}_{i_1,\ldots,i_j}\,\mathbf{t} \in S$ **then return t**
Step 4 **else return** "no"

We shall start by studying the procedure's correctness. First observe that every tuple in U belongs initially to R' (and hence to R) or it has been obtained by applying φ to some tuples $\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}$ that previously belonged to U. Therefore, since R is invariant under φ, we can conclude that $U \subseteq R$. Consequently, if a tuple $\mathbf{t}$ is returned in Step 3, then it belongs to R and also satisfies the condition $\mathrm{pr}_{i_1,\ldots,i_j}\,\mathbf{t} \in S$, as desired. It remains only to show that if a "no" is returned in Step 4, then there exists no tuple $\mathbf{t}$ in R such that $\mathrm{pr}_{i_1,\ldots,i_j}\,\mathbf{t} \in S$. In order to do this we need to show some simple facts about U. Notice that at any point of the execution of the procedure $R' \subseteq U$. Then U is also a representation of R and hence $\langle U \rangle = R$. Therefore we have that

$$\langle \mathrm{pr}_{i_1,\ldots,i_j}\,U \rangle = \mathrm{pr}_{i_1,\ldots,i_j} \langle U \rangle = \mathrm{pr}_{i_1,\ldots,i_j}\,R.$$

By the condition on the "while" of Step 2 we have that when the procedure leaves the execution of Step 2 it must be the case that for all $\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3} \in U$, $\mathrm{pr}_{i_1,\ldots,i_j}\,\varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}) \in \mathrm{pr}_{i_1,\ldots,i_j}\,U$ and, consequently, $\mathrm{pr}_{i_1,\ldots,i_j}\,U = \langle \mathrm{pr}_{i_1,\ldots,i_j}\,U \rangle = \mathrm{pr}_{i_1,\ldots,i_j}\,R$. Hence, if there exists some $\mathbf{t}$ in R such that $\mathrm{pr}_{i_1,\ldots,i_j} \in S$, then there must exist some $\mathbf{t'}$ in U such that $\mathrm{pr}_{i_1,\ldots,i_j} \in S$, and we are done.

Let us study now the running time of the procedure. It is only necessary to focus on Steps 2 and 3. At each iteration of the loop in Step 2, cardinality of U increases by one. So we can bound the number of iterations by the size $|U|$ of U at the end of the execution of the procedure.

The cost of each iteration is basically dominated by the cost of checking whether there exist some tuples $\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3} \in U$ such that $\mathrm{pr}_{i_1,\ldots,i_j}\,\varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}) \notin \mathrm{pr}_{i_1,\ldots,i_j}\,U$ which is done in Step 2. In order to try all possible combinations for $\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}$ in U, $|U|^3$ steps suffice. Each one of these steps requires time $O(|U|n)$, as tuples have arity n and checking whether $\varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3})$ belongs to U can be done naively by a sequential search in U. Thus, the running time of `Nonempty` is polynomial in n and in the possible size of U.

Notice now that this is not enough. Indeed, as we do not restrict the arity of the relations used, j may be comparable with n, which implies that the size of U may be exponential in n. To solve this problem, we organize invoking `Nonempty` in such a way that the size of U is bounded by a polynomial in $|S|$, n, and $|A|$. In order to accomplish this, we shall ensure that at each call to `Nonempty` the following condition is satisfied:

$$|\,\mathrm{pr}_{i_1,\ldots,i_j}\,R| \leq |S| \cdot |A|^2. \tag{4.1}$$

Later (see Procedures `Next` and `Next-beta`) we show how this can be achieved. It is not difficult to see that if (4.1) is true, then the size of U can be bounded by a

polynomial. More precisely, the size of U can be bounded by the initial size of R', which is at most $n \cdot |A|^2$ (since R' is compact) plus the number of iterations of Step 2, which is bounded by $|\operatorname{pr}_{i_1,\dots,i_j} R| \leq |S| \cdot |A|^2$.

4.2. Procedure `Fix-values`. This procedure receives as an input a compact representation R' of a relation R invariant under φ and a sequence $a_1, \dots, a_m$, $m \leq n$, of elements of A (n is the arity of R). The output is a compact representation of the relation given by

$$\{\mathbf{t} \in R : \operatorname{pr}_1 \mathbf{t} = a_1, \dots, \operatorname{pr}_m \mathbf{t} = a_m\}.$$

Procedure `Fix-values`$(R', a_1, \dots, a_m)$
Step 1 **set** $j := 0$; $U_j := R'$
Step 2 **while** $j < m$ **do**
Step 2.1 **set** $U_{j+1} := \emptyset$
Step 2.2 **for each** $(i, a, b) \in [n] \times A^2$ **do**
Step 2.2.1 **if** $\exists \mathbf{t_2}, \mathbf{t_3} \in U_j$ witnessing (i, a, b)
(we assume that if $a = b$ then $\mathbf{t_2} = \mathbf{t_3}$) and
`Nonempty`$(U_j, j+1, i, \{(a_{j+1}, a)\}) \neq$ "no" and
$(i > j+1$ or $a = b = a_i)$ **then**
(let $\mathbf{t_1}$ be the tuple returned
by the call to `Nonempty`$(U_j, j+1, i, \{a_{j+1}, a\})$)
set $U_{j+1} := U_{j+1} \cup \{\mathbf{t_1}, \varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3})\}$
endfor
Step 2.4 **set** $j := j + 1$
endwhile
Step 3 **return** U_m

Let us study the correctness of the procedure. We shall show by induction on $j \in \{0, \dots, m\}$ that U_j is a compact representation of $R_j = \{\mathbf{t} \in R : \operatorname{pr}_1 \mathbf{t} = a_1, \dots, \operatorname{pr}_j \mathbf{t} = a_j\}$. The case $j = 0$ is correctly settled in Step 1. Hence it is only necessary to show that at every iteration of the **while** loop in Step 2, if U_j is a compact representation of R_j, then U_{j+1} is a compact representation of R_{j+1}. It is easy to see that if any of the conditions of the **if** in Step 2.2.1 is falsified, then (i, a, b) is not in $\operatorname{Sig}_{R_{j+1}}$. So it remains only to see that when the **if** in Step 2.2.1 is satisfied, we have that (a) $\mathbf{t_1}$ and $\varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3})$ are tuples in R_{j+1}, and (b) $(\mathbf{t_1}, \varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}))$ witnesses (i, a, b).

Proof of (a). As $\mathbf{t_1} =$ `Nonempty`$(U_j, j+1, i, \{(a_{j+1}, a)\})$, we can conclude that $\mathbf{t_1}$ belongs to R_j, $\operatorname{pr}_{j+1} \mathbf{t_1} = a_{j+1}$, and $\operatorname{pr}_i \mathbf{t_1} = a$. Consequently $\mathbf{t_1}$ belongs to R_{j+1}. Furthermore, as $\mathbf{t_1}$, $\mathbf{t_2}$, and $\mathbf{t_3}$ are in R_j and R_j is invariant under φ, $\varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3})$ belongs to R_j. Let us see now that $\operatorname{pr}_{j+1} \mathbf{t_2} = \operatorname{pr}_{j+1} \mathbf{t_3}$ by means of a case analysis. If $i > j+1$, then we have that $\operatorname{pr}_{j+1} \mathbf{t_2} = \operatorname{pr}_{j+1} \mathbf{t_3}$ as $(\mathbf{t_2}, \mathbf{t_3})$ witnesses (i, a, b). If $i \leq j+1$, then $a = b = a_i$ and hence $\mathbf{t_2}$ and $\mathbf{t_3}$ are identical.

Finally, since φ is Mal'tsev, $\operatorname{pr}_{j+1} \varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}) = \operatorname{pr}_{j+1} \mathbf{t_1} = a_{j+1}$ and hence $\varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3})$ belongs to R_{j+1}. □

Proof of (b). Since $(\mathbf{t_2}, \mathbf{t_3})$ witnesses (i, a, b), we have that $\operatorname{pr}_{1,\dots,i-1} \mathbf{t_2} = \operatorname{pr}_{1,\dots,i-1} \mathbf{t_3}$. Consequently, $\operatorname{pr}_{1,\dots,i-1} \varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}) = \operatorname{pr}_{1,\dots,i-1} \mathbf{t_1}$. Furthermore, we also have that $\operatorname{pr}_i \varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}) = \operatorname{pr}_i \varphi(a, a, b) = b$.

Notice that each iteration adds at most two tuples for some (i, a, b) in $\operatorname{Sig}_{R_{j+1}}$. Consequently, U_{j+1} is compact. This completes the proof of its correctness. □

Let us study now the time complexity of `Fix-values`. The **while** loop at Step 2 is performed $m \leq n$ times. At each iteration the procedure executes another loop

(Step 2.2). The **for each** loop at Step 2.2 is executed for each (i, a, b) in $[n] \times A^2$, that is, a total number of $n|A|^2$ times. The cost of each iteration of the loop is basically dominated by the cost of the call to Procedure `Nonempty`. Therefore, the time complexity of the procedure is polynomial in n and the maximal time complexity of `Nonempty`. Observe that in this case condition (4.1) in the call of `Nonempty` is satisfied.

4.3. Procedure `Next`. We are now almost in a position to introduce Procedure `Next`. Procedure `Next` receives as input a compact representation R' of a relation R invariant under φ, a sequence $i_1, \dots, i_j$ of elements in $[n]$ where n is the arity of R, and a j-ary relation S invariant under φ. The output of `Next` is a compact representation of the relation $R^* = \{\mathbf{t} \in R : \mathrm{pr}_{i_1,\dots,i_j}\, \mathbf{t} \in S\}$. It is an easy exercise to verify that R^* is also invariant under φ.

We shall start by defining a procedure called `Next-beta` that is equivalent to `Next` but has a worse time complexity running time. In particular, the running time of `Next-beta` might be exponential in the size of its input.

Procedure `Next-beta`$(R', i_1, \dots, i_j, S)$
Step 1 **set** $U := \emptyset$
Step 2 **for each** $(i, a, b) \in [n] \times A^2$ **do**
Step 2.1 **if** `Nonempty`$(R', i_1, \dots, i_j, i, S \times \{a\}) \neq$ "no" **then**
(let $\mathbf{t}$ be the tuple returned by `Nonempty`$(R', i_1, \dots, i_j, i, S \times \{a\})$)
Step 2.2 **if** `Nonempty`(`Fix-values`$(R', \mathrm{pr}_1\, \mathbf{t}, \dots, \mathrm{pr}_{i-1}\, \mathbf{t})$, $i_1, \dots, i_j, i, S \times \{b\}) \neq$ "no"
(let $\mathbf{t}'$ be the tuple returned by `Nonempty`(`Fix-values`$(R', \mathrm{pr}_1\, \mathbf{t}, \dots, \mathrm{pr}_{i-1}\, \mathbf{t}), i_1, \dots, i_j, i, S \times \{b\})$)
set $U := U \cup \{\mathbf{t}, \mathbf{t}'\}$
endfor
Step 3 **return** U

The overall structure of Procedure `Next-beta` is similar to that of Procedure `Fix-values`. Observe that the condition of the **if** statement

$$\texttt{Nonempty}(R', i_1, \dots, i_j, i, S \times \{a\}) \neq \text{“no”}$$

of Step 2.1 is satisfied if and only if there exists a tuple $\mathbf{t} \in R$ such that $\mathrm{pr}_{i_1,\dots,i_j}\, \mathbf{t} \in S$ and $\mathrm{pr}_i\, \mathbf{t} = a$. Hence if such a tuple does not exist, then (i, a, b) is not in Sig_{R^*} and nothing needs to be done for (i, a, b). Now consider the condition of the **if** statement in Step 2.2 which is given by

$$\texttt{Nonempty}(\texttt{Fix-values}(R', \mathrm{pr}_1\, \mathbf{t}, \dots, \mathrm{pr}_{i-1}\, \mathbf{t}), i_1, \dots, i_j, i, S \times \{b\}) \neq \text{“no”}.$$

This condition is satisfied if and only if there exists some $\mathbf{t}'$ in R such that $\mathrm{pr}_{i_1,\dots,i_j}\, \mathbf{t}' \in S$ such that $\mathrm{pr}_{1,\dots,i-1}\, \mathbf{t}' = \mathrm{pr}_{1,\dots,i-1}\, \mathbf{t}$ and $\mathrm{pr}_i\, \mathbf{t}' = b$. It is immediate to see that if the condition holds, then $(\mathbf{t}, \mathbf{t}')$ witnesses (i, a, b). It only remains to show that if $(i, a, b) \in \mathrm{Sig}_{R^*}$, then such a $\mathbf{t}'$ must exist. In order to do this, it is necessary only to verify that if $\mathbf{t_a}, \mathbf{t_b}$ are tuples in R^* witnessing (i, a, b), then, since φ is Mal'tsev, the tuple $\varphi(\mathbf{t}, \mathbf{t_a}, \mathbf{t_b})$ satisfies the desired properties (here $\mathbf{t}$ is the tuple returned by the call to Procedure `Nonempty` in Step 2.1).

Again, the cardinality of U is bounded by $2|\,\mathrm{Sig}_{R^*}\,|$, and hence, U is a compact representation.

Let us study the running time of Procedure `Next-beta`. The loop of Step 2 is performed $n|A|^2$ times and the cost of each iteration is basically the cost of Steps

2.1 and 2.2 in which other procedures are called. Therefore, the running time of `Next-beta` is polynomial in n and the running time of `Nonempty` and `Fix-values`. However, when invoking `Nonempty` on Steps 2.1 and 2.2, we cannot be sure that condition (4.1) holds, and therefore the running time of `Nonempty` and consequently `Next-beta` may be exponential in the size of the input. To avoid this problem, we define a new procedure, `Next`, which makes a sequence of calls to `Next-beta` such that the following condition is satisfied:

$$|\operatorname{pr}_{i_1,\dots,i_j} R| \le |S| \cdot |A|. \tag{4.2}$$

It is easy to observe that if condition (4.2) in the call of `Next-beta` is guaranteed, then condition (4.1) in every call of `Nonempty` is also satisfied. Consequently, if `Next-beta` is called with parameters satisfying condition (4.2), then its running time is polynomial on n, S, and $|A|$.

Procedure `Next`$(R', i_1, \dots, i_j, S)$
Step 1 **set** $l := 0$, $U_l := R'$
Step 2 **while** $l < j$ **do**
Step 2.1 **set** $U_{l+1} :=$ `Next-beta`$(U_l, i_1, \dots, i_{l+1}, \operatorname{pr}_{i_1,\dots,i_{l+1}} S)$
end while
Step 3 **return** U_j

Let us show that condition (4.2) is satisfied in any call to Procedure `Next-beta` in Step 2.1. In the first iteration ($l = 0$), condition (4.2) holds as $\operatorname{pr}_{i_1} R' \subseteq A$. For every subsequent iteration $l \in \{1, \dots, j-1\}$ observe that at the beginning of the iteration,

$$\operatorname{pr}_{i_1,\dots,i_l}\langle U_l\rangle = \operatorname{pr}_{i_1,\dots,i_l} S.$$

Consequently, at each call to Procedure `Next-beta` we have

$$|\operatorname{pr}_{i_1,\dots,i_{l+1}}\langle U_l\rangle| \le |\operatorname{pr}_{i_1,\dots,i_l} S||A| \le |S| \cdot |A|.$$

Therefore, condition (4.2) holds and the running time of the call is polynomial in n, $|S|$, and $|A|$.

Therefore, we have just proved the following lemma.

LEMMA 4.1. *For every $n \ge 1$, every n-ary relation R invariant under φ, every compact representation R' of R, every $i_1, \dots, i_j \in [n]$, and every j-ary relation S invariant under φ,* `Next`$(R', i_1, \dots, i_j, S)$ *computes a compact representation of $R^* = \{\mathbf{t} \in R : \operatorname{pr}_{i_1,\dots,i_j} \in S\}$ in time polynomial in n, $|S|$, and $|A|$. Furthermore, R^* is invariant under φ.*

COROLLARY 4.2. *Algorithm* `Solve` *decides correctly if an arbitrary instance $\mathcal{P}$ of* $\mathrm{CSP}(\mathrm{Inv}(\varphi))$ *is satisfiable in time polynomial in n, s, and $|A|$, where n is the number of variables of $\mathcal{P}$, s is the total number of tuples in the constraint relations, and A is the domain.*

REFERENCES

[1] A. A. BULATOV AND P. G. JEAVONS, *Algebraic Structures in Combinatorial Problems*, Tech. Report MATH-AL-4-2001, Technische Universität Dresden, Dresden, Germany, 2001.

[2] A. A. BULATOV, A. A. KROKHIN, AND P. G. JEAVONS, *Constraint satisfaction problems and finite algebras*, in Automata, Languages and Programming (ICALP'00), Lecture Notes in Comput. Sci. 1853, Springer-Verlag, Berlin, 2000, pp. 272–282.

[3] A. BULATOV, P. JEAVONS, AND A. KROKHIN, *Classifying the complexity of constraints using finite algebras*, SIAM J. Comput., 34 (2005), pp. 720–742.

[4] A. A. Bulatov, *A dichotomy theorem for constraints on a three-element set*, in Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS'02), 2002, pp. 649–658.

[5] A. A. Bulatov, *Mal'tsev Constraints Are Tractable*, Tech. Report PRG-02-05, Computing Laboratory, Oxford University, Oxford, UK, 2002.

[6] A. A. Bulatov, *Tractable conservative constraint satisfaction problems*, in Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS'03), 2003, pp. 321–330.

[7] A. A. Bulatov, H. Chen, and V. Dalmau, *Learnability of relatively quantified generalized formulas*, in Algorithmic Learning Theory (ALT '04), Lecture Notes in Comput. Sci. 3244, Springer-Verlag, Berlin, 2004, pp. 365–379.

[8] A. A. Bulatov and V. Dalmau, *Towards a dichotomy theorem for the counting constraint satisfaction problem*, in Proceedings of the 44th IEEE Symposium on Foundations of Computer Science (FOCS'03), Boston, 2003, pp. 562–571.

[9] H. M. Chen and V. Dalmau, *(Smart) look-ahead arc consistency and the pursuit of CSP tractability*, in Principles and Practice of Constraint Programming (CP'04), Lecture Notes in Comput. Sci. 3258, Springer-Verlag, Berlin, 2004, pp. 182–196.

[10] S. A. Cook, *The complexity of theorem-proving procedures*, in Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC'71), 1971, pp. 151–158.

[11] V. Dalmau, *A new tractable class of constraint satisfaction problems*, Ann. Math. Artif. Intell., 44 (2005), pp. 61–85.

[12] V. Dalmau, R. Gavaldà, P. Tesson, and D. Thérien, *Tractable clones of polynomials over semigroups*, in Principles and Practice of Constraint Programming (CP'05), Lecture Notes in Comput. Sci. 3709, Springer-Verlag, Berlin, New York, 2005, pp. 196–210.

[13] V. Dalmau and J. Pearson, *Closure functions and width* 1 *problems*, in Principles and Practice of Constraint Programming (CP'99), Lecture Notes in Comput. Sci. 1713, Springer-Verlag, Berlin, New York, 1999, pp. 159–173.

[14] T. Feder, *Constraint Satisfaction on Finite Groups with Near Subgroups*, Electronic Colloquium on Computational Complexity (ECCC), TR05-005, 2005.

[15] T. Feder and M. Y. Vardi, *The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory*, SIAM J. Comput., 28 (1998), pp. 57–104.

[16] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[17] P. G. Jeavons, *On the algebraic structure of combinatorial problems*, Theoret. Comput. Sci., 200 (1998), pp. 185–204.

[18] P. G. Jeavons, D. A. Cohen, and M. C. Cooper, *Constraints, consistency and closure*, Artificial Intelligence, 101 (1998), pp. 251–265.

[19] P. Jeavons, D. Cohen, and M. Gyssens, *Closure properties of constraints*, J. ACM, 44 (1997), pp. 527–548.

[20] A. A. Krokhin, A. A. Bulatov, and P. G. Jeavons, *The complexity of constraint satisfaction: An algebraic approach*, in Proceedings of the SMS-NATO Meeting on Structural Theory of Automata, Semigroups and Universal Algebra, Montreal, QB, Canada, 2003, pp. 181–213.

[21] A. K. Mackworth, *Consistency in networks of relations*, Artificial Intelligence, 8 (1977), pp. 99–118.

[22] C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, MA, 1994.

[23] T. J. Schaefer, *The complexity of satisfiability problems*, in Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC'78), 1978, pp. 216–226.

SIAM J. COMPUT.
Vol. 36, No. 1, pp. 28–55

IDENTIFYING CLUSTERS FROM POSITIVE DATA*

JOHN CASE†, SANJAY JAIN‡, ERIC MARTIN§, ARUN SHARMA¶, AND FRANK STEPHAN‖

Abstract. The present work studies clustering from an abstract point of view and investigates its properties in the framework of inductive inference. Any class S considered is given by a hypothesis space, i.e., numbering, $A_0, A_1, \ldots$ of nonempty recursively enumerable (r.e.) subsets of $\mathbb{N}$ or $\mathbb{Q}^k$. A clustering task is a finite and nonempty set of r.e. indices of pairwise disjoint such sets. The class S is said to be *clusterable* if there is an algorithm which, for every clustering task I, converges in the limit on any text for $\bigcup_{i\in I} A_i$ to a finite set J of indices of pairwise disjoint clusters such that $\bigcup_{j\in J} A_j = \bigcup_{i\in I} A_i$. A class is called *semiclusterable* if there is such an algorithm which finds a J with the last condition relaxed to $\bigcup_{j\in J} A_j \supseteq \bigcup_{i\in I} A_i$.

The relationship between natural topological properties and clusterability is investigated. Topological properties can provide sufficient or necessary conditions for clusterability, but they cannot characterize it. On the one hand, many interesting conditions make use of both the topological structure of the class and a well-chosen numbering. On the other hand, the clusterability of a class does not depend on which numbering of the class is used as a hypothesis space for the clusterer.

These ideas are demonstrated in the context of naturally geometrically defined classes. Besides the text for the clustering task, clustering of many of these classes requires the following additional information: the class of convex hulls of finitely many points in a rational vector space can be clustered with the number of clusters as additional information. Interestingly, the class of polygons (together with their interiors) is clusterable if the number of clusters and the overall number of vertices of these clusters is given to the clusterer as additional information. Intriguingly, this additional information is not sufficient for classes including figures with holes.

While some classes are unclusterable due to their topological structure, others are only computationally intractable. An oracle might permit clustering all computationally intractable clustering tasks but fail on some classes which are topologically difficult. It is shown that an oracle E permits clustering all computationally difficult classes iff $E \geq_T K \wedge E' \geq_T K''$. Furthermore, no 1-generic oracle below K and no 2-generic oracle permits clustering any class which is not clusterable without an oracle.

Key words. inductive inference, clustering, hypothesis space, numbering, Turing degree, topological and geometric properties of clusterable classes

AMS subject classifications. 03D20, 03D25, 68Q32, 68T05

DOI. 10.1137/050629112

*Received by the editors April 12, 2005; accepted for publication (in revised form) December 2, 2005; published electronically May 3, 2006. The work of J. Case is supported in part by NSF grant CCR-0208616 and by USDA IFAFS grant 01-04145. The work of S. Jain is supported in part by NUS grant R252-000-127-112. A. Sharma and F. Stephan conducted most of this research while working at National ICT Australia which is funded by the Australian Government's Department of Communications, Information Technology and the Arts and by the Australian Research Council through Backing Australia's Ability and the ICT Centre of Excellence Program. The work of F. Stephan is supported in part by NUS grant R252-000-212-112.

http://www.siam.org/journals/sicomp/36-1/62911.html

†Computer and Information Sciences Department, University of Delaware, Newark, DE 19716-2586, (case@cis.udel.edu).

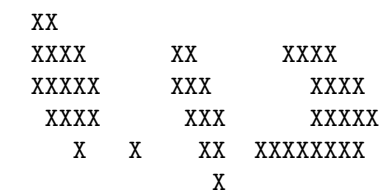
‡School of Computing, National University of Singapore, Singapore 117543 (sanjay@comp.nus.edu.sg).

§School of Computer Science and Engineering, UNSW Sydney NSW 2052, Sydney, Australia (emartin@cse.unsw.edu.au).

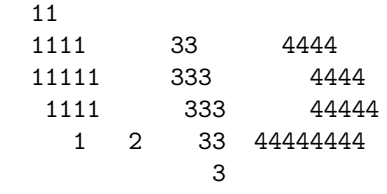
¶Division of Research and Commercialization, Queensland University of Technology, GPO Box 2434, Brisbane QLD 4001, Australia (arun.sharma@qut.edu.au).

‖Department of Mathematics and School of Computing, National University of Singapore, Singapore 117543 (fstephan@comp.nus.edu.sg).

1. Introduction. The basic idea of clustering, given a set of points



is to find a natural way to group the points into clusters (sets) such that every point belongs to exactly one cluster; the current example gives four clusters:



Clustering has been widely studied in several forms in the fields of machine learning and statistics [2, 5, 10, 17, 19]. Abstract treatments of the topic are rare; however, Kleinberg [14] provides an axiomatic approach to clustering. The present work investigates clustering from the perspective of Gold-style learning theory [9, 11], where limitations can stem from uncomputable phenomena.

The purpose of this paper is to study the roles of computation, topology, and geometry in the clustering process. In this interest, the following topics are investigated in an abstract model of clustering:

1. necessary or sufficient topological conditions for clustering;
2. various relationships between clustering, learning, and hypothesis spaces;
3. clusterability of many natural classes of geometrically defined objects;
4. oracles as a method to distinguish between topological and computational aspects of clustering.

The basic setting is that a hypothesis space of potential clusters is given. This space is recursively enumerable. A finite set I of (r.e. indices of) pairwise disjoint clusters from the given class is called a *clustering task*. Given such a clustering task, the clusterer—which might be any algorithmic device—receives a text containing all the data occurring in these clusters and is supposed to find in the limit a set J of (indices of) pairwise disjoint clusters which cover all the data to be seen. There are two variants with respect to a third condition: if one requires that the union of the clusters given by I is the same as the union of the clusters given by J, then one refers to this problem as *clustering*; if this condition is omitted, then one refers to this problem as *semiclustering*.

Clustering is, in some cases, more desirable than semiclustering; for example, the clustering tasks from the class $S_{\mathrm{conv},k}$ defined in Definition 8.1 are collections of convex sets having a positive distance from each other. The solution to such a clustering task is unique since each of these sets corresponds to a cluster. A clusterer has to identify these sets, while a semiclusterer can just converge to the convex hull of all data to be seen. Such a solution is legitimate for semiclustering since it is again a member of the class $S_{\mathrm{conv},k}$. But it fails to meet the intuition behind clustering since it does not distinguish the data from the various clearly different clusters.

Note that in the process of clustering, it is sufficient to find the set J of indices mentioned above. From this J one can find, for every data-item x in the set $\bigcup_{j\in J} A_j$ of all permitted data, the unique cluster to which x belongs. One just enumerates the

sets with the indices in J until the data-item appears in one of them and then uses an index of this set as a description for the cluster to which this data-item belongs. Thus, from a recursion-theoretic point of view, finding the set J is the relevant part of a given clustering problem.

For every indexed class of recursively enumerable sets, there is a canonical translation from these indices to type-0 grammars in the Chomsky hierarchy which generate the corresponding sets. This links the current setting of clustering to grammatical inference, although there is no need herein to exploit the detailed structure of the grammars obtained by such a translation.

We now summarize some of our results.

1. A class has the finite containment property iff any finite union of its members contains only finitely many other members. In section 5 it is shown that classes satisfying this natural property separate the basic notions of clusterability, semiclusterability, and learnability. There is no purely topological characterization of clusterable classes: if a class contains an infinite set C and all singleton sets disjoint from C, then the class is clusterable iff C is recursive. Proposition 6.1 gives a further characterization, which depends on the numbering: a class of disjoint sets is clusterable iff it has a numbering in which every set occurs only finitely often. Section 6 provides some further sufficient criteria for clusterability that take into account topological aspects as well as properties of the given hypothesis space. These criteria are refinements of the finite containment property.

2. Clusterable classes are learnable, but learnable classes may not be clusterable. Although clusterable classes are, by definition, uniformly recursively enumerable, the set of clustering tasks might fail to be. Proposition 3.2 shows that a class that can be clustered using a class comprising hypothesis space, that is, a hypothesis space which enumerates the members of a superclass, can also be clustered using any hypothesis space which enumerates the members of the class only. But by Example 3.3, a clusterable class might not be clusterable with respect to some class comprising hypothesis space.

3. In sections 7 and 8 it is demonstrated how one can map concrete examples into this general framework. These concrete examples are geometrically defined subsets of $\mathbb{Q}^k$: affine sets, classes of sets with distinct accumulation points, and convex hulls of finite sets. This third example is not clusterable, but it turns out to be clusterable if some additional information about the clustering task given to the clusterer is revealed. While there are several natural candidates for the additional information in the case of convex hulls of finite sets, this approach becomes much more difficult when dealing with clusters of other shapes. In the case of polygons in the 2-dimensional space, the additional information provided can consist of the number of clusters plus the overall number of vertices in the polygons considered. Still, this additional information is insufficient for clustering classes of geometrical objects, some of which have holes. But the k-dimensional area is sufficient additional information as long as one rules out that the symmetric difference of two clusters has k-dimensional area 0.

4. Oracles are a way to distinguish between topological and computational difficulty of a clustering problem. In section 4 the relationship between an oracle E and the classes clusterable relative to E is investigated. For example, every 1-generic oracle E which is Turing reducible to the halting problem is trivial: every class which is clusterable relative to E is already clusterable without any oracle. On the other hand, some classes are not clusterable relative to any oracle. Proposition 4.3 characterizes the maximal oracles which permit clustering of any class which is clusterable relative to *some* oracle; in particular, it is shown that such oracles exist.

2. The basic model. Most of the notation follows the books [11, 18]. The next paragraph summarizes the most important notions used in the present work.

Basic notation. A class S is assumed to consist of recursively enumerable subsets of a countable underlying set $\mathbb{U}$ where, in sections 3–6, $\mathbb{U}$ is the set of natural numbers $\mathbb{N}$ and where, in sections 7 and 8, $\mathbb{U}$ is a rational vector space of finite positive dimension. Most of the time, S is even required to be *uniformly recursively enumerable*, which means that there is a sequence $A_0, A_1, \ldots$ of subsets of $\mathbb{U}$ such that first, $S = \{A_0, A_1, \ldots\}$, and second, $\{(i, x) \in \mathbb{N} \times \mathbb{U} : x \in A_i\}$ is recursively enumerable. Such a sequence $A_0, A_1, \ldots$ is called a *numbering* for S. The only exceptions are Proposition 3.9 and Remark 4.2, where clusterability is used for general classes as defined in Definition 3.8.

The letters I, J, H always range over finite subsets of $\mathbb{N}$. The norm is used to induce a one-one ordering of the finite sets: it is defined as $\mathrm{norm}(I) = \sum_{i \in I} 2^i$. Note that $\mathrm{norm}(I) \leq \mathrm{norm}(J)$ whenever $I \subseteq J$. Define A_I as $\bigcup_{i \in I} A_i$. Let $A_{i,s}$ denote the set of elements enumerated into A_i within s steps, and let $A_{I,s} = \bigcup_{i \in I} A_{i,s}$. Without loss of generality, $A_{i,s} \subseteq \{0, 1, \ldots, s\}$ for all s. The sets $A_{i,s}$ are uniformly recursive; that is, $\{(i, s, x) : x \in A_{i,s}\}$ is recursive.

Let $\mathrm{disj}(S)$ contain all finite sets I such that $A_i \cap A_j = \emptyset$ for all different $i, j \in I$. The sets in $\mathrm{disj}(S)$ are called *clustering tasks*. There is an approximation $\mathrm{disj}_s(S)$ to $\mathrm{disj}(S)$ such that $I \in \mathrm{disj}_s(S)$ if $A_{i,s} \cap A_{j,s} = \emptyset$ for all different $i, j \in I$.

For any set A, let $|A|$ be the cardinality of A. Let A^* be the set of all finite sequences of members of A and $|\sigma|$ be the length of a string $\sigma \in A^*$.

A text for a nonempty set $A \subseteq \mathbb{U}$ is any infinite sequence containing all elements, but no nonelements, of A. Clusterers and semiclusterers are recursive functions from $\mathbb{U}^*$ to finite subsets of $\mathbb{N}$; learners are recursive functions from $\mathbb{U}^*$ to $\mathbb{N}$. Also, mappings M^E represented by a machine M having access to an oracle E are considered. An element $\sigma \in A^*$ is called a *stabilizing sequence* (*for A and M*) if $M(\sigma\tau) = M(\sigma)$ for all $\tau \in A^*$.

The sequence $W_0, W_1, \ldots$ denotes an acceptable numbering of all recursively enumerable sets, and W_e can be interpreted as the domain of the eth partial-recursive function φ_e. The set $K = \{e : e \in W_e\}$ is called the halting problem and this notion can be generalized to computation relative to oracles: A' is the halting problem relative to A; in particular, K' is the halting problem relative to K, and K'' is relative to K'. For more information on iterated halting problems, see [18, p. 450].

An oracle G is k-generic if for every Σ^0_k-set T of strings there is a prefix $\eta \preceq G$ such that either $\eta \in T$ or $\eta' \notin T$ for all $\eta' \succeq \eta$. There are 1-generic sets but no 2-generic sets below K. Nevertheless, k-generic sets exist for all $k \in \{1, 2, \ldots\}$.

DEFINITION 2.1. *A class $S = \{A_0, A_1, \ldots\}$ of clusters is called* clusterable *iff there is a clusterer M which, for every $I \in \mathit{disj}(S)$, converges on every text for A_I to a $J \in \mathit{disj}(S)$ with $A_J = A_I$. Such an M is called a* clusterer for S.

S is called semiclusterable *if one replaces $A_J = A_I$ with the weaker condition that $A_J \supseteq A_I$.*

S is called learnable in the limit from positive data with respect to the hypothesis space $A_0, A_1, \ldots$ *iff there is a learner M which, for every $L \in S$, converges on every text for L to a $j \in \mathbb{N}$ with $A_j = L$. In the following, "learnable" stands for "learnable in the limit from positive data with respect to the hypothesis space $A_0, A_1, \ldots$."*

Note that every learner, clusterer, or semiclusterer M which succeeds on A has a stabilizing sequence $\sigma \in A^*$. Furthermore, $M(\sigma)$ is then also a correct hypothesis for A.

Remark 2.2. A clusterer M for $S = \{A_0, A_1, \ldots\}$ might also use a different hypothesis space instead of the default one. Here a numbering $B_0, B_1, \ldots$ is called the *hypothesis space of* M iff for every clustering task I and any text for A_I, M converges on this text to a finite set J such that $B_J = A_I$ and $B_i \cap B_j = \emptyset$ for all different $i, j \in J$. The hypothesis space is *class preserving* if $S = \{B_0, B_1, \ldots\}$ and *class comprising* if $S \subseteq \{B_0, B_1, \ldots\}$. Nevertheless, in light of Proposition 3.2, it is assumed that a clusterer uses the default numbering $A_0, A_1, \ldots$ as its hypothesis space unless explicitly stated otherwise.

Remark 2.3. Many learning criteria have analogous definitions for clustering. For example, a machine M is confident iff it converges on every input to some hypothesis. Thus, one could consider the notion of confidently clusterable classes. This notion is more restrictive; that is, there are classes which are clusterable but not confidently clusterable. In many respects, the theory developed on the basis of these notions is very similar to the corresponding one for learning due to the following reason.

Many separations of different criteria C_1 and C_2 in learning from positive data can be carried over to separations of the corresponding criteria $\tilde{C}_1$ and $\tilde{C}_2$ in clustering. Given a class S separating the learning criterion C_1 from C_2, one can consider the class

$$\tilde{S} = \{\tilde{A} : A \in S\} \text{ where } \tilde{A} = \{0\} \cup \{x + 1 : x \in A\}$$

to separate $\tilde{C}_1$ from $\tilde{C}_2$. The main idea is to use the 0 in order to avoid any two members of $\tilde{S}$ being disjoint. Then every clustering task and every reasonable hypothesis is a singleton set. Learners for S and clusterers for $\tilde{S}$ can be translated into each other.

For example, there is a class S which is learnable but not confidently learnable. Then the class $\tilde{S}$ is clusterable but not confidently clusterable. That is, $\tilde{S}$ witnesses that the notion of confident clustering is more restrictive than the notion of clustering.

This explains some of the many similarities between learning and clustering. Thus, the present work does not focus on the introduction and study of clusterability notions parallel to the many variants of learning in the limit. Instead, emphasis is given more on the relations between clusterability on the one hand and topological, recursion-theoretic, and geometrical properties of classes under consideration on the other.

3. Numberings and clustering. The main topic of this section is the investigation of the role of numberings in clustering. A natural question is whether clustering is independent of the numbering chosen as the hypothesis space. Another important issue is the relationship between numberings of the class of clusters and numberings of the class of finite disjoint unions of clusters. The latter, which represents the clustering tasks, might not have a numbering despite the fact that the former does, as shown in the next example. The class of sets representing the clustering tasks in this example cannot be made recursively enumerable by changing the numbering of the class of clusters.

EXAMPLE 3.1. *For every* $i \in \mathbb{N}$, *let*

$$A_i = \begin{cases} \{i+1\} & \text{if } i \notin K, \\ \{0, i+1\} & \text{if } i \in K. \end{cases}$$

The class $S = \{A_0, A_1, \ldots\}$ *is uniformly recursively enumerable, but the class* $\{A_I : I \in \mathit{disj}(S)\}$ *is not. Taking* i *to be the minimum of* K, *one has for all* $j > i$,

$$j \notin K \Leftrightarrow (\exists I \in \mathit{disj}(S))\,[\{0, i+1, j+1\} \subseteq A_I].$$

This connection holds for all numberings of S but fails for any numbering of the superclass of all finite sets.

Thus, there are clusterable classes in which the corresponding class of all clustering tasks does not have a numbering. Nevertheless, a fundamental result of de Jongh and Kanazawa [4] carries over to clustering: whenever a class is clusterable with respect to a class comprising hypothesis space, the class is also clusterable with respect to every class-preserving hypothesis space. Actually, the following result is even a bit stronger since it does not require that the hypothesis space $B_0, B_1, \ldots$ be class comprising but only that it satisfy the following more technical, but also more general, condition:

$$S \subseteq \{B_J : (\forall i, j \in J)\,[i = j \vee B_i \cap B_j = \emptyset]\}.$$

Thus, for every $I \in \mathrm{disj}(S)$ there is a finite set J such that $\{B_j : j \in J\}$ is pairwise disjoint and $A_I = B_J$. Note that this condition is indeed more general than the class-comprising condition, as it holds in the case that $A_0, A_1, \ldots$ is an enumeration of all two-element sets and $B_0, B_1, \ldots$ is an enumeration of all one-element sets. An application of the next result is that every uniformly recursively enumerable class consisting only of finite sets is clusterable.

PROPOSITION 3.2. *Let $A_0, A_1, \ldots$ be a numbering of a class S and $B_0, B_1, \ldots$ be another numbering (of a possibly different class) such that for every $I \in disj(S)$ there is a J with $A_I = B_J$. If there is a clusterer for S using the hypothesis space $B_0, B_1, \ldots$, then there is another clusterer that uses the original numbering $A_0, A_1, \ldots$ as its hypothesis space.*

Proof. Assume that M is a clusterer for S using the numbering $B_0, B_1, \ldots$ as its hypothesis space. Note that M is required to be correct only on clustering tasks from S, whereas the superclass $\{B_0, B_1, \ldots\}$ is not required to be clusterable.

The clusterer M has on every A_I with $I \in \mathrm{disj}(S)$ a stabilizing sequence σ_I which can be found in the limit: $\sigma_I = \lim_s \sigma_{I,s}$ with $\sigma_I, \sigma_{I,s} \in A_I^*$ and $M(\sigma_I\tau) = M(\sigma_I)$ for all $\tau \in A_I^*$. Then the following clusterer N uses $A_0, A_1, \ldots$ as its hypothesis space.

ALGORITHM N. On input of length s, N computes the output J of M fed with the same input and searches for the set $H \subseteq \{0, 1, \ldots, s\}$ of least norm satisfying the following conditions:

- $H \in \mathrm{disj}_s(S)$;
- $\sigma_{H,s} \in B_{J,s}^*$;
- $M(\sigma_{H,s}\tau) = M(\sigma_{H,s})$ for all $\tau \in B_{J,s}^*$ of length up to s.

If H is found, then output H, else output $\emptyset$.

Verification. Given a clustering task and a text for this task, let J be the hypothesis to which M converges. Let I be the set of least norm such that $A_I = B_J$ and $I \in \mathrm{disj}(S)$. Note that for all H with $\mathrm{norm}(H) < \mathrm{norm}(I)$, either $H \notin \mathrm{disj}(S)$, or $\mathrm{range}(\sigma_H) \not\subseteq B_J$, or there is a $\tau \in B_J^*$ such that $M(\sigma_H\tau) \neq M(\sigma_H)$. Thus, if the length s of the part of the text fed into N is sufficiently large, then the following properties hold:

- $I \subseteq \{0, 1, \ldots, s\}$;
- for all H with $\mathrm{norm}(H) \leq \mathrm{norm}(I)$, $H \in \mathrm{disj}_s(S) \Leftrightarrow H \in \mathrm{disj}(S)$;
- for all $H \in \mathrm{disj}(S)$ with $\mathrm{norm}(H) \leq \mathrm{norm}(I)$, $\sigma_{H,s} = \sigma_H$;
- $\sigma_I \in B_{J,s}^*$ and $M(\sigma_I)$ outputs J;
- for all $H \in \mathrm{disj}(S)$ with $\mathrm{norm}(H) < \mathrm{norm}(I)$, either $\sigma_H \notin B_J^*$ or there is a $\tau \in B_{J,s}^*$ of length up to s with $M(\sigma_H\tau) \neq M(\sigma_H)$.

Hence I satisfies the search conditions of N, but no H with $\mathrm{norm}(H) < \mathrm{norm}(I)$ does. Thus, N converges on a text for B_J to the set I and N witnesses that S is clusterable using the hypothesis space $A_0, A_1, \ldots$ for S. □

EXAMPLE 3.3. *The converse of Proposition* 3.2 *does not hold: there is a clusterable class S and a numbering of a superclass of S such that no clusterer for S can use this numbering as a hypothesis space.*

Proof. For every i, let $A_i = \{\langle i, x\rangle : x \leq |W_i|\}$ and let $S = \{A_0, A_1, \ldots\}$. It is easy to see that S is clusterable using the numbering $A_0, A_1, \ldots$ as the hypothesis space: on input σ the clusterer just outputs $\{i : \langle i, 0\rangle \in \mathrm{range}(\sigma)\}$.

For better readability, the second numbering has two indices. One defines that $B_{i,j} = \{\langle i, x\rangle : \min(\{j, x\}) \leq |W_i|\}$. Note that $B_{i,j} = \{\langle i, 0\rangle, \langle i, 1\rangle, \ldots\}$ iff either W_i is infinite or $j \leq |W_i|$. Furthermore $\{A_i : i \in \mathbb{N}\} \subseteq \{B_{i,j} : i, j \in \mathbb{N}\}$; that is, the hypothesis space of the $B_{i,j}$ is class comprising.

Assume by way of contradiction that M is a clusterer for S using the numbering of the $B_{i,j}$ as its hypothesis space. Given any i, M converges on every text for A_i to a singleton $\{(i, j)\}$ with $A_i = B_{i,j}$. If W_i is finite, then $j > |W_i|$. Thus, one can compute relative to K whether W_i is finite as follows:

1. Taking a default enumeration of A_i, one can use K to determine j such that M—using this enumeration as a text for A_i—converges to $\{(i, j)\}$;
2. one can use K to determine whether $|W_i| > j$;
3. if $|W_i| > j$, W_i is infinite; if not, W_i is finite.

This K-recursive algorithm contradicts the fact that the set $\{i : W_i \text{ is finite}\}$ has the same Turing degree as K' and gives the desired contradiction. □

Although there are classes $S = \{A_0, A_1, \ldots\}$ such that $\{A_I : I \in \mathrm{disj}(S)\}$ is not uniformly recursively enumerable, the superclass $\{A_I : I \subseteq \mathbb{N} \wedge |I| \text{ is finite}\}$ is uniformly recursively enumerable. A clusterer for S can easily be converted to a learner for S using the hypothesis space given by the numbering $B_0, B_1, \ldots$, which satisfies $B_{\mathrm{norm}(I)-1} = A_I$ for all nonempty sets I. But learnability of uniformly recursively enumerable classes does not depend on the hypothesis space; following a result of de Jongh and Kanazawa [4], there is also a learner for S which uses $A_0, A_1, \ldots$ as its hypothesis space. Thus, every clusterable class is learnable, although the converse direction does not hold.

Property 3.4. Every clusterable class is learnable.

EXAMPLE 3.5. (a) *The class S_{gold} consisting of $\mathbb{N}$ and all its finite subsets is neither learnable nor clusterable. But S_{gold} is semiclusterable.*

(b) *The class S_{sing} consisting of all singletons and the set $\mathbb{N}$ is learnable and semiclusterable but not clusterable.*

(c) *Let C be infinite and recursively enumerable. The class S_C consisting of C and all singletons disjoint from C is learnable. Furthermore, S_C is clusterable iff S_C is semiclusterable iff C is recursive.*

Proof. (a) Gold [9] observed that S_{gold} is not learnable. By Property 3.4, the class S_{gold} is also not clusterable. But S_{gold} is semiclusterable by the trivial algorithm, which always conjectures an index for $\mathbb{N}$.

(b) The class S_{sing} is learnable by the algorithm which conjectures an index for $\mathrm{range}(\sigma)$ if $|\mathrm{range}(\sigma)| = 1$ and an index for $\mathbb{N}$ if $|\mathrm{range}(\sigma)| \neq 1$. Since every finite set belongs to a clustering task from S_{sing}, the structure of the clustering tasks of S_{sing} is equal to that of S_{gold}. Thus, S_{sing} is semiclusterable but not clusterable.

(c) Note that the class S_C has a numbering by taking $A_i = C$ if $i \in C$ and $A_i = \{i\}$ otherwise. One first enumerates i into A_i and, whenever i shows up in C, one enumerates also the other elements of C into A_i.

The class S_C can be learned by conjecturing the cluster A_i for the first number i occurring in the text; once selected, the output is kept forever.

If C is recursive, then S_C is clusterable: on input σ, one outputs $\text{range}(\sigma)$ if $\text{range}(\sigma) \cap C = \emptyset$ and $\{\min(C)\} \cup (\text{range}(\sigma) - C)$ otherwise. What this algorithm does is output the set containing the minimal indices of the clusters which intersect the set of data items seen so far. Note that every clusterer is also a semiclusterer. Thus, S_C is semiclusterable as well.

It remains to show that C is recursive whenever there is a semiclusterer M for S_C. The set C has a stabilizing sequence σ with respect to M. Now let $J = M(\sigma)$. There is a finite and possibly empty set D disjoint from C such that $A_J = C \cup D$. Thus, one has that

$$x \notin C \Leftrightarrow x \in D \vee (\exists \tau \in C^*)\,[M(\sigma x \tau) \neq M(\sigma)].$$

These formulas witness that $\overline{C}$ is recursively enumerable. Since C itself is also recursively enumerable, the set C is actually recursive. □

The classes S_{sing} and S_C, where C is nonrecursive, are learnable but not clusterable. Both have the property that they are not closed under disjoint union. The next, easy-to-verify, result shows that this property is essential for getting examples which are learnable but not clusterable.

Property 3.6. Let a class S be closed under disjoint union, that is, $A \cup B \in S$ for all disjoint $A, B \in S$. Then S is clusterable iff S is learnable.

A learner M for a class S is called *prudent* if it outputs only indices of sets it learns. One can enumerate all possible hypotheses $e_0, e_1, \ldots$ of M and thus obtain a numbering $B_0, B_1, \ldots$ with $B_i = W_{e_i}$ of a learnable superclass of S. Fulk [8] showed that every learnable class has a prudent learner. Therefore, it is sufficient to consider only uniformly recursively enumerable classes for learning. So Fulk's result can be stated as follows.

Property 3.7 [see 11, Proposition 5.20]. Every learnable class has a prudent learner. In particular, every learnable class is contained in some learnable and uniformly recursively enumerable class.

Thus, every learnable class can be extended to one which is learnable and uniformly recursively enumerable. But in contrast to learning in the limit, this requirement turns out to be restrictive for clustering. Indeed, Proposition 3.9 below gives for every $\{0,1\}$-valued function $F \not\leq_T K''$ a clusterable class which is not contained in any uniformly recursively enumerable clusterable class. Furthermore, the union of any two such classes, given by different functions F, F', is no longer clusterable. So one cannot cover these classes by countably many clusterable superclasses. Most interesting results are based on Definition 2.1 with the consequence that only countably many classes are clusterable. The more general notion below expands the collection of clusterable classes to an uncountable one. Although the latter collection contains many irregular classes of limited interest, it still gives some fundamental insights.

This notion of "clustering in the general sense" is formally introduced in Definition 3.8 and used only in Proposition 3.9 and Remark 4.2 below. As the enumeration $A_0, A_1, \ldots$ is not available, a fixed acceptable numbering $W_0, W_1, \ldots$ is used as hypothesis space instead.

DEFINITION 3.8. *A class S of recursively enumerable sets is* clusterable in the general sense *iff there is a machine M which converges on every text for the union of finitely many disjoint sets $L_0, L_1, \ldots, L_n \in S$ to a finite set J of indices of pairwise disjoint members of S such that $L_0 \cup L_1 \cup \cdots \cup L_n = \bigcup_{e \in J} W_e$.*

PROPOSITION 3.9. *Let F be a $\{0,1\}$-valued function which is not computable relative to the oracle K''. For all $x, y \in \mathbb{N}$ and $z \in \{0,1\}$, let $A_{x,z}, B_{x,y}$ be defined as*

$$A_{x,z} = \{\langle x, 0, z\rangle, \langle x, 1, z\rangle, \langle x, 2, z\rangle, \ldots\},$$
$$B_{x,y} = \{\langle x, y, 0\rangle, \langle x, y, 1\rangle\}.$$

Then the class S containing all sets $A_{x,z}$ and $B_{x,y}$ with $x, y \in \mathbb{N}$ and $z = F(x)$ is clusterable in the general sense but not contained in any clusterable class which is uniformly recursively enumerable.

Proof. A clustering algorithm outputs on input σ a set J which contains indices of the following sets:

- $A_{x,z}$ whenever $\langle x, 0, z\rangle \in \text{range}(\sigma)$ but $\langle x, 0, 1-z\rangle \notin \text{range}(\sigma)$;
- $B_{x,y}$ whenever $B_{x,y} \subseteq \text{range}(\sigma)$.

The verification of the correctness of this algorithm can be carried out by taking into account that for every x the following holds: S contains exactly one of the sets $A_{x,0}, A_{x,1}$; every clustering task never contains both $A_{x,z}$ and $B_{x,y}$.

Assume by way of contradiction that $C_0, C_1, \ldots$ is a numbering of a clusterable superclass of S. This numbering contains exactly one, and only one, of the sets $A_{x,0}, A_{x,1}$ since $A_{x,F(x)} \in S$ and every class containing both $A_{x,0}, A_{x,1}$ together with the sets $B_{x,y}$ for all $y \in \mathbb{N}$ is not clusterable. The class of all $A_{x,0}, A_{x,1}$, and $B_{x,y}$ has a basic principle in common with the class S_{sing} from Example 3.5: the set $A_{x,0} \cup A_{x,1}$ is the disjoint union of the subsets $B_{x,0}, B_{x,1}, \ldots$, and therefore no clusterable superclass of S contains both sets $A_{x,0}$ and $A_{x,1}$. Thus, one can get F from the numbering $C_0, C_1, \ldots$ as follows:

$$F(x) = z \Leftrightarrow (\exists i)\,[C_i = A_{x,z}].$$

Since the equality of two recursively enumerable sets can be tested relative to the oracle K', the function F would be computable relative to K'' in contradiction to the choice of F. □

4. Clustering and oracles. Oracles are a method for measuring the complexity of a problem. Some classes are clusterable with a suitable oracle, while others cannot be clustered with any oracle. Thus, the use of oracles permits us to distinguish problems caused by the computational difficulty of the class involved from those which are unclusterable for topological reasons. This is illustrated in the following remark.

Remark 4.1. Recall the classes S_C and S_{gold} from Example 3.5. The class S_C is clusterable iff the set C in its definition is recursive. It is easy to see that supplying C as an oracle to the clusterer resolves all computational problems in the case that C is not recursive. But the class S_{gold} is unclusterable because of its topological structure and remains unclusterable relative to every oracle.

Oracles have been extensively studied in the context of inductive inference [1, 6, 13, 16]. These studies considered arbitrary classes but not uniformly recursively enumerable ones. The results for arbitrary classes carry over directly from learning to clustering in the general sense.

Remark 4.2. Fortnow and coworkers [6] investigated for many settings of learning the question of which oracles are *maximal for learning* in the sense that they enable us to solve all principally solvable learning problems. Jain and Sharma [13] showed that in the setting of learning languages from positive data there is no maximal oracle. The same holds for clustering: for every oracle E, the class S^E_{jump} consisting of all

sets $\{2x, 2x+1\}$ with $x \in E'$ and $\{2x\}, \{2x+1\}$ with $x \notin E'$ is clusterable in the general sense relative to an oracle F iff $E' \leq_T F'$. The reason is that a clusterer M^F on a text for $\{2x, 2x+1\}$ can figure out in the limit how many clusters of S^E_{jump} are needed to cover $\{2x, 2x+1\}$:

$$x \in E' \Leftrightarrow M^F \text{ converges on } 2x\,(2x+1)\,(2x+1)\ldots \text{ to } I \text{ with } |I| = 1;$$
$$x \notin E' \Leftrightarrow M^F \text{ converges on } 2x\,(2x+1)\,(2x+1)\ldots \text{ to } I \text{ with } |I| = 2.$$

Thus, there is no oracle E which is *maximal for clustering in the general sense*, meaning that every class which is clusterable in the general sense relative to some oracle is also clusterable in the general sense relative to E.

An oracle is called *trivial for clustering in the general sense* iff every class which is clusterable in the general sense relative to this oracle is also clusterable in the general sense without it. Now it is shown that a nonrecursive oracle E is trivial for clustering in the general sense iff it has a 1-generic degree and is Turing reducible to the halting problem, that is, Case 1 below is satisfied.

Case 1. $E \leq_T G$ *for a* 1*-generic set* $G \leq_T K$. Let S be any class which is clusterable in the general sense relative to E. By [6, Lemma 4.19] there is a clusterer M^G which asks on every text belonging to any clustering task from S only finitely many queries to G. The answers to these queries can be successfully figured out in the limit—thus there is a recursive clusterer for S which converges on every text of any finite disjoint union of sets in S to exactly the same output as M^G. In particular, G (and thus E) is trivial for clustering in the general sense.

Case 2. $E \not\leq_T G$ *for any* 1*-generic set* $G \leq_T K$. Kummer and Stephan [16, Theorem 10.5] showed that there is a class S^E_{func} which is learnable relative to E but not without any oracle. This class S^E_{func} consists of graphs of recursive functions and, following Remark 2.3, one can assume without loss of generality that $f(0) = 0$ for every function whose graph is in S^E_{func}. The class S^E_{func} is, on the one hand, clusterable in the general sense relative to E and, on the other hand, not clusterable in the general sense without any oracle. In particular, S^E_{func} witnesses that E is not trivial for clustering in the general sense.

The previous remark completes the investigation of clustering in the general sense within the present work. From now on, S denotes again a uniformly recursively enumerable family of clusters. That is, $S = \{A_0, A_1, \ldots\}$, and the set $\{(i, x) \in \mathbb{N}^2 : x \in A_i\}$ is recursively enumerable.

The usefulness of oracles with respect to clustering differs much from the case of clustering in the general sense. Dealing only with uniformly recursively enumerable classes reduces our ability to separate oracles by suitable classes. The definitions for maximal and trivial oracles for clustering are the following.

Call an oracle E *maximal for clustering* if every uniformly recursively enumerable class which is clusterable relative to some oracle is already clusterable relative to E. Call an oracle E *trivial for clustering* if every uniformly recursively enumerable class which is clusterable relative to E is already clusterable without any oracle.

Here the word "maximal" instead of "omniscient" is used since by Remark 4.1 some classes are not clusterable with any oracle. In contrast, omniscient oracles for learning functions permit us to learn the class of all recursive functions [1] and do not leave any function learning problem unsolved.

The next result shows that, in contrast to the case of clustering in the general sense, there are maximal oracles for clustering. It turns out that for an oracle E below K the following three conditions are equivalent: E is trivial for clustering, E is trivial

for learning sets, E is trivial for learning functions; see [6] for the equivalence of the last two statements.

PROPOSITION 4.3. *For every oracle E the following statements are equivalent:*

(a) *$E \geq_T K$ and $E' \geq_T K''$;*

(b) *the oracle E is maximal for learning from positive data—every uniformly recursively enumerable class is either not learnable with any oracle or learnable with oracle E;*

(c) *the oracle E is maximal for clustering—every uniformly recursively enumerable class is either not clusterable with any oracle or clusterable with oracle E.*

Proof. Assume that E satisfies $E \geq_T K$ and $E' \geq_T K''$ and assume that $S = \{A_0, A_1, \ldots\}$ is clusterable relative to some oracle. Then S satisfies Angluin's telltale condition below and one can actually give an algorithm which succeeds with the oracle E.

Angluin's condition (see [3]). The class S is clusterable with the help of some oracle iff for every $I \in \mathrm{disj}(S)$, there is a finite set D, called a *telltale set for I*, such that $D \subseteq A_I$ and no $J \in \mathrm{disj}(S)$ satisfies $D \subseteq A_J \subset A_I$.

Note that one can test with oracle K'' whether the telltale condition holds for given I, D: $F(D, I) = 1 \Leftrightarrow (\forall J \in \mathrm{disj}(S))\,[D \not\subseteq A_J \vee A_J \not\subset A_I]$. This condition has an E-recursive approximation $F_s(D, I)$ which converges for $s \to \infty$ to 1 if $F(D, I) = 1$ holds and to 0 otherwise.

ALGORITHM M. Given an E-recursive enumeration of $\{(D, J) : D$ is a finite subset of $\mathbb{N}$ and $J \in \mathrm{disj}(S)\}$, $M(\sigma)$ outputs J from the first pair (D, J) satisfying the following conditions:

- $D \subseteq \mathrm{range}(\sigma) \subseteq A_J$;
- $F_{|\sigma|}(D, J) = 1$.

In order to guarantee that M is total, the search is limited to the first $|\sigma|$ pairs, and $M(\sigma)$ outputs $\emptyset$ if none of the first $|\sigma|$ pairs qualifies.

Verification. Since every clustering task $I \in \mathrm{disj}(S)$ has a telltale set D' such that $D' \subseteq A_I$ and $F(D', I) = 1$, the algorithm converges to some pair (D, J) with $F(D, J) = 1$. One has that $D \subseteq A_I \subseteq A_J$ and it then follows from Angluin's condition that $A_I = A_J$.

Complete class. It remains to show that condition (a) on E is necessary. The class S_{comp} considered here consists of the sets $A_{i,D}$ defined below, where $i \in \mathbb{N}$ and D is a finite subset of $\mathbb{N}$. Note that below, the entry for $\emptyset$ is given explicitly, and therefore $D \neq \emptyset$ in the second entry; in particular, $\max(D)$ is defined there:

$$\begin{aligned} A_{i,\emptyset} &= \{\langle i, x\rangle : x \in W_i \cup \{0\}\}, \\ A_{i,D} &= \{\langle i, x\rangle : x \in D \cup \{0\} \\ &\qquad \vee\ (x > \max(D) \wedge \{z : \max(D) \leq z < x\} \subseteq W_i) \\ &\qquad \vee\ (x \leq \max(D) \wedge \{z : x \leq z \leq \max(D)\} \subseteq W_i)\}. \end{aligned}$$

Clusterer N with oracle K''. Given input σ, $N^{K''}$ determines the sets

$$B_i = \mathrm{range}(\sigma) \cap \{\langle i, 0\rangle, \langle i, 1\rangle, \ldots\}.$$

Then J consists of the pairs (i, D), where $B_i \neq \emptyset$ and D is a finite set of least norm satisfying one of the following conditions:

1. $A_{i,D} = B_i$;
2. $D = \emptyset$, $B_i \subseteq A_{i,\emptyset}$, and W_i coinfinite;

3. $A_{i,D} = B_i \cup \{\langle i, x\rangle : x \geq \max(D)\}$.

Then $N^{K''}$ outputs J.

Verification. Given a clustering task I, the clusterer obviously finds all i, where $(i, C) \in I$ for some C. Furthermore, there is at most one C with $(i, C) \in I$ since $A_{i,C}$ always contains $\langle i, 0\rangle$. It can be seen that the above cases 1, 2, 3 in the algorithm of $N^{K''}$ are disjoint and that $N^{K''}$ converges syntactically whenever $N^{K''}$ converges semantically:

- If $A_{i,C}$ is finite, then eventually all elements show up and $N^{K''}$ outputs an index for this set.
- If $A_{i,C}$ is infinite and W_i coinfinite, then $C = \emptyset$ and no finite subset of $A_{i,\emptyset}$ is in S_{comp}. Thus, the first case does not apply and $N^{K''}$ puts $(i, \emptyset)$ into J according to the second case.
- If $A_{i,C}$ is infinite and W_i cofinite, then let a_i be the first number such that all $x \geq a_i$ are in W_i. In particular, the set $D = \{x \leq a_i : \langle i, x\rangle \in A_{i,C}\}$ satisfies $A_{i,D} = A_{i,C}$, and therefore (i, D) or some equivalent index goes into J whenever B_i contains all $\langle i, x\rangle \in A_{i,C}$ with $x \leq a_i$.

This completes the verification of the clusterer $N^{K''}$. It is easy to see that S_{comp} is also learnable relative to K''. Furthermore, S_{comp} is clusterable and learnable relative to any oracle which is maximal for clustering.

Hardness. It is sufficient to show that learning is hard since no member of S_{comp} is the disjoint finite union of two or more other members of S_{comp}, and every clusterer therefore has to find for every $L \in S_{\text{comp}}$ a singleton $\{(i, D)\}$ such that $A_{i,D} = L$. In the following, assume that an oracle E and a learner O^E using the oracle E are given. Note that every set $A_{i,\emptyset}$ has a stabilizing sequence. Let σ_i be the first stabilizing sequence found by a search applying the oracle E'. Note that $O^E(\sigma_i)$ has to be an index for $A_{i,\emptyset}$ since $O^E(\sigma_i\tau) = O^E(\sigma_i)$ for all $\tau \in (A_{i,\emptyset})^*$ by the definition of a stabilizing sequence. Let b_i be the maximum of all y with $\langle i, y\rangle \in \text{range}(\sigma_i)$.

Let $H_1 \oplus H_2$ denote the set $\{2x : x \in H_1\} \cup \{2x+1 : x \in H_2\}$. There is an index i such that $W_i = \mathbb{N} \oplus K$. Then $A_{i,\emptyset} = \{\langle i, y\rangle : y \in W_i\}$. Now consider any x with $2x > b_i$ and $A_{i,D}$, where D consists of $2x$ and all y with $\langle i, y\rangle \in \text{range}(\sigma_i)$. If $x \in K$, then $\text{range}(\sigma_i) \cup \{\langle i, 2x+1\rangle\} \subseteq A_{i,\emptyset}$. If $x \notin K$, then $A_{i,D} - A_{i,\emptyset} = \{\langle i, 2x+1\rangle\}$. Therefore,

$$x \in K \Leftrightarrow 2x+1 \in W_i,$$

$$x \notin K \Leftrightarrow (\exists \tau \in (A_{i,\emptyset})^*)\,[O^E(\sigma_i\langle i, 2x+1\rangle\tau) \neq O^E(\sigma_i)].$$

A finite modification of the above formula takes care of the x with $2x \leq b_i$ and shows that K is computable relative to E.

Assume that the set W_i is cofinite and a_i is, as above, the minimum of all x with $\{x, x+1, \ldots\} \subseteq W_i$. Now consider any $y < a_i$ with $y \in W_i$. Then $A_{i,\emptyset} - \{\langle i, y\rangle\}$ is in S_{comp}. Since σ_i is a stabilizing sequence for $A_{i,\emptyset}$, $\langle i, y\rangle$ occurs in σ_i. Thus, there are no elements of W_i strictly between b_i and a_i. In particular, W_i is cofinite iff every $x > b_i$ with $x \in W_i$ actually satisfies $\{x, x+1, \ldots\} \subseteq W_i$. As it is already known that $K \leq_T E$, one has that $K' \leq E'$, and the following algorithm decides relative to E' whether W_i is cofinite.

Given i, compute relative to E' the sequence σ_i and the number b_i. Check whether there is an $x > b_i$ with $x \in W_i$. If not, then W_i is finite and thus coinfinite. If so, one can find such an x with oracle E. Then W_i is cofinite iff $\{x, x+1, \ldots\} \subseteq W_i$, which can again be checked with oracle E'.

Thus, exploiting that $E \geq_T K$ and $E' \geq_T K'$, one can derive that $E' \geq_T K''$ since K'' and the index-set $\{i : W_i \text{ is cofinite}\}$ have the same Turing degree. This completes the proof. □

PROPOSITION 4.4. *Let E be a nonrecursive oracle with $E \leq_T K$.*

(a) *If E has 1-generic degree, then E is trivial and permits us to cluster only classes which can already be clustered without any oracle.*

(b) *If E does not have 1-generic degree, then there is a uniformly recursively enumerable class which can be clustered using the oracle E but not without any oracle.*

The same characterizations hold for learning in place of clustering.

Proof. (a) Clustering S and learning $\tilde{S} = \{A_I : I \in \text{disj}(S)\}$ have the same difficulty if one does not require the use of the hypothesis space $\{A_0, A_1, \ldots\}$. Therefore, if one can cluster S with the help of oracle E, then one can also learn $\tilde{S}$ with the help of the same oracle. Thus, by Kummer and Stephan [16, Theorem 10.5], $\tilde{S}$ can be learned without any oracle. This learner can be interpreted as a clusterer which outputs only singleton classes and uses an acceptable numbering of all recursively enumerable sets as its hypothesis space. By Proposition 3.2 one can translate this learner into a clusterer using $A_0, A_1, \ldots$ as its hypothesis space.

(b) By [16, Theorem 10.5] there is a class S^E_{func} of graphs of recursive functions which can be learned relative to oracle E but not without any oracle. Suppose M^E learns S^E_{func}. The main task is to build a uniformly recursively enumerable superclass which still can be learned with oracle E. Without loss of generality, all functions f with a graph in S^E_{func} satisfy that $f(0) = 0$. Therefore, $\langle 0, 0\rangle$ is in all members of S^E_{func}. Furthermore, $\langle 0, 0\rangle$ will also be in all the members of the superclass S of S^E_{func} to be constructed. Thus, S is clusterable iff it is learnable.

Since $E \leq_T K$, the oracle E has a recursive approximation $E_0, E_1, \ldots$ and the machine M^E has also approximations $M^{E_0}, M^{E_1}, \ldots$ such that M^{E_s} works with E_s instead of E. The sequence of these approximations to M^E is uniformly recursive.

The class S. For every given $i, j, k \in \mathbb{N}$, let $A_{i,j,k}$ contain all pairs $\langle x, y\rangle$ which satisfy one of the three conditions below. The class S consists of all $A_{i,j,k}$ with $i, j, k \in \mathbb{N}$.

Condition 1. The pair $\langle x, y\rangle$ is just $\langle 0, 0\rangle$.

Condition 2. There is a number $s \geq \max(\{i, j, k, x\})$ such that the following statements hold:

- $\varphi_i(z)$ is defined for all $z \leq \max(\{j, x\})$, $\varphi_i(0) = 0$, and $\varphi_i(x) = y$;
- for all t with $k \leq t \leq s$, $M^{E_t}(\langle 0, \varphi_i(0)\rangle\langle 1, \varphi_i(1)\rangle \ldots \langle j, \varphi_i(j)\rangle) = \{i\}$;
- for $z = j, j+1, \ldots, \max(\{j, x\})$, $M^{E_s}(\langle 0, \varphi_i(0)\rangle\langle 1, \varphi_i(1)\rangle \ldots \langle z, \varphi_i(z)\rangle) = \{i\}$;
- either $j = 0$ or $M^{E_s}(\langle 0, \varphi_i(0)\rangle\langle 1, \varphi_i(1)\rangle \ldots \langle j-1, \varphi_i(j-1)\rangle) \neq \{i\}$.

Condition 3. This condition does not depend on $\langle x, y\rangle$ since it covers the case in which the parameters do not permit us to construct a desired set but might already have caused the enumeration of pairs different from $\langle 0, 0\rangle$:

- φ_i is defined on $0, 1, \ldots, j$;
- $M^{E_s}(\eta) \neq M^{E_k}(\eta)$ for some $s > k$ and $\eta \preceq \langle 0, \varphi_i(0)\rangle\langle 1, \varphi_i(1)\rangle \ldots \langle j, \varphi_i(j)\rangle$.

It is easy to see that this class is uniformly recursively enumerable. The intuition behind the conditions is the following. Condition 1 makes the set $A_{i,j,k}$ nonempty and enforces that S is clusterable relative to E iff S is learnable relative to E. Condition 2 tries to put information on the graph of φ_i into $A_{i,j,k}$, where j, k serve as additional information. Condition 3 takes care of the class when the choice of the parameters j, k is inadequate. Note that whenever M^E converges for a total function f to $\{i\}$ such that $\varphi_i = \text{graph}(f)$, there is a position j from which on M^E has converged to $\{i\}$. In

particular, $A_{i,j,k} = \text{graph}(f)$ where k is the least number such that $M^{E_s}(\eta) = M^E(\eta)$ for all $s \geq k$ and $\eta \preceq \langle 0, f(0)\rangle\langle 1, f(1)\rangle \ldots \langle j, f(j)\rangle$. Let $(i_{zero}, j_{zero}, k_{zero})$ be an index of $\{\langle 0,0\rangle\}$ and $(i_{all}, j_{all}, k_{all})$ be an index of $\{\langle x, y\rangle : x, y \in \mathbb{N}\}$.

ALGORITHM N WITH ORACLE E. On input σ, N^E does the following steps:

1. Let $f(m)$ be the least number y such that $\langle m, y\rangle$ is in range(σ);
2. if $f(0)$ or $f(1)$ cannot be recovered from range(σ), then output $(i_{zero}, j_{zero}, k_{zero})$ and halt;
3. if there is $\langle m, z\rangle \in \text{range}(\sigma)$ with $z > f(m)$, then output $(i_{all}, j_{all}, k_{all})$ and halt;
4. find the largest n such that $f(0), f(1), \ldots, f(n)$ can be recovered from range(σ);
5. compute for $m = 0, 1, \ldots, n$ the indices i_m such that
$$M^E(\langle 0, f(0)\rangle\langle 1, f(1)\rangle \ldots \langle m, f(m)\rangle) = \{i_m\}$$
and let $k_{m,|\sigma|}$ be the least number o such that $\langle 0, 1\rangle$ is not enumerated into $A_{i_m,m,o}$ within $|\sigma| - o$ steps;
6. determine all numbers $m \in \{1, 2, \ldots, n\}$ such that either range(σ) consists of the elements enumerated into $A_{i_m,m,k_{m,|\sigma|}}$ within $|\sigma|$ steps or m is the least number with $i_m = i_{m+1} = \cdots = i_n$;
7. output $\{(i_m, m, k_{m,|\sigma|})\}$ for the least m that was selected in step 6 and halt.

Verification. Let L be any set in S. It is easy to see that N^E identifies the sets $A_{i_{zero},j_{zero},k_{zero}}$ and $A_{i_{all},j_{all},k_{all}}$. Thus, one can consider any set $L \in S$ which is of the form $\{\langle x, f(x)\rangle : x < b\}$, where $b \in \{2, 3, \ldots, \infty\}$ and f is a recursive function. It is obvious that any $\langle x, y\rangle \in \text{range}(\sigma)$ satisfies $f(x) = y$, thus f is correctly recovered by N^E and n is the largest integer such that all pairs $\langle x, f(x)\rangle$ with $x \leq n$ occur in σ.

Let j be the least number such that $j < b$ and for $i_j = M^E(f(0), f(1), \ldots, f(j))$, there is a k with $A_{i_j,j,k} = L$. Now fix this k to be the minimal one with $A_{i_j,j,k} = L$. Then $A_{i_j,j,o} \neq L$ for all $o < k$; indeed $A_{i_j,j,o} = A_{i_{all},j_{all},k_{all}}$ for these o. Note that $m, k_{m,|\sigma|}$ as chosen in step 7 of the algorithm, respectively, converge to j and k from below, where k is the least integer such that the $f(0), f(1), \ldots, f(j)$ chosen by the algorithm satisfy

$$(\forall m' \leq j)\,(\forall s \geq k)[M^{E_s}(\langle 0, f(0)\rangle\langle 1, f(1)\rangle \ldots \langle m', f(m')\rangle) = M^E(\langle 0, f(0)\rangle\langle 1, f(1)\rangle \ldots \langle m', f(m')\rangle)\,].$$

Given any text for L, assume that σ is so long that the following statements hold:

1. All pairs $\langle m, f(m)\rangle$ with $m \leq j$ have occurred in σ, and thus N^E knows $f(0), f(1), \ldots, f(j)$;
2. if L is finite, then $L = \text{range}(\sigma)$ and all elements of L are enumerated into $A_{i_j,j,k}$ within $|\sigma|$ steps;
3. for all $m \leq j$ and $t > |\sigma|$, $k_{m,t} = k_{m,|\sigma|}$;
4. an element of $A_{i_m,m,k_{m,|\sigma|}} - L$ is enumerated into $A_{i_m,m,k_{m,|\sigma|}}$ within $|\sigma|$ steps whenever this difference is not empty and $m \leq j$;
5. an element of $L - A_{i_m,m,k_{m,|\sigma|}}$ has occurred in σ whenever this difference is not empty and $m \leq j$.

The first statement implies that N^E can recover the relevant part of f. The second statement implies that whenever L is finite, its elements and the finitely many elements of $A_{i_j,j,k}$ are explicitly known to the learner. The third statement enforces that $k_{j,|\sigma|} = k$, and thus k is known to the learner. The fourth and fifth statements guarantee for all $m < j$ that N^E does not output $\{(i_m, m, k_{m,|\sigma|})\}$ whenever $A_{i_m,m,k_{m,|\sigma|}} \neq L$. By

the choice of j, this holds for all $m < j$ and N^E outputs $\{(i_j, j, k)\}$ on input σ. Thus, N^E identifies L. □

The following example shows that there is a difference between the trivial oracles for clustering in the general sense and clustering of uniformly recursively enumerable classes.

EXAMPLE 4.5. *Every 2-generic oracle is trivial for clustering.*

Proof. Assume that G is 2-generic and $S = \{A_0, A_1, \ldots\}$ is clusterable relative to G via an oracle machine M^G. Without loss of generality, M^E is total for every oracle E. Now consider the following sets of strings:

$$T = \{\eta : (\exists I, J)\,(\exists x, t)\,(\exists \sigma \in A_I^*)\,(\forall \tau \in A_I^*)\,(\forall E \succeq \eta)\,(\forall s \geq t)$$
$$[(J \notin \mathrm{disj}_s(S) \vee A_{I,s}(x) \neq A_{J,s}(x)) \wedge I \in \mathrm{disj}_s(S) \wedge M^E(\sigma\tau) = J]\};$$
$$U_{I,\sigma} = \{\vartheta : (\exists \tau \in A_I^*)\,(\forall E \succeq \vartheta)\,[M^E(\sigma\tau) \neq M^E(\sigma)]\}.$$

The oracles quantified in the definitions above are evaluated only up to a certain point. Thus, one can make the definitions of the sets to be Σ_2^0.

Given any $I \in \mathrm{disj}(S)$, M^G has a stabilizing sequence σ for A_I. If σ is not also a stabilizing sequence for M^E, then there is a τ and a prefix $\vartheta \preceq E$ with $M^E(\sigma\tau) \neq M^E(\sigma)$, where all queries to E while computing these two values target only the domain of ϑ. Thus, $\vartheta \in U_{I,\sigma}$. Since G is 2-generic and σ is a stabilizing sequence for M^G, there is a prefix $\alpha \preceq G$ such that no extension $\vartheta \succeq \alpha$ is in $U_{I,\sigma}$. In particular, σ is a stabilizing sequence for A_I and M^E whenever the oracle E satisfies $E \succeq \alpha$. Thus, the stabilizing sequence σ is uniform for all M^E with $E \succeq \alpha$.

The set T contains all η such that for some $I \in \mathrm{disj}(S)$, for some J and a uniform stabilizing sequence σ for A_I with respect to η, $M^E(\sigma) = J$ for all $E \succeq \eta$ and either $J \notin \mathrm{disj}(S)$ or $A_J \neq A_I$. It follows again that $\eta \not\preceq G$ for all $\eta \in T$. Thus there is a prefix $\theta \preceq G$ such that no extension of θ is in T.

ALGORITHM N. The clusterer N is a variant of the locking sequence hunting construction and searches simultaneously for an $\eta \succeq \theta$ and σ built from the data and a J such that $M^E(\sigma\tau) = J$ for all $E \succeq \eta$ and τ obtained from data seen so far. That is, if at stage s the set D is the range of all data seen so far, N searches the first pair (σ, η) in an enumeration of $\mathbb{N}^* \times \{0,1\}^*$ such that

1. $\sigma \in D^*$ and $\eta \in \theta \cdot \{0,1\}^*$;
2. for all $\tau \in D^*$ with $|\tau| \leq s - |\sigma|$ and all $E, F \succeq \eta$, $M^E(\sigma\tau) = M^F(\sigma)$.

Let $J = M^F(\sigma)$ for the set $F = \{x : \eta(x) \downarrow = 1\}$; N outputs J.

Verification. First, note that the search always terminates, as any $\sigma \in D^*$ with $|\sigma| > s$ and any η extending ϑ trivially satisfy the requirements. Furthermore, there is a pair (σ, η) which is a uniform stabilizing sequence for A_I satisfying $\eta \succeq \theta$, and N finds such a sequence in the limit. Since G strongly avoids T in the sense that no extension of the prefix θ is in T, any pair (σ, η) considered by N infinitely often is a correct uniform stabilizing sequence, and thus N converges to an index $J \in \mathrm{disj}(S)$ of A_I. □

5. The finite containment property. The main topic of this section is to investigate the relationship between the topological structure of the class S and the question of whether S is clusterable. Recall that the classes S_{gold} and S_{sing} are not clusterable for topological reasons: they contain a cluster which is the disjoint infinite union of some other clusters. Thus, one might impose the following natural condition in order to overcome this problem.

DEFINITION 5.1. *A class $S = \{A_0, A_1, \ldots\}$ has the* finite containment property *if every finite union of clusters contains only finitely many clusters. That is, for all i there are only finitely many sets $B \in S$ with $B \subseteq A_{\{0,1,\ldots,i\}}$.*

Note that the finite containment property is not necessary for clusterability. The class $\{\{i, i+1, \ldots\} : i \in \mathbb{N}\}$ is learnable and clusterable but does not satisfy the finite containment property.

It is easy to see that the finite containment property implies Angluin's condition: for every set A_I, there are only finitely many sets A_J with $A_J \subset A_I$. If one takes D to be the set $\{\min(A_I - A_J) : A_J \subset A_I\}$, then D is finite and there is no A_J with $D \subseteq A_J \subset A_I$. Thus, from the proof of Proposition 4.3 one has the following.

Property 5.2. *If $S = \{A_0, A_1, \ldots\}$ has the finite containment property, then S is clusterable relative to every oracle E with $E \geq_T K$ and $E' \geq_T K''$.*

Although the finite containment property guarantees clusterability from the topological point of view, it fails to guarantee clusterability from the recursion-theoretic point of view. Indeed, the class S_{comp} given in the proof of Proposition 4.3 satisfies the finite containment property. If W_i is cofinite, then there are only finitely many subsets of $\{\langle i, 0\rangle, \langle i, 1\rangle, \ldots\}$ in S_{comp}. If W_i is coinfinite, then $A_{i,\emptyset}$ is the only infinite subset of $\{\langle i, 0\rangle, \langle i, 1\rangle, \ldots\}$ in S_{comp} and all further subsets of $\{\langle i, 0\rangle, \langle i, 1\rangle, \ldots\}$ are finite sets which are not contained in $A_{i,\emptyset}$.

Note that the class S_C from Example 3.5 is, for the case that C is nonrecursive, a witness for a class satisfying the finite containment property which is learnable but not semiclusterable. This gives the following property.

Property 5.3. There is a class satisfying the finite containment property which is learnable but neither clusterable nor semiclusterable.

Since the topological structure of S_C is the same whenever C is infinite, clusterability of the class S_C is not determined by its topological structure.

Property 5.4. Clusterability cannot be characterized in topological terms only.

If one takes C to be the halting problem K, then S_C (from Example 3.5) witnesses that the oracle K is necessary for semiclustering, even in the case where classes have to satisfy the finite containment property. Proposition 5.5 below shows that semiclustering is much easier than clustering: every uniformly recursively enumerable class is semiclusterable using the halting problem as an oracle. In particular, no topological condition can make semiclustering impossible, but only computational conditions can.

Furthermore, every uniformly recursive class is semiclusterable. But this condition is not necessary for either semiclusterable or clusterable classes. For example, the class $\{\{x : \varphi_x(x) \downarrow = i\} : i \in \mathbb{N}\}$ is clusterable but consists of pairwise disjoint and recursively inseparable sets.

PROPOSITION 5.5. *Every r.e. class has a semiclusterer using the halting problem as an oracle. Furthermore, a class $S = \{A_1, A_2, \ldots\}$ is semiclusterable without any oracle if the representation of the class is a uniformly recursive family, that is, if $\{(i, x) \in \mathbb{N}^2 : x \in A_i\}$ is recursive and not only recursively enumerable.*

Proof. It is sufficient to assume that M can check whether some x is in A_i. This can be done either by using the halting problem as an oracle or by assuming that the sequence $A_0, A_1, \ldots$ is uniformly recursive.

Now M on input σ determines all $J \subseteq \{0, 1, \ldots, |\sigma|\}$ such that $J \in \text{disj}_{|\sigma|}(S)$ and $\text{range}(\sigma) \subseteq A_J$. If there are several such sets, M outputs the one with the least norm. If there are none, M outputs $\emptyset$.

Note that all J which either do not represent disjoint sets or do not contain all data showing up in the limit are eventually disqualified. On the other hand, the set I representing the clustering task is among the determined sets whenever $|\sigma| \geq \max(I)$.

Thus, M converges in the limit to some J such that $\mathrm{norm}(J) \leq \mathrm{norm}(I)$, $J \in \mathrm{disj}(S)$, and $A_I \subseteq A_J$. Therefore M witnesses that S is semiclusterable. □

By Property 5.3 one can separate learnability from clusterability and semiclusterability by a class satisfying the finite containment property. The next results show that there are no implications between the notions of learnability, clusterability, and semiclusterability except the following two: "clusterable ⇒ semiclusterable" and "clusterable ⇒ learnable." All nonimplications are witnessed by classes satisfying the finite containment property.

PROPOSITION 5.6. *There is a class with the finite containment property which is semiclusterable and learnable but not clusterable.*

Proof. Let S consist of the clusters

$$\begin{aligned} A_{3i} &= \{\langle i, x\rangle : x \in \mathbb{N}\};\\ A_{3i+1} &= \{\langle i, x\rangle : x \text{ is even and } x < 2 + |W_i|\};\\ A_{3i+2} &= \{\langle i, x\rangle : x \text{ is odd and } x < 2 + |W_i|\}. \end{aligned}$$

If W_i is infinite, then A_{3i+1} consists of the $\langle i, x\rangle$ where x is even and W_{3i+2} consists of those $\langle i, x\rangle$ where x is odd. Since the union $A_0 \cup A_1 \cup \cdots \cup A_{3i+2}$ contains only the clusters $A_0, A_1, \ldots, A_{3i+2}$, the class S has the finite containment property. Furthermore, S is semiclusterable by assigning to every input σ the set

$$\{3i : (\exists x \in \mathbb{N})\,[\langle i, x\rangle \in \mathrm{range}(\sigma)]\}.$$

Now it is shown that S is not clusterable. Thus, assume by way of contradiction that a recursive M witnesses S to be clusterable.

For each A_{3i}, one finds by using the oracle K a stabilizing sequence $\sigma_i \in (A_{3i})^*$. One can reduce the question of whether W_i is infinite to the question of whether $\mathrm{range}(\sigma_i) \subseteq A_{\{3i+1,3i+2\}}$, which is decidable relative to K: if W_i is infinite, then $\mathrm{range}(\sigma_i) \subseteq A_{\{3i+1,3i+2\}}$; if W_i is finite, then $\mathrm{range}(\sigma_i) \not\subseteq A_{\{3i+1,3i+2\}}$. The latter holds, since otherwise σ_i would also be a stabilizing sequence for $A_{\{3i+1,3i+2\}}$, and M cannot have the same stabilizing sequence for two different sets in which one set is a subset of the other. The reduction of $\{i : W_i \text{ is infinite}\}$ to the oracle K contradicts the fact that $\{i : W_i \text{ is infinite}\}$ is Turing equivalent to K'.

It remains to show that the class S is learnable. This can be done by considering the following learner N.

ALGORITHM N. On input σ, let i be the least number such that a pair of the form $\langle i, x\rangle$ occurs in $\mathrm{range}(\sigma)$. Then

$$N(\sigma) = \begin{cases} 3i & \text{if there are even and odd } y \text{ with } \langle i, y\rangle \in \mathrm{range}(\sigma);\\ 3i+1 & \text{if there are only even } y \text{ with } \langle i, y\rangle \in \mathrm{range}(\sigma);\\ 3i+2 & \text{if there are only odd } y \text{ with } \langle i, y\rangle \in \mathrm{range}(\sigma). \end{cases}$$

The correctness of the learner N can easily be verified. □

EXAMPLE 5.7. *The class containing all sets $A_{3i}, A_{3i+1}, A_{3i+2}, A_{\{3i+1,3i+2\}}$ from the numbering $A_0, A_1, \ldots$ in the proof of Proposition* 5.6 *is neither learnable nor clusterable. But it satisfies the finite containment property and is semiclusterable.*

A natural variant of the finite containment property is the *finite meet property*, which says that each member of the class S meets only finitely many other members (that is, for each $A \in S$, $|\{A' \in S : A \cap A' \neq \emptyset\}|$ is finite). The class S_C, from Example 3.5, witnesses that for $C = K$ one might need the oracle K to cluster a class satisfying the finite meet property. Since the class given in the proof of Proposition 4.3

satisfies the finite containment property and can be clustered only relative to maximal oracles, the next result shows that classes satisfying the finite meet property are easier and require only the oracle K.

PROPOSITION 5.8. *If a class satisfies the finite meet property, then it is clusterable with the halting-problem oracle K.*

Proof. Let $S = \{A_0, A_1, \ldots\}$ satisfy the finite meet property. Let $b_0, b_1, \ldots$ be a text for A_I with $I \in \mathrm{disj}(S)$; without loss of generality, I consists of minimal indices; that is, for all $i \in I$ and for all j, if $A_i = A_j$, then $i \leq j$. Relative to K and the text, one can enumerate the set

$$H = \{h : A_h \cap \{b_0, b_1, \ldots\} \neq \emptyset \wedge (\forall j < h)\,[A_j \neq A_h]\,\}.$$

Now one considers all subsets $J \subseteq H$ with $J \in \mathrm{disj}(S)$. Note that $I \subseteq H$ and $I \in \mathrm{disj}(S)$, and thus I is among the considered sets. Due to the finite meet property, H is finite and only finitely many J are considered. Since these sets are uniformly recursive relative to K, one can find in the limit a considered set J which satisfies $A_J = \{b_0, b_1, \ldots\}$, that is, $A_J = A_I$. Thus S is clusterable using the oracle K. □

6. Numbering-based properties. Every uniformly recursively enumerable class of pairwise disjoint sets is learnable: the learner just waits until it finds $x \in \mathrm{range}(\sigma)$ and $i \leq |\sigma|$ such that x is enumerated into A_i within $|\sigma|$ steps; from then on the learner outputs the index i. But for nonrecursive sets C, the class S_C witnesses that such a class is not clusterable. Thus, one has to consider not only properties of the class but also properties of some of its numberings. A class $\{A_0, A_1, \ldots\}$ has the *numbering-based finite containment property* if for every I there are only finitely many j with $A_j \subseteq A_I$.

PROPOSITION 6.1. *A class of pairwise disjoint sets has the numbering-based finite containment property iff it is clusterable.*

Proof. Let $S = \{A_0, A_1, \ldots\}$ be a class of pairwise disjoint sets. Due to the numbering-based finite containment property there are, for every i, only finitely many j with $A_j = A_i$. Now consider the following clusterer.

ALGORITHM M. On input σ, find the J of the largest norm which satisfies the following three conditions:

1. $J \subseteq \{0, 1, \ldots, |\sigma|\}$;
2. $J \in \mathrm{disj}_{|\sigma|}(S)$;
3. $A_j \cap \mathrm{range}(\sigma) \neq \emptyset$ for all $j \in J$.

Then output this J.

Verification. Note that the algorithm always terminates since $\emptyset$ satisfies the search conditions. Fix a clustering task I. The set $H = \{h : A_h \cap A_I \neq \emptyset\}$ is finite. Since M always outputs subsets of H, it follows that M converges to some $J \subseteq H$. This J is the set of the highest norm such that $J \subseteq H$ and $J \in \mathrm{disj}(S)$. Since the members of S are pairwise disjoint, it holds for every $j \in J$ that A_j not only meets A_I but, moreover, is contained in A_I. Furthermore, if $i \in I$, then $A_i \cap A_J$ is not empty, since otherwise $J \cup \{i\}$ is also a subset of H, is in $\mathrm{disj}(S)$, and has a norm larger than that of J. Thus, $A_i \subseteq A_J$. Since this holds for all $i \in I$, $A_J = A_I$ and M is a clusterer for S.

Converse direction. Assume that N is a clusterer for S and consider the set

$$E = \{i : (\forall j < i)\,(\forall \sigma \in A^*_{j,i}, |\sigma| \leq i)\,(\exists \tau \in (A_i \cup A_j)^*)\,[N(\sigma\tau) \neq N(\sigma)]\}.$$

The set E is recursively enumerable since the universal quantifiers are bounded and the second one runs over strings of the finite set $A_{j,i}$ of all elements enumerated into

A_j within i steps. Given any set in S, let i be its minimal index. Let $j < i$. Since $A_j \neq A_i$, A_j is disjoint from A_i, $\{j, i\} \in \mathrm{disj}(S)$, and $A_{\{j,i\}}$ is a proper superset of A_i. The clusterer N must converge on texts for A_j and $A_{\{i,j\}}$ to different outputs. Thus, there is no $\sigma \in A_j^*$ with $N(\sigma) = N(\sigma\tau)$ for all $\tau \in A_{\{i,j\}}^*$. The index i is eventually enumerated into E. The set A_i has a stabilizing sequence σ. For all sufficiently large j with $A_i = A_j$, the length of σ is shorter than j and its range enumerated into A_i within j steps. It follows that σ prevents j from being enumerated into E and E contains only finitely many indices of A_i. The set E has a recursive enumeration $e_0, e_1, \ldots$, which defines by $B_h = A_{e_h}$ a new numbering $B_0, B_1, \ldots$ of S having the desired properties. □

Remark 6.2. Proposition 5.6 gives a class which satisfies the numbering-based finite containment property but is not clusterable. A variant of the class S_{comp} given in the proof of Proposition 4.3 satisfies the numbering-based finite containment property but is clusterable only relative to maximal oracles.

Let the numbering-based finite meet property denote that every A_i meets A_j only for finitely many j. It follows from Proposition 5.8 that a class satisfying the numbering-based finite meet property is clusterable with the oracle K. But even this property is not sufficient for clustering without oracles. The class in Example 5.7 actually satisfies the numbering-based finite meet property but is not clusterable.

A further example of a class which satisfies the numbering-based finite meet property but is not clusterable can be constructed using the following result of Jain and Sharma [12]: there is no learner which identifies all recursively enumerable sets from any text for the set plus an upper bound on its least index. The class

$$\{\{\langle i, x\rangle : x \in W_j\} : j \leq i \wedge W_j \neq \emptyset\}$$

has a numbering witnessing that it satisfies the numbering-based finite meet property. But it consists of copies of sets W_j, having coded an upper bound of an index of W_j into its first coordinate. This class cannot be learnable or clusterable because one would get a contradiction to the result of Jain and Sharma otherwise.

In the following, two conditions are presented which are more restrictive than the numbering-based finite containment property and guarantee that a class is clusterable.

PROPOSITION 6.3. *Assume that $A_i \not\subseteq \bigcup_{j \neq i} A_j$ for all i and that it is decidable whether two sets A_i, A_j intersect. Then $S = \{A_0, A_1, \ldots\}$ is clusterable.*

Proof. The clusterer M uses the fact that one can check disjointness effectively, that is, that $\mathrm{disj}(S)$ is recursive.

ALGORITHM M. On input $b_0, b_1, \ldots, b_s$, M considers all $J \subseteq \{0, 1, \ldots, s\}$ satisfying the following conditions:

1. $A_{i,s} \cap \{b_0, b_1, \ldots, b_s\} \neq \emptyset$ for all $i \in J$;
2. $J \in \mathrm{disj}(S)$;
3. there is no $j \in \{0, 1, \ldots, s\} - J$ such that $A_{j,s} \cap \{b_0, b_1, \ldots, b_s\} \neq \emptyset$ and $J \cup \{j\} \in \mathrm{disj}(S)$.

If there are several sets $J_1, J_2, \ldots, J_n \subseteq \{0, 1, \ldots, s\}$ which satisfy all three conditions, then M computes for $m = 1, 2, \ldots, n$ the number

$$c_m = \max\{h \leq s + 1 : \{b_j : j < h\} \subseteq J_{m,s}\}$$

and outputs J_m for the m which maximizes c_m; if there are still several options, M outputs the one with the least norm.

Verification. Assume that a clustering task $I \in \mathrm{disj}(S)$ is given and that $b_0 b_1 \ldots$ is a text for A_I. Let s be so large that there is a c satisfying the following conditions:

- $s \geq \max(I)$;
- for any $i \in I$ there exists an $h \leq c$ with $b_h \in A_i - \bigcup_{j \neq i} A_j$;
- $\{b_0, b_1, \ldots, b_c\} \subseteq A_{I,s}$.

Then I clearly satisfies the first two search conditions of M. The third is also satisfied since, whenever $A_j \cap A_I = \emptyset$, A_j does not contain any of the elements $b_0, b_1, \ldots, b_s$. Thus, any set $J \neq I$ satisfying all three conditions is not a superset of I. In particular, there is an $i \in I - J$ and an $h \leq c$ such that $b_h \in A_i - A_J$. Since $\{b_0, b_1, \ldots, b_c\} \subseteq A_{I,s}$ and $\{b_0, b_1, \ldots, b_c\} \not\subseteq A_J$, M outputs I and not J. Thus, M converges on a text for A_I to I, and M is a clusterer for S. □

PROPOSITION 6.4. *Assume that $S = \{A_0, A_1, \ldots\}$ satisfies the following three conditions:*

1. *Every A_i is infinite;*
2. *if $i \neq j$, then $A_i \cap A_j$ is finite;*
3. *S is uniformly recursive, that is, $\{(i, x) : x \in A_i\}$ is recursive.*

Then S is clusterable. But no two of these three conditions are sufficient for being clusterable.

Proof. On input σ, the clusterer M searches for the J of the least norm satisfying the following properties:

- $J \subseteq \{0, 1, \ldots, |\sigma|\}$;
- $\mathrm{range}(\sigma) \subseteq A_J$;
- $J \in \mathrm{disj}_{|\sigma|}(S)$.

If such a J is found, then M outputs J, else M outputs $\emptyset$.

First, one can easily see that M is recursive since the search space is limited to $2^{|\sigma|+1}$ candidate sets. Second, one considers any clustering task I and any text for it. Every sufficiently long prefix σ of the text satisfies $i \leq |\sigma|$ and $A_i \cap \mathrm{range}(\sigma) \neq \emptyset$ for all $i \in I$. Thus, I satisfies for all sufficiently long σ the three search conditions, and hence M converges to a set J with $\mathrm{norm}(J) \leq \mathrm{norm}(I)$. For every $i \in I$, the set A_i is not a subset of $A_{J-\{i\}}$ since A_i is infinite and $A_i \cap A_{J-\{i\}}$ is finite. Thus, $I \subseteq J$. Since $\mathrm{norm}(J) \leq \mathrm{norm}(I)$ (from above), it follows that $I = J$.

Recall the definition of S_C from Example 3.5. The class $\{A \times \mathbb{N} : A \in S_C\}$ for a nonrecursive parameter-set C satisfies conditions 1 and 2 but is not clusterable. The class of all cofinite sets satisfies conditions 1 and 3 but is neither learnable [11, section 3.6.2] nor clusterable. The class S_{gold} satisfies conditions 2 and 3 but is not clusterable. □

7. Geometric examples. The major topic of this and the following sections is to look at sets of clusters which are characterized by basic geometric properties. Therefore, the underlying set is no longer $\mathbb{N}$ but the k-dimensional rational vector space $\mathbb{Q}^k$, where $k \in \{1, 2, \ldots\}$ is fixed. The classes considered consist of natural subsets of $\mathbb{Q}^k$. This is quite common practice in computer science; for example, the real numbers used in standard programming languages are indeed rationals, as they normally consist only of finitely many binary digits multiplied by a power of 2; furthermore, dealing with rationals avoids the uncountability of the set of reals and also *uncomputable* real points. Except for the class $S_{\mathrm{accu},k}$ in Definition 7.2 below, the following hold: The clusters are built from finitely many parameter-points in $\mathbb{Q}^k$; the clusters are connected sets; and every clustering task consists of clusters having a positive distance from each other. Thus, there is a unique natural way of breaking down a clustering task into clusters.

The space $\mathbb{Q}^k$ is a metric space. The standard metric d between two points is given by the square root of the sum of the squares of the differences of the coordinates, for example, $d((1,2,3),(0,0,5)) = \sqrt{(1-0)^2+(2-0)^2+(3-5)^2} = \sqrt{1+4+4} = 3$ in $\mathbb{Q}^3$. d is also called *distance*.

Recall that a subset $U \subseteq \mathbb{Q}^k$ is *affine* iff for every fixed $x \in U$ the set $V = \{y \in \mathbb{Q}^k : x+y \in U\}$ is a rational vector space, that is, closed under scalar multiplication and addition. The dimension of U is the dimension of V as a vector space; it is independent of the choice of x.

EXAMPLE 7.1. *Let $S_{\mathrm{aff},k}$ be the class of all affine subspaces of $\mathbb{Q}^k$ which have dimension $k-1$. The class $S_{\mathrm{aff},k}$ is clusterable but the class $S_{\mathrm{aff},k} \cup \{\mathbb{Q}^k\}$ is not.*

Proof. If one considers a one-one numbering $A_0, A_1, \ldots$ of $S_{\mathrm{aff},k}$, one can easily verify the following properties:

1. Every A_i has dimension $k-1$;
2. if $i \neq j$, then $A_i \cap A_j$ is either empty or an affine subspace of dimension up to $k-2$;
3. the set $\{(i,x) : x \in A_i\}$ is recursive.

Properties 1 and 2 enforce that $A_i \not\subseteq A_{J-\{i\}}$ for every finite set J. Thus, one can adapt the clusterer for the class in Proposition 6.4 to a clusterer for $S_{\mathrm{aff},k}$. The verification can also easily be transferred.

The class $S_{\mathrm{aff},k} \cup \{\mathbb{Q}^k\}$ is just the geometric version of the class S_{sing} from Example 3.5. Let U be a $(k-1)$-dimensional vector space and W be a 1-dimensional vector space with $U+W = \mathbb{Q}^k$. Furthermore let $U_x = \{x+y : y \in U\}$. Since every U_x is in $S_{\mathrm{aff},k}$ and $\mathbb{Q}^k$ is the disjoint union of all U_x with $x \in W$, it follows that $S_{\mathrm{aff},k} \cup \{\mathbb{Q}^k\}$ is not clusterable. □

DEFINITION 7.2. *Let k be a positive natural number and $S_{accu,k}$ be a class $\{A_0, A_1, \ldots\}$ of bounded subsets of $\mathbb{Q}^k$ for which there is a recursive and one-one sequence $a_0, a_1, \ldots$ of points in $\mathbb{Q}^k$ satisfying the following:*

1. *Every A_i has exactly one accumulation point which is a_i;*
2. *no accumulation point of the set $\{a_0, a_1, \ldots\}$ is contained in this set.*

Comment. Every set $A_i \cup \{a_i\}$ is compact, but it is not required that $a_i \in A_i$, and therefore the set A_i itself might fail to be compact.

PROPOSITION 7.3. *The class $S_{accu,k}$ is clusterable.*

Proof. The following machine M witnesses that $S_{\mathrm{accu},k}$ is clusterable.

ALGORITHM M. On input $b_0 b_1 \ldots b_s$, let

$$H_s = \{i \leq s : (\exists h \leq s)\,(\forall j \leq h, j \neq i)\,[b_h \notin \{b_0, b_1, \ldots, b_i\} \wedge d(a_i, b_h) < d(a_j, b_h)]\},$$

$$J_s = \{i \in H_s : (\exists h)\,(\forall j \in H_s - \{i\})\,[b_h \in A_{i,s} - A_{j,s}]\}$$

and output J_s.

Verification. Since a_i is an accumulation point of A_i but not of $\{a_0, a_1, \ldots\}$, and since $a_i \neq a_j$ whenever $i \neq j$, for every i there is a threshold $\epsilon_i > 0$ such that

- for all $q \in \mathbb{Q}^k$ there is at most one i with $d(a_i, q) < \epsilon_i$;
- for almost all $q \in A_i$, $d(a_i, q) < \epsilon_i$.

Consider now a text $b_0 b_1 \ldots$ for a set A_I with $I \in \mathrm{disj}(S_{\mathrm{accu},k})$. Let $H = \bigcup_{s \in \mathbb{N}} H_s$ where H_s, J_s are the sets constructed by the algorithm with input $b_0, b_1, \ldots, b_s$. There are only finitely many $q \in A_I$ which do not satisfy $d(a_i, q) < \epsilon_i$ for an $i \in I$, and there is a stage $t \geq \max(I)$ such that $\{b_0, b_1, \ldots, b_t\}$ contains all these q. It follows that $H \subseteq \{0, 1, \ldots, t\}$. Note that the intersection $A_i \cap A_j$ is finite for any different i, j since

the sets A_i, A_j are bounded and have different accumulation points. So all sufficiently large s satisfy the following conditions:

- $H = H_s$;
- for all different $i, j \in H$, $A_i \cap A_j = A_{i,s} \cap A_{j,s}$;
- for all $i \in I$, there is an $h \leq s$ such that $b_h \in A_i - (\bigcup_{j \in H} A_j \cup \{b_0, b_1, \ldots, b_i\})$ and $d(b_h, a_i) < \epsilon_i$.

It follows that on the one hand, $I \subseteq H_s$, and on the other hand, that every $j \in H_s - I$ satisfies $A_I \cap A_j \subseteq A_{I,s}$. Since $b_0, b_1, \ldots$ is a text for A_I, it follows that $J_s = I$ and M is a clusterer for $S_{\mathrm{accu},k}$. □

The class $S_{\mathrm{accu},k}$ is clusterable but the machine M makes use of the sequence $a_0, a_1, \ldots$ as an auxiliary source of information. Nevertheless, this information is implicit. One can build a program for it into the machine M, which simulates this program, in order to get some further information on $S_{\mathrm{accu},k}$.

8. Clustering with additional information. Freivalds and Wiehagen [7] introduced a learning model in which the learner receives—in addition to the graph of the function to be learned—an upper bound on the size of some program for this function. This additional information increases the learning power and enables a machine to learn the class of all recursive functions.

Similarly, a machine receiving adequate additional information can solve every clustering task for the class $S_{\mathrm{conv},k}$ defined below. But without that additional information, $S_{\mathrm{conv},k}$ is not clusterable. Thus, the main goal of this section is to determine which pieces of additional information are sufficient to cluster certain geometrically defined classes where clustering without additional information is impossible.

Recall that the convex hull of a set $D = \{x_0, x_1, \ldots, x_n\}$, denoted by $\mathrm{hull}(D)$, is the set given by

$$\mathrm{hull}(D) = \{q_0 x_0 + q_1 x_1 + \cdots + q_n x_n : q_0, q_1, \ldots, q_n \in \mathbb{Q} \\ \wedge q_0, q_1, \ldots, q_n \geq 0 \wedge q_0 + q_1 + \cdots + q_n = 1\}.$$

Given a set E as a convex hull of a finite set, there is a unique minimal set D such that $E = \mathrm{hull}(D)$.

DEFINITION 8.1. *For a given positive natural number k, the class $S_{conv,k}$ contains all subsets of $\mathbb{Q}^k$ which are the rational points in the convex hull of a finite subset of $\mathbb{Q}^k$.*

Note that $S_{\mathrm{conv},k}$ has the following nice properties, which will be used in the proofs implicitly: every cluster is ϵ-connected for all $\epsilon > 0$; any two clusters A_i, A_j have either a point in common or have a positive distance from each other, where the distance is defined as $d(A_i, A_j) = \inf\{d(x, y) : x \in A_i, y \in A_j\}$.

PROPOSITION 8.2. *The class $S_{conv,k}$ is semiclusterable but not clusterable.*

Proof. A semiclusterer M for $S_{\mathrm{conv},k}$ works as follows.

ALGORITHM M. On input σ, M searches for the first $i \in \{0, 1, \ldots, |\sigma|\}$ such that $\mathrm{range}(\sigma) \subseteq \mathrm{hull}(A_{i,|\sigma|})$. If this i is found, then M outputs $\{i\}$, else M outputs $\emptyset$.

Verification. Let I be the clustering task and i be the least index of a set with $A_I \subseteq A_i$. Given any text for A_I, every sufficiently long prefix σ of the text satisfies the following three conditions:

- $|\sigma| \geq i$;
- $\mathrm{range}(\sigma) \not\subseteq A_j$ for all $j < i$;
- $\mathrm{hull}(A_{i,|\sigma|}) = A_i$.

It is easy to see that $M(\sigma) = \{i\}$ for the input σ. Therefore, M converges to i and $S_{\mathrm{conv},k}$ is semiclusterable.

Gold's condition. Note that every singleton in $\mathbb{Q}^k$ belongs to $S_{\mathrm{conv},k}$ and that there are also infinite clusters. Then, given an M and an infinite A_i, M has a stabilizing sequence $\sigma \in (A_i)^*$. Thus, either M fails on the clustering task I representing all singletons $\{x\}$ with $x \in \mathrm{range}(\sigma)$ or M fails on the clustering task $\{i\}$ representing A_i. □

PROPOSITION 8.3. *The class $S_{conv,k}$ is clusterable with additional information if for any clustering task I one of the following pieces of information is also provided to the machine M:*

- *the number $|I|$ of clusters of the clustering task;*
- *a positive lower bound ϵ for $\gamma = \min(\{1\} \cup \{d(A_i, A_j) : i, j \in I \wedge i \neq j\})$;*
- *the minimal number p of points which are needed to generate all the convex sets A_i with $i \in I$.*

Proof. The algorithm tries to identify in the limit the following pieces of information:

- finite sets $E_0, E_1, \ldots, E_m$;
- for each $l \in \{0, 1, \ldots, m\}$, an index j_l such that $A_{j_l} = \mathrm{hull}(E_l)$.

The final conjecture of the algorithm will then be the set $J = \{j_0, j_1, \ldots, j_m\}$.

The algorithm uses the notion of an ϵ-component. Given $\epsilon > 0$, a subset $E \subseteq U$ is an ϵ-component of U if the following two conditions hold:

- For any $x, y \in E$, there is a sequence $z_1, z_2, \ldots, z_h$ of elements of E such that $x = z_1, y = z_h$, and $d(z_l, z_{l+1}) < \epsilon$ for all l with $1 \leq l < h$;
- $d(x, y) \geq \epsilon$ for any $x \in E$ and $y \in U - E$.

Note that for every ϵ and finite set U, the partition of U into ϵ-components is unique.

ALGORITHM M. On input σ, the clusterer goes into the first case applicable from the following:

- If $|I|$ is given and there is a maximal $\epsilon \in \{\frac{1}{|\sigma|}, \frac{2}{|\sigma|}, \ldots\}$ such that $\mathrm{range}(\sigma)$ has exactly $|I|$ ϵ-components, then let $m = |I|$ and $F_0, F_1, \ldots, F_{m-1}$ be these components. For $l = 0, 1, \ldots, m-1$, let E_l be the smallest subset of F_l with $\mathrm{hull}(E_l) = \mathrm{hull}(F_l)$.
- If ϵ is given, then let m be the number of ϵ-components $F_0, F_1, \ldots, F_{m-1}$ of $\mathrm{range}(\sigma)$. For $l = 0, 1, \ldots, m-1$, let E_l be the smallest subset of F_l with $\mathrm{hull}(E_l) = \mathrm{hull}(F_l)$.
- If p is given and there is a number $m \in \{0, 1, \ldots, p-1\}$, an $\epsilon \in \{\frac{1}{|\sigma|}, \frac{2}{|\sigma|}, \ldots\}$, and $E_0, E_1, \ldots, E_{m-1} \subseteq \mathrm{range}(\sigma)$ such that
  - $p = |E_0| + |E_1| + \cdots + |E_{m-1}|$,
  - each set E_l consists of the corners of $\mathrm{hull}(E_l)$, and
  - the sets $\mathrm{hull}(E_l) \cap \mathrm{range}(\sigma)$ are the ϵ-components of $\mathrm{range}(\sigma)$,

  then fix m and the sets $E_0, E_1, \ldots, E_{m-1}$.
- If none of the previous cases hold, then let $m = |\sigma|$ and $E_0, E_1, \ldots, E_{m-1}$ be the m singleton subsets of $\mathrm{range}(\sigma)$.

Now find for each $l \in \{0, 1, \ldots, m-1\}$ the least $s \geq |\sigma|$ such that there is a $j_l \leq s$ with $\mathrm{hull}(E_l) = \mathrm{hull}(A_{j_l,s})$; if there are several candidates for this j_l, then choose the least one. Having found m and $j_0, j_1, \ldots, j_{m-1}$, the output is the set $J = \{j_0, j_1, \ldots, j_{m-1}\}$.

Verification. It is easy to verify that M is computable and is defined on every σ. Fix a clustering task $I \in \mathrm{disj}(S_{\mathrm{conv},k})$ and a text for A_I. In the case of additional information of the second type, let δ be the given lower bound ϵ for γ; otherwise let $\delta = 1$ if $|I| = 1$ and $\delta = \min\{d(A_i, A_j) : i, j \in I \wedge i \neq j\}$ if $|I| > 1$. Assume that a prefix σ of the given text is so long that for each $i \in I$, the following conditions hold:

- For all $j \in \{0, 1, \ldots, i\}$, $\mathrm{hull}(A_{j,|\sigma|}) = A_i$ iff $A_j = A_i$;

- $\mathrm{hull}(\mathrm{range}(\sigma) \cap A_{i,|\sigma|}) = A_i$;
- for all $x \in A_i$, there is a $y \in \mathrm{range}(\sigma)$ such that $d(x,y) < \frac{\delta}{10}$;
- $\frac{1}{|\sigma|} < \frac{\delta}{10}$.

Then one can verify that the algorithm will come up with a lower bound ϵ for δ such that the ϵ-components of $\mathrm{range}(\sigma)$ coincide with the δ-components. Furthermore, the parameter m is the cardinality $|I|$, and the sets $E_0, E_1, \ldots, E_{m-1}$ are sets of minimal cardinality such that

$$\{\mathrm{hull}(E_0), \mathrm{hull}(E_1), \ldots, \mathrm{hull}(E_{m-1})\} = \{A_i : i \in I\}.$$

Since σ is a sufficiently long prefix of the text, the output of the algorithm is a finite set J with $\{A_j : j \in J\} = \{A_i : i \in I\}$. It follows that M solves the clustering task I. □

The last results of the present work deal with conditions under which nonconvex geometrical objects can be clustered. Our first approach is to look at unions of convex objects which are still connected. For $k = 1$, this class is the same as $S_{\mathrm{conv},1}$. But for $k = 2$, this class is larger. There the type of additional information used for clustering $S_{\mathrm{conv},k}$ is no longer sufficient. Given both the number of clusters and the number of vertices as additional information, it is possible to cluster the natural subclass $S_{\mathrm{polygon},2}$ of all classes considered. But if one permits holes inside the clusters, this additional information is no longer sufficient. An alternative parameter is the k-dimensional area covered by a geometric object. In Example 8.8 a natural class $S_{\mathrm{area},k}$ is introduced which can be clustered with the area of a clustering task given as additional information. The class $S_{\mathrm{area},2}$ contains $S_{\mathrm{polygon},2}$ and the class from Example 8.7 as subclasses.

DEFINITION 8.4. *A polygon is given by n vertices $q_1, q_2, \ldots, q_n \in \mathbb{Q}^2$ and is the union of n sides, which are the convex hulls of $\{q_1, q_2\}, \{q_2, q_3\}, \ldots, \{q_n, q_1\}$. The sides do not cross each other, and exactly two sides contain one vertex. Every side has positive length and the angle between the two sides meeting at a vertex is never 0, 180, or 360 degrees. Let $p_0, p_1, \ldots$ be an enumeration of the polygons, and let P_i be the set of all points in $\mathbb{Q}^2$ which are on the polygon p_i or in its interior. Let n_i denote the minimum number of vertices to define the polygon p_i, and let $S_{polygon,2}$ be the class $\{P_0, P_1, \ldots\}$.*

Remark 8.5. Note that every polygon has the same number of sides as vertices. The length 0 of sides and the angle of 180 degrees are forbidden in order to make the representation unique up to some permutation of the vertices. The angles of 0 and 360 degrees are forbidden in order to avoid irregularities.

The following fact will be used below. Assume that $P_i \subseteq P_j$, $n_i \leq n_j$, and every side of p_j contains at least $n_j + 2$ points of P_i. Then $P_i = P_j$. To see this, consider any side T of p_j. Let $c_0, c_1, \ldots, c_{n_j}, c_{n_j+1}$ be $n_j + 2$ points on $T \cap P_i$. These points are all on p_i since they are on p_j and $P_i \subseteq P_j$. There is a $u \in \{0, 1, \ldots, n_j\}$ such that no vertex of p_i is properly between c_u and c_{u+1}. Then the convex hull of $\{c_u, c_{u+1}\}$ is part of a side U_T of P_i. So every side T of p_j has at least two points in common with some side U_T of p_i.

The first claim is that for $T \neq \tilde{T}$, $U_T \neq U_{\tilde{T}}$. So suppose by way of contradiction that there are two sides $T, \tilde{T}$ of P_j such that $U_T = U_{\tilde{T}}$. Let $d_1 \in T \cap U_T$ and $d_2 \in \tilde{T} \cap U_{\tilde{T}}$. Then U_T contains $\mathrm{hull}(\{d_1, d_2\})$ and is a subset of P_j, although not a side of P_j. Since U_T touches two sides of P_j and goes through the interior of P_j, U_T splits P_j into two halves, each of which has some sides different from U_T. On these sides are points of $p_i \cap p_j$, and thus P_i would also be split into two halves by U_T, a

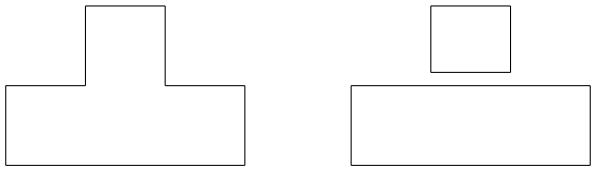


FIG. 1. *Left and right hand clusters. Each has* 8 *vertices and* 8 *sides.*

contradiction. Thus, $n_i = n_j$.

The next claim is that if U_T and $U_{\tilde{T}}$ are neighbors, then so are T and $\tilde{T}$ (and thus, the intersection point of U_T and $U_{\tilde{T}}$ is the same as that of T and $\tilde{T}$). To see this, suppose otherwise. Then the angle $U_T, U_{\tilde{T}}$ splits P_j into two halves, each of which has some sides different from U_T and $U_{\tilde{T}}$. On these sides are points of $p_i \cap p_j$, and thus P_i would also be split into two halves by the angle $U_T, U_{\tilde{T}}$, a contradiction.

It follows from above that $T = U_T$, $U_{\tilde{T}} = \tilde{T}$, and thus $p_i = p_j$.

Note that this property no longer holds if one permits a set of polygons instead of a single polygon. So there is a polygon p_j such that one can find, for any finite subset $F \subseteq P_j$, a set $\{i_1, i_2\} \in \mathrm{disj}(S_{\mathrm{polygon},2})$ with $F \subseteq P_{\{i_1,i_2\}} \subseteq P_j$. More precisely, let p_j be given by $(0,0), (0,1), (1,1), (1,2), (2,2), (2,1), (3,1), (3,0)$, and let F be any finite subset of P_j. Then take $q = \min(\{y : (\exists x)\,[(x,y) \in F \wedge y > 1]\})$ and take i_1, i_2 representing the rectangles given by $(0,0), (0,1), (3,1), (3,0)$ and $(1,q), (1,2), (2,2), (2,q)$.

Figure 1 illustrates the last counterexample. More information on polygons can be found in textbooks on geometry such as [15].

PROPOSITION 8.6. *The class $S_{polygon,2} = \{P_0, P_1, \ldots\}$ is clusterable with additional information in the sense that it is clusterable from the following input provided to a clusterer for clustering task I in addition to a text for P_I: the cardinality $|I|$ and the number $\sum_{i \in I} n_i$. Clustering is impossible if only one of these two pieces of information is available.*

Proof. Assume that the algorithm M knows $|I|$ and $\sum_{i \in I} n_i$ and receives as input a prefix σ of a text for A_I. Then M searches for the $J \subseteq \{0, 1, \ldots, |\sigma|\}$ of least norm which satisfies the following conditions:

1. $J \in \mathrm{disj}(S_{\mathrm{polygon},2})$;
2. $|J| = |I|$;
3. $\sum_{j \in J} n_j = \sum_{i \in I} n_i$;
4. $\mathrm{range}(\sigma) \subseteq P_J$;
5. the vertices of the p_j with $j \in J$ are in $\mathrm{range}(\sigma)$;
6. if T is a side of p_j and $j \in J$, then $|T \cap \mathrm{range}(\sigma)| \geq \sum_{i \in I} n_i + 2$.

M outputs J if J is found, and $\emptyset$ otherwise.

For the verification, it is easy to see that M is recursive. Now consider any clustering task $I \in \mathrm{disj}(S_{\mathrm{polygon},2})$. Since I satisfies the search conditions for all sufficiently long prefixes σ of the text, the clusterer converges to a J with $\mathrm{norm}(J) \leq \mathrm{norm}(I)$, $P_I \subseteq P_J$, $|J| = |I|$, and $\sum_{j \in J} n_j = \sum_{i \in I} n_i$. If $i \in I$, then $P_i \subseteq P_J$. If $P_i \not\subseteq P_j$ for any single $j \in J$, then the P_j with $j \in J$ intersecting P_i would have a positive distance from each other; but since P_i is connected, some points of P_i would not be in any P_j with $j \in J$. Thus, this case cannot happen. Furthermore, if P_j is disjoint from P_I, then the vertices of P_j never show up in the input, and thus $j \notin J$. It follows that there is a one-one correspondence between the $i \in I$ and $j \in J$ such that $P_i \subseteq P_j$. Since $\sum_{j \in J} n_j = \sum_{i \in I} n_i$, there are $i \in I$ and $j \in J$ with $P_i \subseteq P_j$ and $n_i \leq n_j$. Furthermore $P_j \cap P_{I - \{i\}} = \emptyset$, and thus all points of P_j which have

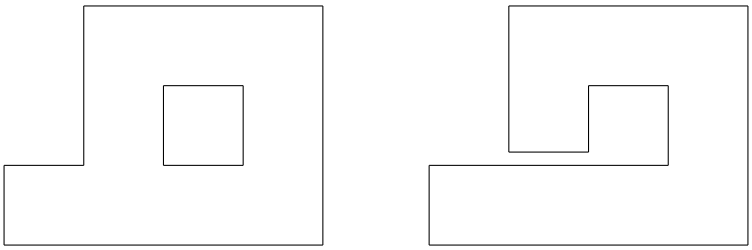


FIG. 2. *Opening a hole while preserving* 10 *vertices and* 10 *sides.*

shown up in the input are actually from P_i. It follows for every side T of p_j that $n_j + 2 \leq |T \cap \text{range}(\sigma)| \leq |T \cap P_i|$. Thus, by Remark 8.5, $P_i = P_j$ and $n_i = n_j$. In particular, there are no $i \in I$, $j \in J$ with $P_i \subseteq P_j$ and $n_i < n_j$. Since $|I| = |J|$ and $\sum_{j \in J} n_j = \sum_{i \in I} n_i$, one can conclude that there are also no $i \in I$, $j \in J$ with $P_i \subseteq P_j$ and $n_i > n_j$. Thus, $n_i = n_j$ whenever $i \in I, j \in J, P_i \subseteq P_j$. By the previous considerations, this gives that $P_i = P_j$ whenever $i \in I, j \in J, P_i \subseteq P_j$. In particular, $P_J = P_I$ and M is a clusterer for $S_{\text{polygon},2}$ which succeeds whenever it receives on the input, in addition to a text for P_I, the numbers $|I|$ and $\sum_{i \in I} n_i$.

Now it is shown that, in addition to the text, the other two pieces of information given to M are needed. That is, M cannot succeed while receiving only one of them.

If only the additional information $|I|$ is given, then consider a stabilizing sequence σ for the rectangle P_i with vertices $(0,0), (0,2), (1,2), (1,0)$. Since $\text{range}(\sigma)$ is finite, there are rationals q_1, q_2 with $0 < q_1 < q_2 < 1$ such that no point of the form (q, r) with $q_1 < q < q_2$ is in $\text{range}(\sigma)$. Thus σ is also a stabilizing sequence for the P_j given by the polygon through the vertices $(0,0), (0,2), (q_1,2), (q_1,1), (q_2,1), (q_2,2), (1,2), (1,0)$ and $P_j \subset P_i$. Thus, the clusterer fails to identify either the clustering task $\{i\}$ or the clustering task $\{j\}$.

If only the additional information $\sum_{i \in I} n_i$ is given, one can take $I = \{i\}$ such that p_i, P_i is given by $(0,0), (0,1), (1,1), (1,2), (2,2), (2,1), (3,1), (3,0)$, and $n_i = 8$ as done in Remark 8.5 (see Figure 1). Now let $\sigma \in P_i^*$ be a stabilizing sequence for P_i and $q = \min(\{y : (\exists x)\,[(x,y) \in \text{range}(\sigma) \wedge y > 1]\})$. Then σ is also a stabilizing sequence for a cluster consisting of the two rectangles which are given as $(0,0), (0,1), (3,1), (3,0)$ and $(1,q), (1,2), (2,2), (2,q)$. See Figure 1 for an illustration. □

EXAMPLE 8.7. *Let $B_{i,j} = p_j \cup (P_i - P_j)$ and $m_{i,j} = n_i + n_j$ if $P_j \subseteq P_i - p_i$; otherwise let $B_{i,j} = P_i$ and $m_{i,j} = n_i$. Let $S_{\text{hole},2}$ consist of all sets $B_{i,j}$. Then it is impossible to cluster $S_{\text{hole},2}$ if besides a text the only pieces of additional information supplied are $|I|$ and $\sum_{(i,j) \in I} m_{i,j}$.*

Proof. The counterexample here is an adaptation of the counterexample from Proposition 8.6. The idea is just to connect the two parts by a bridge and to cutout only the lower connection.

Now take i, j such that the polygons p_i, p_j are given by $(0,0), (0,1), (1,1), (1,3), (4,3), (4,0)$ and $(2,1), (2,2), (3,2), (3,1)$. Note that $m_{i,j} = 10$. Let $\sigma \in B_{i,j}^*$ be a stabilizing sequence for $B_{i,j}$ and

$$q = \min(\{y : (\exists x)\,[(x,y) \in \text{range}(\sigma) \wedge y > 1]\}).$$

Then σ is also a stabilizing sequence for a polygon $P_h \subseteq B_{i,j}$ given by $(0,0), (0,1), (3,1), (3,2), (2,2), (2,q), (1,q), (1,3), (4,3), (4,0)$. The polygon P_h has also 10 vertices and is obtained by connecting the hole with the outside world. Figure 2 illustrates this counterexample. □

Alternatively, one might not restrict the dimension but require that the class under consideration be the union of convex hulls of finite sets which have a positive k-dimensional area. Then this area is a natural parameter for clustering with additional information. Note that in the 2-dimensional case the class $S_{\mathrm{hole},2}$ from Example 8.7 is a subclass of $S_{\mathrm{area},k}$ as defined below, and thus is clusterable using the area as additional information.

EXAMPLE 8.8. *Let $S_{area,k} = \{A_0, A_1, \ldots\}$ be the class of finite unions of members of $S_{conv,k}$ which are connected and have a positive k-dimensional area. Without loss of generality, the set $\{(i,x) : x \in A_i\}$ and the function mapping i to the area of A_i are recursive. Then there is a clusterer for $S_{area,k}$ which uses the area of the members of a cluster as additional information. But $S_{area,k}$ cannot be clustered without additional information.*

Proof. Assume that $A_I \subset A_J$, and let $x \in A_J - A_I$ be given. The point x has a positive distance r from A_I. But the area of $A_J \cap R$, where R is the k-dimensional cube of side-length $\frac{r}{10k}$ with center x, is positive. It follows that the area of A_J is at least the sum of the areas of A_I and $R \cap A_J$. So whenever two sets A_I, A_J have the same area, they are either equal or incomparable. Thus, one can use the following algorithm.

For any given clustering task I, M receives the additional information q and a prefix σ of a text for A_I. Then M outputs the first J such that $J \in \mathrm{disj}_{|\sigma|}(S_{\mathrm{area},k})$, $\mathrm{range}(\sigma) \subseteq A_J$, and A_J has the k-dimensional area q.

It is easy to see that M is recursive and total. Furthermore, M converges to the least J with $\mathrm{norm}(J) = \mathrm{norm}(I)$, A_J having the area q and $A_I \subseteq A_J$. It follows from the arguments above that $A_I = A_J$ and that M satisfies the required properties. □

9. Conclusion. Clustering is a process which makes important use of prior assumptions. Indeed, not every set of points in an underlying space is a potential cluster; for instance, geometric conditions play an important role in the definition of the class of admissible clusters. Whereas such conditions have been taken into account in previous studies, none of those has investigated the consequences of the more fundamental requirement that clustering be a computable process. This paper shows that recursion-theoretic and geometric conditions can both yield substantial insights on whether or not clustering is possible. It also explores the extent to which clustering depends on computational properties by characterizing the power of oracles for clustering. It is expected that further studies of the interaction between topological, recursion-theoretic, and geometrical properties will turn out to be fruitful.

REFERENCES

[1] L. ADLEMAN AND M. BLUM, *Inductive inference and unsolvability*, J. Symbolic Logic, 56 (1991), pp. 891–900.
[2] M. R. ANDERBERG, *Cluster Analysis for Applications*, Academic Press, New York, 1973.
[3] D. ANGLUIN, *Inductive inference of formal languages from positive data*, Inform. and Control, 45 (1980), pp. 117–135.
[4] D. DE JONGH AND M. KANAZAWA, *Angluin's theorem for indexed families of r.e. sets and applications*, in Proceedings of the 9th Annual Conference on Computational Learning Theory, ACM, New York, 1996, pp. 193–204.
[5] R. DUDA, P. HART, AND D. STORK, *Pattern Classification*, 2nd ed., Wiley, New York, 2001.
[6] L. FORTNOW, W. GASARCH, S. JAIN, E. KINBER, M. KUMMER, S. A. KURTZ, M. PLESZKOCH, T. A. SLAMAN, R. SOLOVAY, AND F. STEPHAN, *Extremes in the degrees of inferability*, Ann. Pure Appl. Logic, 66 (1994), pp. 231–276.

[7] R. Freivalds and R. Wiehagen, *Inductive inference with additional information*, Elektron. Informationsverarb. Kybernetik, 15 (1979), pp. 179–185.
[8] M. Fulk, *Prudence and other conditions on formal language learning*, Inform. Comput., 85 (1990), pp. 1–11.
[9] E. M. Gold, *Language identification in the limit*, Inform. Control, 10 (1967), pp. 447–474.
[10] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice–Hall, Englewood Cliffs, NJ, 1988.
[11] S. Jain, D. Osherson, J. Royer, and A. Sharma, *Systems that Learn: An Introduction to Learning Theory*, 2nd ed., MIT Press, Cambridge, MA, 1999.
[12] S. Jain and A. Sharma, *Learning with the knowledge of an upper bound on program size*, Inform. Comput., 102 (1993), pp. 118–166.
[13] S. Jain and A. Sharma, *On the non-existence of maximal inference degrees for language identification*, Inform. Process. Lett., 47 (1993), pp. 81–88.
[14] J. Kleinberg, *An impossibility theorem for clustering*, in Advances in Neural Information Processing Systems 15 (NIPS 2002), MIT Press, Cambridge, MA, 2003, pp. 446–453.
[15] F. Kürpig and O. Niewiadomski, *Grundlehre Geometrie. Begriffe, Lehrsätze, Grundkonstruktionen*, Vieweg, Braunschweig, Germany, 1992.
[16] M. Kummer and F. Stephan, *On the structure of the degrees of inferability*, J. Comput. System Sci., 52 (1996), pp. 214–238.
[17] P. B. Mirchandani and R. L. Francis, eds., *Discrete Location Theory*, Wiley, New York, 1990.
[18] P. Odifreddi, *Classical Recursion Theory*, North-Holland, Amsterdam, 1989.
[19] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, Academic Press, New York, 1998.

SIAM J. COMPUT.
Vol. 36, No. 1, pp. 56–82

FAULT-TOLERANT GATHERING ALGORITHMS FOR AUTONOMOUS MOBILE ROBOTS*

NOA AGMON† AND DAVID PELEG‡

Abstract. This paper studies fault-tolerant algorithms for the problem of gathering N autonomous mobile robots. A gathering algorithm, executed independently by each robot, must ensure that all robots are gathered at one point within finite time. In a failure-prone system, a gathering algorithm is required to successfully gather the nonfaulty robots, independently of the behavior of the faulty ones. Both crash and Byzantine faults are considered. It is first observed that most existing algorithms fail to operate correctly in a setting allowing crash failures. Subsequently, an algorithm tolerant against one crash-faulty robot in a system of three or more robots is presented.

It is then observed that all known algorithms fail to operate correctly in a system prone to Byzantine faults, even in the presence of a single fault. Moreover, it is shown that in an asynchronous environment it is impossible to perform a successful gathering in a 3-robot system, even if at most one of them might fail in a Byzantine manner. Thus, the problem is studied in a fully synchronous system. An algorithm is provided in this model for gathering $N \geq 3$ robots with at most a single faulty robot, and a more general gathering algorithm is given in an N-robot system with up to f faults, where $N \geq 3f + 1$.

Key words. robot swarms, autonomous mobile robots, convergence

AMS subject classifications. 68Q22, 70B15

DOI. 10.1137/050645221

1. Introduction.

Background. Systems of multiple autonomous mobile robots engaged in cooperative activities have been extensively studied throughout the past decade [10, 5, 16, 17, 7, 11, 3, 27]. This subject is of interest for a number of reasons. For one, it may be possible to use a multiple robot system in order to accomplish tasks that no single spatially limited robot can achieve. Another advantage of multiple robot systems has to do with the decreased cost due to the use of simpler and cheaper individual robots. Also, these systems have immediate applicability in a wide variety of tasks, such as military operations and space missions. Subsequently, studies of autonomous mobile robot systems can be found in different disciplines, from engineering to artificial intelligence (e.g., [18, 4, 15, 19]).

Our interest is in problems related to the *distributed control* of systems of autonomous mobile robots. Most studies on robot control problems resulted in the design of algorithms based on heuristics, with little emphasis on formal analysis of the correctness, termination, or complexity properties of the algorithms. During the last few years, various aspects of this problem have been studied from the point of view of distributed computing (cf. [5, 20, 25, 26, 23, 2]), where the focus is on trying to model an environment consisting of mobile robots, and studying the capabilities the robots must have in order to achieve their common goal. A number of computational

*Received by the editors August 4, 2005; accepted for publication (in revised form) December 1, 2005; published electronically May 3, 2006. An extended abstract of this paper appeared in *Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms*, New Orleans, 2004.
http://www.siam.org/journals/sicomp/36-1/64522.html

†Department of Computer Science, Bar-Ilan University, Ramat-Gan, 52900 Israel (segaln@cs.biu.ac.il).

‡Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, 76100 Israel (david.peleg@weizmann.ac.il). The work of this author was supported in part by a grant from the Israel Science Foundation.

models were proposed in the literature, and some studies attempted to characterize the influence of the models on the ability of a group of robots to perform certain basic tasks under different constraints.

The primary motivation of the studies presented in [23, 26, 20, 21, 25] is to identify the minimal capabilities a collection of distributed robots must have in order to accomplish certain basic tasks and produce interesting interaction. Consequently, the models adopted in these studies assume the robots to be relatively weak and simple. In particular, these robots are generally assumed to be dimensionless (namely, treated as points that do not obstruct each other's visibility or movement), oblivious (or memoryless; namely, they do not remember their previous actions or the previous positions of the other robots), lacking a common coordinate system, orientation, or scale, using no explicit communication, and anonymous (some of these assumptions are modified in order to achieve goals that are otherwise unfeasible). They operate in simple "look-compute-move" cycles. Thus the robots base their movement decisions on viewing their surroundings and analyzing the configuration of robot locations. A robot is capable of locating all robots within its visibility range (which can be either limited or unlimited) and laying them in its private coordinate system, thereby calculating their position (distance and angles) with respect to one another and with respect to itself. Hence, from the "distributed computing" angle, such problems are particularly interesting since they give rise to a different type of communication model, based solely on "positional" or "geometric" information exchange.

A basic task that has received considerable attention is the *gathering* problem, defined as follows. Given an initial configuration of N autonomous mobile robots, all N robots should occupy a single point within a finite number of steps. The closely related *convergence* problem is defined similarly, except that the robots are required only to converge to a single point, rather than reach it. Namely, instead of demanding that the robots gather to one point within finite time, the convergence requirement is that for every $\epsilon > 0$, there is a time t_ϵ by which all robots are within distance of at most ϵ of each other.

Fault tolerance. As the common models of multiple robot systems assume cheap, simple, and relatively weak robots, the issue of resilience to failure becomes prominent, since in such systems one cannot possibly rely on assuming fail-proof hardware or software, especially when such robot systems are expected to operate in hazardous or harsh environments. At the same time, one of the main attractive features of multiple-robot systems is their potential for enhanced fault tolerance. It seems plausible that the inherent redundancy of such systems may be exploited in order to enable them to perform their tasks even in the presence of faults.

Following the common "f of N" classification often used in the area, a fault-tolerant algorithm for a given task is required to ensure that in a system consisting of N robots where it is assumed that at most f robots might fail in any execution, the task is achieved by all nonfaulty robots, regardless of the actions taken by the faulty ones. In the gathering task, for example, when faults are introduced into the system, the requirement applies only for the nonfaulty robots; i.e., if f' robots fail, then all the remaining $N - f'$ nonfaulty robots are required to occupy a single point within a finite time.

Perhaps surprisingly, however, this aspect of multiple robot systems has been explored to very little extent so far. In fact, almost all results we are aware of in the literature rely on the assumption that all robots function properly and follow their protocol without any deviation. One exception concerns *transient* failures. As

observed in [26, 23, 13], any algorithm that works correctly on oblivious robots is necessarily *self-stabilizing*; i.e., it guarantees that after any transient failure the system will return to a correct state and the goal will be achieved. Yet another line of study concerns a fault model where it is assumed that restricted sensor and control failures might occur, but if faults do occur in the system, then the identity of the faulty robots becomes known to all robots [23]. This may be an unrealistic assumption in many typical settings, and it clearly provides an easy means of overcoming the faults: Each nonfaulty robot may simply ignore the failed ones, effectively removing them from the group of robots, so the algorithm continues to function properly. However, in case unidentified faults occur in the system, it is no longer guaranteed that the algorithms of [23, 26] remain correct; i.e., the goal might not be achieved. The only concrete attempt we are aware of for dealing with crash faults is described in [29], where an algorithm is given for the *active robot selection problem* (ARSP) in the presence of initial crash faults. The ARSP creates a subgroup of nonfaulty robots from a group that also includes initially crashed robots and makes the robots in that subgroup recognize one another. This allows the nonfaulty robots in the subgroup to overcome the existence of faults in the system, and they can further execute any algorithm within the group.

Hence the design of fault-tolerant distributed control algorithms for multiple robot systems is still a largely unexplored direction, which the current paper investigates.

Related work. A number of basic mobile robot coordination problems were considered in the literature. One class of problems involves the formation of geometric patterns. The robots are required to arrange themselves in a given geometric form, such as a circle, a simple polygon or a line, within finite time (see, e.g., [23, 9, 12]). The task of *flocking*, requiring the robots to follow the movement of a predefined leader, was studied in [22]. The even distribution problem, requiring the robots to spread out uniformly over a specified region of a simple geometric shape, and the related task of partitioning the robots into groups were studied in [23].

The problem of gathering autonomous mobile robots, dealt with in this paper, requires the robots to gather to the same point within finite time (see, e.g., [24, 25, 14, 7, 6, 8]). This problem was studied extensively in two computational models. The first is the model of [23, 26], hereafter referred to as the *semisynchronous* ($\mathcal{SSYNC}$) model. The second is the closely related CORDA model [20, 21, 25], hereafter referred to as the *asynchronous* ($\mathcal{ASYNC}$) model.

The gathering problem was first discussed in [25, 26] in the $\mathcal{SSYNC}$ model. It was proved there that it is impossible to achieve gathering of *two* oblivious autonomous mobile robots that have no common sense of orientation under the $\mathcal{SSYNC}$ model. The algorithms presented therein for $N \geq 3$ robots rely on the assumption that a robot can identify a point p^* occupied by two or more robots (also known as *multiplicity point*). This assumption was later proved to be essential for achieving gathering in all asynchronous and semisynchronous models [22]. Another necessary requirement for solvability in the $\mathcal{SSYNC}$ and $\mathcal{ASYNC}$ models is that the input configuration does not include more than one multiplicity point of nonfaulty robots (it is easy to show that if two multiplicity points of nonfaulty robots are allowed, the situation is equivalent to the 2-robot system, and thus gathering is impossible). In fact, all known gathering algorithms for $N \geq 3$ rely on a strategy by which a single multiplicity point p^* is formed during the execution of the algorithm, and once this happens, all robots move to the point p^*. Under these assumptions, an algorithm is developed in [26] for gathering $N \geq 3$ robots in the $\mathcal{SSYNC}$ model. In the $\mathcal{ASYNC}$ model, an algorithm

for gathering $N = 3, 4$ robots is brought in [22, 7], and an algorithm for gathering $N \geq 5$ robots is described in [6].

The gathering problem was also studied (in both the $\mathcal{SSYNC}$ and $\mathcal{ASYNC}$ models) in a system where the robots have limited visibility. The visibility conditions are modelled by means of a *visibility graph*, representing the (symmetric) visibility relation of the robots with respect to one another; i.e., an edge exists between R_i and R_j if and only if R_i and R_j are visible to each other. It was shown that the problem is unsolvable in the case that the visibility graph is not connected [14]. In [1] a convergence algorithm was provided for any N in limited visibility systems. An algorithm that achieves gathering in the $\mathcal{ASYNC}$ model is described in [14], under the assumption that all robots share a compass (i.e., agree on a direction in the plane).

Our results. This paper presents a systematic study of failure-prone robot systems through examining the gathering problem under the crash and Byzantine fault models. An (N, f)-*fault system* is a system consisting of N robots, of which at most f might fail at any execution. An (N, f)-*crash system* (resp., (N, f)-*Byzantine system*) is an (N, f)-fault system where the faults considered are according to the crash or Byzantine model. A fault-tolerant algorithm for a given task in an (N, f)-fault system is required to ensure that so long as at most f robots have failed, the task is achieved by all nonfaulty robots, regardless of the actions taken by the faulty ones.

Under the crash fault model, we show that the gathering problem is solvable in current computational models such as the $\mathcal{SSYNC}$ model, though most existing algorithms fail to deal correctly with such faults and, in particular, there is currently no algorithm for $N \geq 4$ that solves the gathering problem in the presence of one faulty robot under the crash fault model. We propose an algorithm that solves the gathering problem in an $(N, 1)$-crash system, for any $N \geq 3$, under the $\mathcal{SSYNC}$ model.

We then consider (N, f)-Byzantine systems for $N \geq 3$. We first observe that all existing algorithms fail to deal correctly with this situation. Moreover, we show that it is impossible to perform a successful gathering in $(3, 1)$-Byzantine systems under the $\mathcal{SSYNC}$ model. We then introduce the *fully synchronous* ($\mathcal{FSYNC}$) model, which is similar to the synchronous model mentioned in [26], and present an algorithm solving the gathering problem under this model in (N, f)-Byzantine systems for every[1] $N \geq 3f + 1$.

2. The model. We follow the common computational model of distributed robot systems. In particular, we make the following assumptions: The visibility range of the robots is assumed to be unlimited. The robots are treated as points (dimensionless objects) which do not obstruct each other's visibility or movement. The robots are anonymous and cannot communicate with each other. For the sake of analysis, denote the robots in the system by $R_1, \ldots, R_N$. Each robot R_i has its private coordinate system, consisting of the position of the origin, direction of the positive x-axis, and the size of one unit distance. It is assumed that the direction of the positive y-axis is 90° counterclockwise of the direction of the positive x-axis. The coordinate systems of the various robots might all be different and not share the same direction or scale.

Following most previous papers on the gathering problem in the literature [24, 25, 14, 7, 22], the model adopted throughout this paper is the *oblivious* model, where it is assumed that the robots cannot remember their previous states, and thus the

[1] A peculiarity of our algorithms is that $N = 3$ robots can tolerate $f = 1$ failures, but $N > 3$ robot systems require $N \geq 3f + 1$ rather than $N \geq 3f$.

decisions they make in each step are based only on the current configuration. The main motivation for developing algorithms for the oblivious model is twofold. First, solutions developed on the basis of assuming nonobliviousness do not necessarily work in a dynamic environment where the robots are activated in different cycles or might be added/removed from the system dynamically. Second, as mentioned earlier, any algorithm that works correctly for oblivious robots is inherently self-stabilizing, i.e., it withstands transient errors. More generally, it is advantageous to develop algorithms for the weakest robot types possible, as an algorithm that works correctly for weak robots will clearly work correctly in a system of stronger robot types. In contrast, our lower bounds serve mainly to draw the borderlines where the various models become too weak to allow solutions.

Robot operation cycle. Each robot R_i in the system is assumed to operate individually in simple cycles. Every cycle consists of three steps, Look, Compute, and Move. In the $\mathcal{FSYNC}$ and $\mathcal{SSYNC}$ models the length of this cycle is uniform for all robots.

- *Look*: Identify the locations of all robots in R_i's private coordinate system; the result of this step is a multiset of points $P = \{p_1, \ldots, p_N\}$ defining the current *configuration*. As the robots are indistinguishable, each robot R_i knows its own location p_i but does not know the identity of the robots at each of the other points.
- *Compute*: Execute the given algorithm, resulting in a goal point p_G.
- *Move*: Move towards the point p_G. The robot might stop before reaching its goal point p_G but is promised to traverse a distance of at least S (unless it has reached the goal).

Note that the Look and Move steps are carried out identically in every cycle, independently of the algorithm used. The differences between different algorithms occur in the Compute step. Moreover, the procedure carried out in the Compute step is identical for all robots. If the robots are oblivious, then the algorithm cannot rely on information from previous cycles; thus the procedure can be fully specified by describing a single Compute step, and its only input is the current configuration $P = \{p_1, \ldots, p_N\}$, giving the robot locations. Throughout, we may denote the location of R_i in the configuration P by $p(R_i)$. Also, whenever no confusion may arise, we identify $p(R_i)$ as the point p_i.

Three synchronization models. As mentioned earlier, our computational model for studying and analyzing problems of coordinating and controlling a set of autonomous mobile robots follows two well-studied models: the $\mathcal{SSYNC}$ model and the $\mathcal{ASYNC}$ model. The semisynchronous ($\mathcal{SSYNC}$) model is partially synchronous, in the sense that all robots operate according to the same clock cycles, but not all robots are necessarily active in all cycles. The activation of the different robots can be thought of as managed by a hypothetical "scheduler," whose only "fairness" obligation is that each robot must be activated and given a chance to operate infinitely often in any infinite execution. The fully asynchronous ($\mathcal{ASYNC}$) model differs from the $\mathcal{SSYNC}$ model in that each robot acts independently in a cycle composed of *four* steps: Wait, Look, Compute, Move. The length of this cycle is finite, but not bounded. Consequently, there is no bound on the length of the walk in a single cycle, and different cycles of the same robot may vary in length. In contrast, in the $\mathcal{SSYNC}$ model a bound exists on the cycle length due to the common clock, and as a result the robot will not necessarily reach the target point p in the current cycle but stop somewhere on its trajectory to p.

In this paper we also consider the extreme *fully synchronous* ($\mathcal{FSYNC}$) model.

This model is similar to the $\mathcal{SSYNC}$ model, where the robots operate according to the same clock cycles, except that here all robots are active on all cycles. In this model we assume discrete time $0, 1, \ldots$, and let $p_i(t)$ denote the position of R_i at time t, where $p_i(0)$ is the initial position of R_i. In each cycle t, the set of positions is a multiset, as two robots are not prohibited from occupying the same position simultaneously. In each cycle t, each robot R_i is capable of moving over distance at least $S > 0$ in one step (S is unknown to the robots). Therefore it is guaranteed that if $\text{dist}(p_G^i, p_i(t)) \leq S$, where p_G^i is R_i's goal point in the current cycle, then R_i will reach its goal in the current cycle. Otherwise, it will traverse a distance of at least S towards p_G^i.

Failure models. The fault models discussed throughout the paper are the *crash* fault model and the *Byzantine* fault model. In the *Byzantine* fault model, it is assumed that a faulty robot might behave in arbitrary and unforeseeable ways. For the sake of analysis, it is convenient to model the behavior of the system by means of an *adversary* which has the ability to control the behavior of the faulty robots, as well as the "undetermined" features in the behavior of the nonfaulty processors (e.g., the distance to which they move). Specifically, in each cycle the adversary has the following roles. For each faulty robot, it determines its course of action in that cycle, which can be arbitrary. For each nonfaulty robot, it determines the distance to which the robot will move in this cycle (for a robot R_i located at p_i and headed for the goal point p_G, if $\text{dist}(p_i, p_G) \leq S$, then the robot must be allowed to reach p_G; else, the adversary may stop R_i at any point on the line segment $\overline{p_i p_G}$ that is at least distance S away from p_i).

In the *crash* fault model, the behavior of the system is similar to the one described in the Byzantine fault model, except that for each faulty robot the adversary is only allowed to stop its movement. This may be done at any point in time during the cycle, i.e., either during the movement toward the goal point or before it has started. Once the adversary has crashed the faulty robot, that robot will remain stationary indefinitely.

3. Gathering under the crash fault model.

3.1. Inadequacy of known algorithms. Most gathering algorithms proposed in the literature fail to withstand even a single crash failure because they depend, in certain configurations, on the movement of a single robot. More formally, let $\mathcal{A}$ be a gathering algorithm for an N-robot system. In every configuration C, the algorithm instructs some robots to move and some to remain stationary. Denote the number of robots $\mathcal{A}$ instructs to move in configuration C by $M(C, \mathcal{A})$, and let

$$\check{M}(N, \mathcal{A}) \;=\; \min\{M(C, \mathcal{A}) \mid C \text{ is a configuration in an } N\text{-robot system}\}.$$

LEMMA 3.1. *In an (N, f)-crash system, an algorithm $\mathcal{A}$ with $\check{M}(N, \mathcal{A}) \leq f$ will fail in achieving gathering or convergence.*

Proof. Consider an (N, f)-crash system and a gathering algorithm $\mathcal{A}$ with $\check{M}(N, \mathcal{A}) \leq f$, and let C' be the configuration of the system realizing $\check{M}$, i.e., such that $M(C', \mathcal{A}) \leq f$. Starting in that configuration, the adversary can fail the (f or fewer) robots instructed by the algorithm to move. This will cause the next configuration to be identical to C' again, and the $N - f$ nonfaulty robots will remain in the same configuration C' indefinitely. □

In fact, every gathering algorithm $\mathcal{A}$ we are aware of in the $\mathcal{SSYNC}$ and $\mathcal{ASYNC}$ models [24, 25, 26, 7] has $\check{M}(N, \mathcal{A}) = 1$ for $N \geq 4$, and, consequently, by Lemma 3.1,

these algorithms fail to achieve gathering even in the presence of one crash faulty robot. The algorithm described in [6] for gathering $N \geq 5$ robots in the $\mathcal{ASYNC}$ model can also fail in the presence of one crash faulty robot if that robot lies between some other robot and its goal point. On the positive side, it turns out that the gathering algorithm given in [7] for $N = 3$ under the $\mathcal{ASYNC}$ model can be shown to operate correctly also in the presence of one crashed robot (we give a slightly simpler algorithm for this case in the $\mathcal{SSYNC}$ model below), and the algorithm given therein for $N = 4$ can be transformed into an algorithm for $(4, 1)$-crash systems with some minor changes.

An additional difficulty in handling a robot system with crash faults is caused by the assumption, made by all current algorithms, that only a single multiplicity point is created throughout the execution of the algorithm. In the presence of faults, the fact that the adversary has the ability to stop the nonfaulty robots after traversing a minimal distance S and the ability to crash the faulty robots at any step during the execution makes it easy for the adversary to create a second multiplicity point once the first is created, whenever the trajectories of two or more robots moving towards their goals intersect. In particular, it is easy to see that in a collinear configuration with $N > 3$ robots and given an algorithm that instructs all robots to move towards a point on the line, the adversary can create two multiplicity points on the line, simply by crashing some robot R at a point p between the multiplicity point and another robot R', thus forcing R' to pass through p, and stopping it there.

3.2. An algorithm for a $(3, 1)$-crash system. Consider the following Procedure $\mathsf{3\text{-}Gather}_{\mathrm{crash}}$ for gathering in a $(3, 1)$-crash system in the $\mathcal{SSYNC}$ model. As discussed earlier, we need only present the procedure used for the Compute step. The input to this procedure is the configuration $P = \{p_1, p_2, p_3\}$. The procedure classifies the configuration according to its state, and acts in each case as follows.

Procedure $\mathsf{3\text{-}Gather}_{\mathrm{crash}}(P)$

1. **State [MULT]:** P contains a multiplicity point p^*:
   Set $p_G \leftarrow p^*$.
2. **State [Collinear]:** p_1, p_2, p_3 are collinear (say, with p_2 in the middle):
   Set $p_G \leftarrow p_2$.
3. **State [Obtuse]:** $\exists i \in \{1, 2, 3\}$ such that $\angle p_j p_i p_k \geq \pi/2$:
   Set $p_G \leftarrow p_i$.
4. **State [Acute]:** p_1, p_2, p_3 form an acute triangle:
   Set p_G to be the intersection point of the three angle bisectors.

Note that state [Collinear] is redundant, since it is covered by state [Obtuse]. It is included merely for convenience of presentation.

Analysis. In analyzing our algorithms, we use the following notation regarding points and lines in the Euclidean plane. Denote the Euclidean distance between two points p and q by $\mathrm{dist}(p, q)$. Also, denote the Euclidean distance between two current locations p_i and p_j of the two robots R_i and R_j, respectively, by $\mathrm{dist}(R_i, R_j)$. Denote the line segment between the points p and q by $\overline{pq}$. We use the following well-known fact.

LEMMA 3.2. *In a triangle $\triangle p_1 p_2 p_3$, the intersection point p_M of the three angle bisectors satisfies $\angle p_i p_M p_j \geq \pi/2$ for every $1 \leq i < j \leq 3$.*

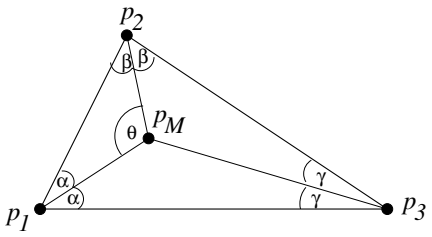


FIG. 1. *Proof of Lemma* 3.2.

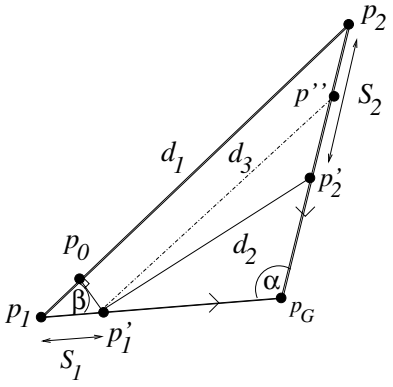


FIG. 2. *Proof of Lemma* 3.3.

Proof. Let $\alpha = \angle p_M p_1 p_2$, $\beta = \angle p_M p_2 p_3$, $\gamma = \angle p_M p_3 p_1$, and $\theta = \angle p_1 p_M p_2$ (see Figure 1). It follows that $2\alpha + 2\beta + 2\gamma = \pi$; hence $\alpha + \beta = \pi/2 - \gamma < \pi/2$, and since in the triangle $\triangle p_M p_1 p_2$, $\theta + \alpha + \beta = \pi$, it follows that $\theta > \pi/2$. A similar argument applies in $\triangle p_M p_2 p_3$ and $\triangle p_M p_1 p_3$. ∎

LEMMA 3.3. *If two robots R_1 and R_2, initially located at the points p_1 and p_2, respectively, move towards a common meeting point p_G and $\alpha = \angle p_1 p_G p_2 \geq \pi/2$, then the distance between them decreases by at least* $0.7S$.

Proof. For $i = 1, 2$, let S_i denote the distance traversed by the robot R_i, and let p_i' denote the new location of R_i. Consider the triangle $\triangle p_1 p_2 p_G$, and let $\beta = \angle p_2 p_1 p_G$ (see Figure 2). Denote the distance between the robot locations before and after the movement by $d_1 = \text{dist}(p_1, p_2)$ and $d_2 = \text{dist}(p_1', p_2')$, respectively. Without loss of generality, suppose that p_1' is closer than p_2' to the line $\overline{p_1 p_2}$. Draw a line parallel to d_1 through p_1', denote its intersection with $\overline{p_2 p_G}$ by p'', and let $d_3 = \text{dist}(p_1', p'')$.

We need to prove that $d_1 - d_2 > 0.7S$. Since it is clear that $d_2 \leq d_3$, it suffices to show that $\Delta = d_1 - d_3 > 0.7S$. Drop a perpendicular line from p_1' to $\overline{p_1 p_2}$, and let p_0 be its intersection point with the line $\overline{p_1 p_2}$. Let $\Delta' = \text{dist}(p_0, p_1)$. It is also clear that $\Delta' \leq \Delta$; hence it remains to prove that $\Delta' \geq 0.7S$. Since $\alpha \geq \pi/2$, the remaining two angles in $\triangle p_1 p_2 p_G$ sum to at most $\pi/2$. Without loss of generality, let $\beta \leq \pi/4$. Also, according to our model assumption, each robot moves a distance of at least S in each cycle, i.e., $S_i \geq S$ for $i = 1, 2$. Therefore, by the sine theorem on the triangle $\triangle p_1 p_1' p_0$ (see Figure 3), $S \leq S_1 = \frac{S_1}{\sin(\pi/2)} = \frac{\Delta'}{\sin(\pi/2 - \beta)}$, and hence $\Delta' \geq S \cdot \cos(\beta) \geq S \cdot \cos(\pi/4) \geq 0.7S$, completing the proof. ∎

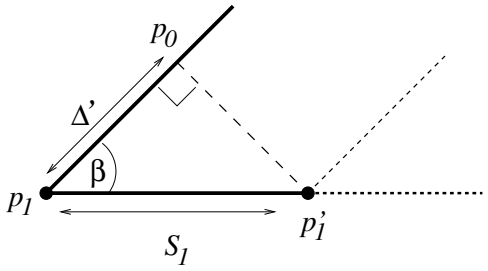


FIG. 3. *Triangle enlargement from proof of Lemma* 3.3.

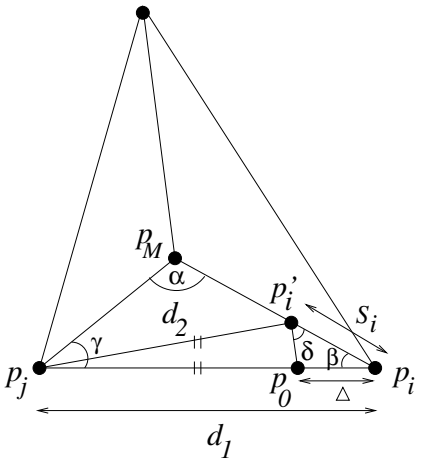


FIG. 4. *Proof of Lemma* 3.4.

LEMMA 3.4. *There exists a constant $c > 0$ such that, given three robots R_1, R_2, and R_3 located at points p_1, p_2, and p_3, respectively, where $\triangle p_1p_2p_3$ is an acute triangle, if one or more of the robots traverses a distance of at least S towards the intersection point p_M of the three angle bisectors, then the circumference of the triangle decreases by at least cS.*

Proof. By Lemma 3.2, $p_ip_Mp_j > \pi/2$ for every $1 \leq i, j \leq 3$. Thus, by Lemma 3.3, if two of the three robots move towards p_M, then the distance between them decreases by at least $0.7S$, and as the other distances do not increase, the circumference of the triangle decreases by at least $0.7S$. It remains to show that even if only one robot moves towards p_M it decreases the distance between itself and its neighbors by at least $c'S$ for some constant c'; thus altogether the circumference of the triangle decreases by at least $2c'S = cS$.

Consider the triangle $\triangle p_ip_Mp_j$, $1 \leq i, j \leq 3$. Let $\alpha = \angle p_ip_Mp_j$, $\beta = \angle p_Mp_ip_j$, and $\gamma = \angle p_Mp_jp_i$ (see Figure 4). Since $\triangle p_1p_2p_3$ is acute, it follows that $2\beta < \pi/2$; thus $\beta < \pi/4$ and, similarly, $\gamma < \pi/4$. Assume, without loss of generality, that robot R_i traversed a distance $S_i \geq S$ towards p_M and lies on a point p'_i. Let $d_1 = \text{dist}(p_i, p_j)$, $d_2 = \text{dist}(p'_i, p_j)$, and $\Delta = d_1 - d_2$. By the sine theorem on the triangles $\triangle p_ip_jp_M$ and $\triangle p'_ip_jp_M$ it follows that $\frac{d_1}{d_2} = \frac{\sin(\angle p_jp'_ip_M)}{\sin\beta}$, and since $\angle p_jp'_ip_M > \beta$, it follows that $d_1 > d_2$. Let p_0 be the point on the segment $\overline{p_ip_j}$ that creates an isosceles triangle

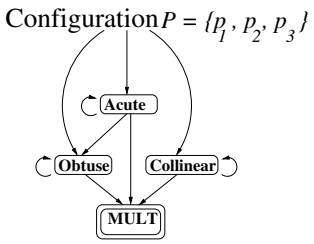


FIG. 5. *Statechart for Procedure* 3-Gather$_{\text{crash}}$.

$\triangle p_0 p_j p_i'$; i.e., $\text{dist}(p_j, p_0) = d_2$. Note that $\text{dist}(p_0, p_i) = \Delta$. Let $\delta = \angle p_i p_i' p_0$. Since

$$3\pi/8 < \frac{\pi - \gamma}{2} < \frac{\pi - \angle p_i p_j p_i'}{2} = \angle p_j p_0 p_i' = \angle p_j p_i' p_0 < \pi/2,$$

it follows that $3\pi/8 < \beta + \delta < \pi/2$, and as $\beta < \pi/4$, we have $\pi/8 < \delta < \pi/2$. By the sine theorem on the triangle $\triangle p_i p_i' p_0$ it follows that

$$\Delta = S_i \frac{\sin\delta}{\sin(\delta+\beta)} \geq S \frac{\sin\delta}{\sin(\delta+\beta)} \geq S \frac{\sin(\pi/8)}{\sin(\pi/2)} > 0.3S.$$

Therefore by choosing $c' = 0.3$, the lemma holds for $c = 0.6$. ∎

THEOREM 3.5. *Algorithm* 3-Gather$_{\text{crash}}$ *solves the gathering problem in a* $(3,1)$*-crash system under the* $\mathcal{SSYNC}$ *model.*

Proof. Consider an initial configuration $P = \{p_1, p_2, p_3\}$. It suffices to show that the algorithm causes the system to reach state [MULT]; i.e., either it gathers all robots together in one point or it causes the creation of one multiplicity point, since if the remaining robot is nonfaulty, then it will join the multiplicity point in finite time by step 1 of the algorithm, and if it is faulty, then gathering has been achieved. Consider the flow of states the system could be in. It suffices to show that the states used for classifying the configurations in Procedure 3-Gather$_{\text{crash}}$ form a finite connected directed acyclic graph (DAG) (possibly with self-loops), where all paths lead to a final state [MULT] in which a multiplicity point exists (see Figure 5), such that starting with a configuration in any of the states, we reach the final state within a finite number of cycles.

If in the initial configuration p_1, p_2, p_3 are collinear, then the configuration will remain collinear, and within finite time, either both extreme robots will arrive at the location of the middle robot (if both are nonfaulty) or only one of them will arrive (if one of the extreme robots is faulty); thus in any case a multiplicity point will be created in the location of the middle robot, leading to state [MULT].

Next, suppose that the initial configuration is not collinear but obtuse; i.e., there exists a point p_i such that $\angle p_j p_i p_k \geq \pi/2$. Then the configuration remains obtuse until one robot reaches R_i. Since at least one of the robots instructed to move towards R_i is nonfaulty, it will reach its goal point within finite time, thus reaching state [MULT].

Finally, if p_1, p_2, p_3 create an acute triangle in cycle t, then one of the following two cases holds. In the first case, at least one robot was active in the current cycle and traversed a distance of at least S towards p_M. There are two subcases to be examined. If the system remains in state [Acute], then by Lemma 3.4 the circumference of $\triangle p_1 p_2 p_3$ decreases by at least $0.6S$. Therefore, if the robots constantly remain in state [Acute], then at least two robots will eventually meet in p_M, leading to state [MULT]. The other subcase is that this movement causes $\triangle p_1 p_2 p_3$ to become

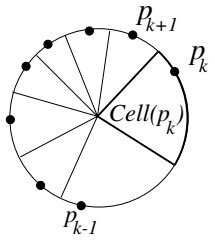


FIG. 6. *Example of a circle division according to the Voronoi cells.*

obtuse. In this case, the system changes to state [Obtuse]. In the second case, all active robots in this cycle traversed a distance smaller than S towards p_M; thus they are now located at p_M. Again there are two subcases. If two or more robots were active, then they meet at p_M, leading to state [MULT]. Otherwise, only one robot, say R_i, is located in p_M. Then by Lemma 3.2, $\angle p_j p_i p_k \geq \pi/2$; thus $\triangle p_1 p_2 p_3$ becomes obtuse, and the system changes to state [Obtuse], leading to gathering as discussed above. □

3.3. An algorithm for an $(N, 1)$-crash system, $N \geq 3$. Let us start with some terminology. A *legal* configuration in the $(N, 1)$-crash system is a set P of robot locations that has at most one multiplicity point. Denote the *smallest enclosing circle* of the set P by $\mathcal{SEC}(P)$ and the points on its circumference by $\mathcal{C}_{cir}(P)$ and let $\mathcal{C}_{\rm int}(P) = P \setminus \mathcal{C}_{\rm cir}(P)$. For a circle C and the set of points $P = \{p_1, \ldots, p_l\}$ on its circumference, denote the partition of the circle C into Voronoi cells according to the points in P by $\mathrm{Vor}(C, P)$, and denote by $\mathrm{Cell}(p_k)$ the cell defined by the point $p_k \in P$ (see Figure 6). Two points q and q' in C are said to share the cell $\mathrm{Cell}(p_k)$ if they both lie inside the cell or on its boundary.

Consider the following Algorithm $\mathsf{Gather}_{\rm crash}$ for gathering all nonfaulty robots in an $(N, 1)$-crash system under the $\mathcal{SSYNC}$ model. The input to this algorithm is a legal configuration $P = \{p_1, \ldots, p_N\}$. The algorithm classifies the configuration according to its state and acts in each case as follows. If there are no multiplicity points in the configuration, then each robot performs Procedure $\mathsf{Create\_Mult}$ in order to reach a configuration with a multiplicity point. Figure 7 illustrates the three possible cases of substate [IN2] in this procedure. Once a multiplicity point p^* is detected, each robot performs Procedure $\mathsf{GoTo\_Mult}$ in order to achieve gathering of all nonfaulty robots in p^*, while avoiding creation of additional multiplicity points.

Algorithm $\mathsf{Gather}_{\rm crash}(P)$

1. **State [Singletons]:** The configuration P does not contain a multiplicity point: Invoke Procedure $\mathsf{Create\_Mult}(P)$.
2. **State [MULT]:** The configuration P contains a single multiplicity point p^*: Invoke Procedure $\mathsf{GoTo\_Mult}(P)$.

The input to Procedure $\mathsf{GoTo\_Mult}$ is the configuration $P = \{p_1, \ldots, p_N\}$. We say that robot R_i has a *"free corridor"* to the point p if no other robot is currently located on the straight line segment $\overline{p_i p}$. Note that as robots are viewed as dimensionless objects, the availability of a free corridor is not necessarily a prerequisite for allowing a robot to get home free. However, allowing a robot to follow a trajectory through the location of another robot makes the algorithm prone to the creation of more

Procedure Create_Mult(P)
State [N3]: $N = 3$:
Invoke Procedure 3-Gather$_{\text{crash}}$ on p_1, p_2, p_3.
State [N4+]: $N \geq 4$:

1. **Substate [IN0]:** $|\mathcal{C}_{\text{int}}(P)| = 0$:
   Set p_G to be the center of $\mathcal{SEC}(P)$.
2. **Substate [IN1]:** $|\mathcal{C}_{\text{int}}(P)| = 1$ with p_j as the single point in $\mathcal{C}_{\text{int}}(P)$:
   Set $p_G \leftarrow p_j$.
3. **Substate [IN2]:** $|\mathcal{C}_{\text{int}}(P)| = 2$ with p_i and p_j as the two points in $\mathcal{C}_{\text{int}}(P)$:
   Each robot R_k in $\mathcal{C}_{\text{cir}}(P)$ sets $p_G(R_k) \leftarrow p(R_k)$.
   The two robots R_i and R_j in $\mathcal{C}_{\text{int}}(P)$ do:
   (a) Compute the Voronoi partition $\text{Vor}(\mathcal{SEC}(P), \mathcal{C}_{cir}(P))$.
   (b) **Substate [IN2(a)]:** p_i and p_j do not share cells:
   R_i and R_j move towards the center of $\mathcal{SEC}(P)$.
   (c) **Substate [IN2(b)]:** p_i and p_j share a single cell, Cell(R_k):
   R_i and R_j move towards R_k.
   (d) **Substate [IN2(c)]:** p_i and p_j share two cells; i.e., both robots lie on the radius forming the boundary between two adjacent cells Cell(R_k) and Cell(R_{k+1}):
   The robot closer to the circle, say R_i, chooses the first of R_k, R_{k+1} in its clockwise direction, say R_k, and sets $p_G(R_i) \leftarrow p(R_k)$.
   The other robot, R_j, sets $p_G(R_j) \leftarrow p(R_i)$.
4. **Substate [IN3]:** $|\mathcal{C}_{\text{int}}(P)| \geq 3$:
   Each robot R_k in $\mathcal{C}_{\text{cir}}(P)$ sets $p_G(R_k) \leftarrow p(R_k)$.
   Each robot R_k in $\mathcal{C}_{\text{int}}(P)$ recursively invokes Procedure Create_Mult($\mathcal{C}_{\text{int}}(P)$).

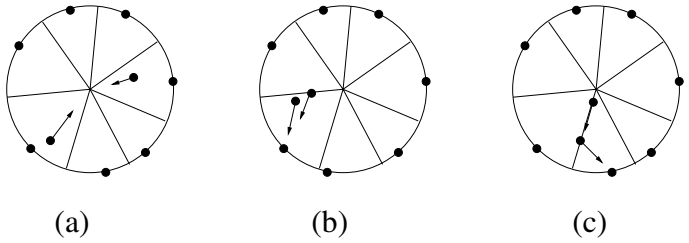


FIG. 7. *Illustration of the three substates of substate* [IN2] *in Procedure* Create_Mult.

than one multiplicity point. Therefore Procedure GoTo_Mult attempts to avoid such trajectories.

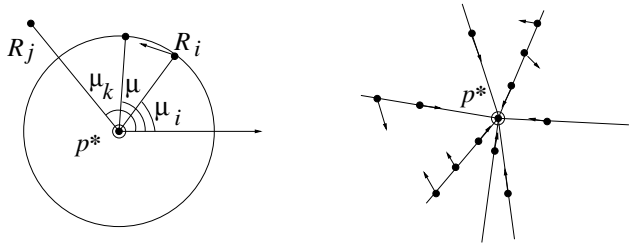
Analysis.

LEMMA 3.6. *If the initial configuration is in state* [Singletons]*, i.e., it contains no multiplicity points, then Procedure* Create_Mult *leads, within finite time, to a configuration in state* [MULT]*, i.e., including a single multiplicity point.*

Proof. We prove the lemma by looking at the flow of states the system could be in. It suffices to show that the states used for classifying the configurations in Procedure Create_Mult form a finite connected DAG (possibly with self-loops), where all paths lead to a final state [MULT] in which a multiplicity point exists (see Figure 9), such that starting with a configuration in any of the states, we reach the final state within a finite number of cycles.

Procedure GoTo_Mult(P) (for robot R_i)
/* The configuration contains a multiplicity point p^* */

1. **State [Free]:** R_i has a free corridor to p^*:
   Set $p_G \leftarrow p^*$.
2. **State [Blocked]:** There exist one or more robots on R_i's trajectory towards p^*:
   a. Translate your coordinate system to be centered at p^*.
   b. Compute for each robot R_j the angle μ_j of $\overrightarrow{p^*p_j}$ counterclockwise from the x-axis.
   c. Find the robot R_k with smallest angle $\mu_k > \mu_i$.
   Let $\mu = (\mu_k + 2\mu_i)/3$, and $d = \text{dist}(R_i, R_k)$ (see Figure 8(a)).
   d. Let p'_i be the point at distance d and angle μ from p^*.
   e. Set $p_G \leftarrow p'_i$.



(a) (b)

FIG. 8. *Illustration of state* [Blocked] *in Procedure* GoTo_Mult.

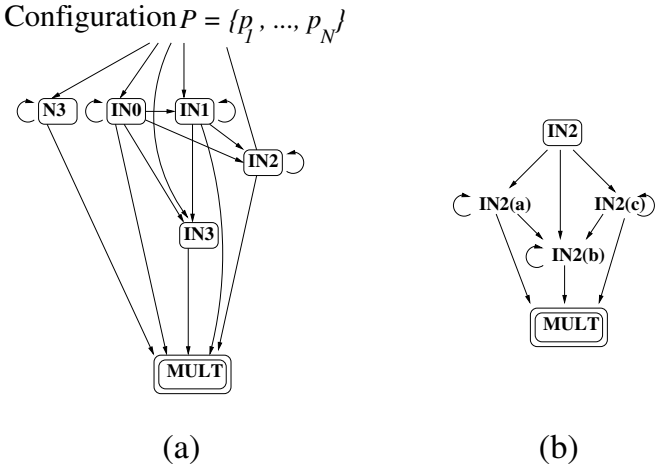


FIG. 9. *Statechart for Procedure* Create_Mult. (a) *The general statechart.* (b) *The substates of state* [IN2].

State [N3]: If $N = 3$, then by Theorem 3.5, Procedure 3-Gather$_{\text{crash}}$ achieves gathering, and in particular one multiplicity point is created.

State [IN0]: If $N \geq 4$ and $|\mathcal{C}_{\text{int}}(P)| = 0$, then if not all robots move together

to the same distance, the system changes either to state [MULT] or to a state where $|\mathcal{C}_{\text{int}}(P)| \geq 1$, i.e., [IN1], [IN2], or [IN3]. If robots on the circumference move to a new configuration P' in which they are all again on $\mathcal{SEC}(P')$, then the system remains in the same state, [IN0], but the radius of $\mathcal{SEC}(P)$ is reduced by at least S, since each of the robots moved by at least S towards the center of $\mathcal{SEC}(P)$. Therefore the self-loop at state [IN0] can be repeated only finitely many times, ending in a configuration where either $|\mathcal{C}_{\text{int}}(P)| \geq 1$, two robots meet and create a multiplicity point, or all robots meet. Note also that if a multiplicity point is created after this step, then it is necessarily unique, as it could be created only by two or more robots from $\mathcal{C}_{\text{cir}}(P)$ meeting at the center point of $\mathcal{SEC}(P)$, which is the only possible intersection point for the trajectories of the robots.

State [IN1]: If $N \geq 4$ and $|\mathcal{C}_{\text{int}}(P)| = 1$, then a similar argument holds; thus also here the self-loop can be taken only finitely many times or the system's state changes to a state where $|\mathcal{C}_{\text{int}}(P)| \geq 2$, namely, [IN2] or [IN3], or a multiplicity point is created, leading to state [MULT]. If a multiplicity point is created as a result of this step, then it is unique, as it could be created only by one or more robots from $\mathcal{C}_{\text{cir}}(P)$ and the inner robot R_i, since the trajectories of any two robots moving towards R_i intersect only at the location of R_i.

State [IN2]: If $N \geq 4$ and $|\mathcal{C}_{\text{int}}(P)| = 2$, then the only outcome of this state could be a single multiplicity point, as can be verified by inspecting the possible substates. In this state, only two robots are active and move towards one goal point; thus the multiplicity point is unique.

State [IN2(a)]: If p_i and p_j do not share a cell, then both robots are instructed to go to the center of $\mathcal{SEC}(P)$. Eventually, either both robots will meet there, leading to state [MULT], or only one will arrive at the center; thus the two robots now share a cell, leading to state [IN2(b)].

State [IN2(b)]: If p_i and p_j share a single cell, $\text{Cell}(R_k)$, then either they meet on their way to R_k or one or both meet R_k at location p_k, thus leading to state [MULT].

State [IN2(c)]: If p_i and p_j share two cells, then the following possibilities may occur. Either the robot closer to $\mathcal{C}_{\text{cir}}(P)$, say R_i, meets with its target, say R_k, or R_j will meet R_i on its way, thus creating a multiplicity point, leading to state [MULT], or R_j enters the interior of the sector $\text{Cell}(R_k)$, thus leading to state [IN2(b)].

State [IN3]: Finally, if $N \geq 4$ and $|\mathcal{C}_{\text{int}}(P)| \geq 3$, then the procedure is applied recursively on the inner robots, while the outer robots remain stationary. Thus, as seen above, a single multiplicity point is created on the lowest level of the recursion. □

LEMMA 3.7. *In an $(N, 1)$-crash system, if the initial configuration is in state* [MULT]*, i.e., it contains a single multiplicity point p^*, then Procedure* GoTo_Mult *guarantees that within finite time all nonfaulty robots gather at p^* while avoiding the creation of additional multiplicity points.*

Proof. Every robot with a free corridor towards p^* is instructed to go towards p^*; thus all nonfaulty robots will arrive at p^* within a finite time. If a robot R_i detects another robot on its trajectory towards p^*, it looks for a free corridor by moving orthogonally to the multiplicity point, while making sure that it does not obstruct the free corridor of any other robot. This is ensured by moving only so as to change its angle with respect to the x-axis and p^* by a third of the angle to the closest-angle neighboring robot R_j (see Figure 8(a)). Note that it is possible that R_j will also enter the same sector, due to the lack of consistent coordinate system (and in particular,

the absence of common orientation, which may cause the R_jth clockwise sector to be the same as the R_ith clockwise sector). However, even if R_j enters that clear sector, it will be in the "far" third of the sector.

It is also possible for $k \geq 3$ robots $R_{i_1}, \ldots, R_{i_k}$ to share a common corridor to p^* (see Figure 8(b)). In this case the one closest to p^*, say R_{i_1}, will move towards p^*, and the others might take the same new trajectory to p^*. However, on this new trajectory, only $k-1$ robots collide, so the closest to p^* has a free corridor, and only $k-2$ robots must shift orthogonally again. Hence if a robot has more than one robot on its trajectory towards p^*, then it will remain in state [Blocked] during finitely many cycles until it has a free corridor towards p^*; thus it will eventually switch to state [Free] and arrive at p^*. □

THEOREM 3.8. *Algorithm* $\mathsf{Gather}_{\mathrm{crash}}$ *solves the gathering problem in an* $(N, 1)$*-crash system under the* $\mathcal{SSYNC}$ *model for any* $N \geq 3$.

Proof. Since the initial configuration is legal, by Lemma 3.6 it is guaranteed that Procedure Create_Mult will lead to a single multiplicity point. By Lemma 3.7, applying Procedure GoTo_Mult on a system with one multiplicity point leads to the gathering of all nonfaulty robots at that point. □

4. Impossibility of gathering under Byzantine faults.

4.1. Impossibility results in the $\mathcal{SSYNC}$ and $\mathcal{ASYNC}$ models. In [21] it is shown that the class of problems solvable in $\mathcal{ASYNC}$ is contained in the class of problems solvable in the $\mathcal{SSYNC}$ model. It follows that proving impossibility of gathering in an $(N, 1)$-Byzantine system in $\mathcal{SSYNC}$ also proves impossibility in $\mathcal{ASYNC}$. We next prove that in the $\mathcal{SSYNC}$ model it is impossible for any algorithm to achieve either gathering or convergence of three robots in the Byzantine fault model, even in the presence of at most one faulty robot.

Definition. A gathering algorithm $\mathcal{A}$ is called *hyperactive* if it instructs every robot to make a move in every cycle until the task is achieved; i.e., $\check{M}(N, \mathcal{A}) = N$.

THEOREM 4.1. *In a* $(3, 1)$*-Byzantine system under the* $\mathcal{SSYNC}$ *model, any non-hyperactive gathering algorithm will fail in achieving gathering or convergence.*

Proof. Suppose the system consists of three robots R_1, R_2, R_3, and there exists a scenario σ in which at some configuration C_1, R_1 is active, but the algorithm instructs it to stay in place. In this system, the adversary can do the following. It designates R_3 as faulty and executes the scenario σ with R_3 acting correctly up to a configuration C_1. At this cycle, it makes R_1 active and R_2 passive. As a result, neither R_1 nor R_2 moves in this cycle. In addition, the adversary moves R_3 to create a configuration C_2 that from R_2's point of view is equivalent to what R_1 has seen in C_1 (see Figure 10). The adversary now makes R_1 passive and R_2 active. Since R_2's state is equivalent to R_1's state in the previous configuration, the algorithm will now instruct R_2 to stay in place. The adversary can now switch from configuration C_1 to C_2 and back, forcing R_1 and R_2 to stay in place indefinitely. Therefore the algorithm fails to achieve gathering or convergence of the nonfaulty robots. □

Definition. A distributed robot algorithm is N*-diverging* if there exists an (N, f)-Byzantine system and a configuration in which the instructions of the algorithm combined with the actions of the adversary can cause two nonfaulty robots to increase the distance between them. An example of divergence caused by the instructions of the algorithm is illustrated in Figure 11(a). An example of divergence caused by the intervention of the adversary is illustrated in Figure 11(b), where robot R_1 is stopped short of reaching its goal point.

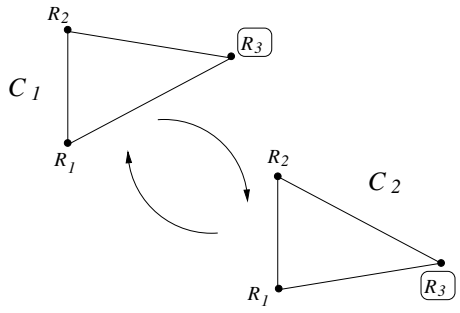


FIG. 10. *Theorem* 4.1.

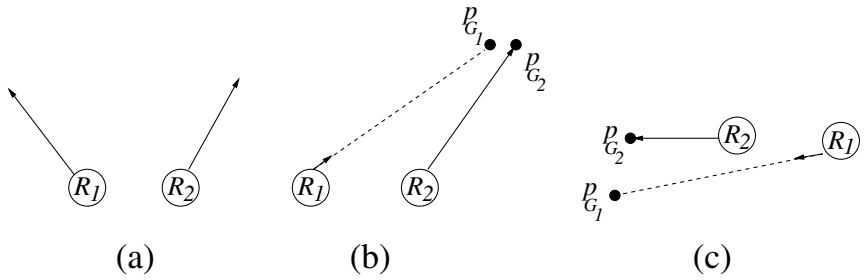


FIG. 11. *Divergence of robots.*

The premature-stopping technique. Our impossibility proofs make extensive use of the following technique. In order to cause two robots to diverge in some given configuration C of a given system $\mathcal{T}$, the adversary can stop a nonfaulty robot R_i after traversing a relatively small distance S_i. Note that S_i might be smaller than S in the current system, in which case the adversary is not permitted to stop R_i prematurely. However, as the algorithm is required to be valid in any system, it is intuitively clear that we may always consider a different system $\mathcal{T}'$ with S small enough to allow a movement of distance S_i. Moreover, since the algorithm is unaware of the value of S, it cannot distinguish between identical configurations in the two systems $\mathcal{T}$ and $\mathcal{T}'$ and will issue the same instructions to each robot in configuration C in $\mathcal{T}$ and $\mathcal{T}'$. Therefore the premature-stopping technique can be applied with any movement length greater than zero. We make this argument more formal in the proofs that follow.

As for the applicability of the premature-stopping technique, the adversary can apply it to cause the robots to diverge in any case where two robots move towards their respective goals on nonintersecting trajectories (see Figure 11(c)). In addition, even if the trajectories do intersect, the adversary can still apply the technique in some cases and again cause divergence, as seen in Figure 11(b). (One example for a case in which the premature stopping technique cannot help the adversary to force divergence is when the trajectories of the two robots R_1 and R_2 intersect and the angle between $p(R_1)$, $p(R_1)$ and the intersection point is at least $\pi/2$; see Lemma 3.3 and Figure 2.)

LEMMA 4.2. *In the $\mathcal{SSYNC}$ (or even the $\mathcal{FSYNC}$) model, a 3-diverging algorithm will fail to achieve gathering or convergence.*

Proof. Suppose, towards contradiction, that there exists a 3-diverging algorithm $\mathcal{A}$ that solves the gathering problem. Consider a $(3, 1)$-Byzantine system $\mathcal{T}$ with robots

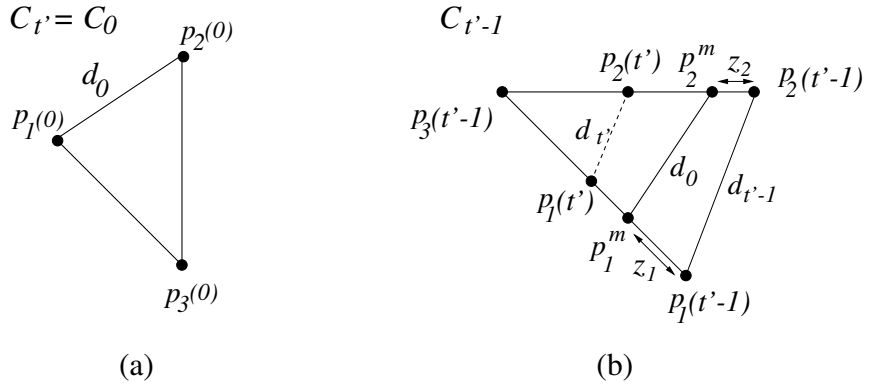


FIG. 12. *Lemma* 4.2.

R_1, R_2, and R_3, and a configuration C_0 on which $\mathcal{A}$'s instructions and the adversary's actions cause R_1 and R_2 to increase their distance. For $t \geq 0$ and $i = 1, 2$, denote by $p_i(t)$ the location of robot R_i in configuration C_t, and let $d_t = \text{dist}(p_1(t), p_2(t))$. Let $\sigma = \{C_0, C_1, \ldots, C_k\}$ be the sequence of configurations in an execution of the algorithm in which the adversary intervenes on the transition from C_0 to C_1 so as to increase $\text{dist}(R_1, R_2)$ but does not intervene thereafter, and in C_k all robots are gathered in one point. Note that $d_1 > d_0$. Let $t' = \min\{t \mid d_t \leq d_0, 2 \leq t \leq k\}$. By continuity considerations, as $d_{t'-1} > d_0$ and $d_{t'} \leq d_0$, there must be a time during the transition from $C_{t'-1}$ to $C_{t'}$ in which the robots R_1 and R_2 were located in middle points p_1^m and p_2^m, respectively, at distance exactly $\text{dist}(p_1^m, p_2^m) = d_0$. For $i = 1, 2$, let $z_i = \text{dist}(p_i(t'-1), p_i^m)$ (see Figure 12) and denote the minimum distance any robot traversed at any cycle $0 \leq t \leq t'-1$ by z_3.

Now replace the $(3,1)$-Byzantine system $\mathcal{T}$ by another system $\mathcal{T}'$ where $S = \min\{z_1, z_2, z_3\}$ and consider the following scenario. The adversary designates R_3 as faulty and executes the scenario σ up to $C_{t'-1}$. At this cycle, it stops R_1 and R_2 at points p_1^m and p_2^m, respectively, and moves the faulty robot R_3 to the exact same position it occupied in C_0. Thus, the new configuration $\tilde{C}_t$ is identical to C_0; hence the robots R_1 and R_2 will diverge again, and the system can be made to cycle through the configuration sequence $\{C_0, \ldots, \tilde{C}_t\}$ indefinitely, and thus R_1 and R_2 will never meet, contradicting the assumption. □

OBSERVATION 4.3. *Let $\mathcal{A}$ be an algorithm operating in a $(3,1)$-Byzantine system. Let $\vec{L}_{i,j}$ be the straight half-line starting at p_i and going through p_j. Suppose that in some configuration C, $\mathcal{A}$ instructs R_i and R_j to move on vectors $\vec{\mathrm{v}}_i$ and $\vec{\mathrm{v}}_j$ towards destination points g_i and g_j, respectively. Denote the angle between $\vec{L}_{i,j}$ and $\vec{\mathrm{v}}_i$ (measured from $\vec{L}_{i,j}$ in the counterclockwise direction) by μ_i, and the angle between $\vec{L}_{i,j}$ and $\vec{\mathrm{v}}_j$ by μ_j (see Figure 13). Then each of the following is a sufficient condition for $\mathcal{A}$ to be 3-diverging:*

(C1) $0 \leq \mu_j \leq \mu_i \leq \pi$.
(C2) $\pi \leq \mu_i \leq \mu_j \leq 2\pi$.
(C3) $0 \leq \mu_i \leq \pi \leq \mu_j \leq 2\pi$ *or* $0 \leq \mu_j \leq \pi \leq \mu_i \leq 2\pi$.
(C4) $0 \leq \mu_i < \mu_j \leq \pi$ *and either* $\mu_i \geq \pi/2$ *or* $\mu_j \leq \pi/2$.
(C5) $\pi \leq \mu_j < \mu_i \leq 2\pi$ *and either* $\mu_i \leq 3\pi/2$ *or* $\mu_j \geq 3\pi/2$.

Proof. To show that $\mathcal{A}$ is 3-diverging in each of these cases, we have to show a scenario in which the instructions of $\mathcal{A}$ combined with the actions of the adversary will

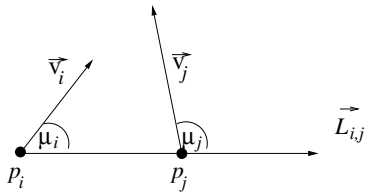


FIG. 13. *Observation* 4.3.

cause R_i and R_j to increase the distance between them. Denote the current distance between R_i and R_j by d_1, the location of R_i after traversing a distance S_i by p'_i, and the location of R_j after traversing a distance S_j by p'_j. Let $d_2 = \text{dist}(p'_i, p'_j)$.

Case (C1). It is easy to see that if $\mu_i = \mu_j = \pi/2$ and the robots move in different distances towards their goals, then $d_2 > d_1$ (as the hypotenuse in a right triangle is the longest side in the triangle). Therefore, in a system where $S = \min\{S_i, S_j\}, S_i \neq S_j$, it is enough that the adversary applies the premature-stopping technique, stops R_i after exactly a distance S_i, and stops R_j after traversing S_j. If $\mu_i = \mu_j < \pi/2$, then, applying again the premature-stopping technique, the adversary stops R_j after exactly a distance S and lets R_i continue traversing any distance greater than S. Similarly, if $\mu_i = \mu_j > \pi/2$, then the adversary stops R_i after traversing exactly a distance S and lets R_j traverse any distance greater than S. Finally, if $0 < \mu_j < \mu_i < \pi$, then the adversary can apply the premature-stopping technique and stop R_j after traversing a small distance and let R_j continue its movement as planned. In all cases, $d_1 > d_2$.

Case (C2). This case is simply a reflection of Case (C1).

Case (C3). It is easy to see that the trajectories of R_i and R_j diverge and never intersect. Traversing on those trajectories might not always cause divergence (for example, if the trajectory of R_j runs close to the current location of R_i as in Figure 11(c)), but by applying the premature-stopping technique as explained earlier, the adversary can cause divergence.

Case (C4). If $0 \leq \mu_i < \mu_j \leq \pi$ and either $\mu_i \geq \pi/2$ or $\mu_j \leq \pi/2$, then $\vec{v}_i$ and $\vec{v}_j$ intersect at some point, p^I. Without loss of generality let $\mu_i \geq \pi/2$. Drop a perpendicular line from p_j to the line going through p^I and p_i and let q_0 be the intersection point (see Figure 14). Let $d_3 = \text{dist}(p_j, p'_i)$. Consider the triangle $\triangle p'_i p_j q_0$. Clearly $d_3 > d_1$; therefore as $S_j \to 0$, $d_2 \to d_3$, and hence $d_2 > d_1$. Now apply the premature-stopping technique by stopping R_j after traversing a distance of exactly S, where S is very small (tending to 0), and allow R_i to traverse a distance S_i, thus causing $d_2 > d_1$.

Case (C5). This case is a reflection of Case (C4). □

THEOREM 4.4. *In a* $(3,1)$*-Byzantine system under the* $\mathcal{SSYNC}$ *model it is impossible to perform successful gathering or convergence.*

Proof. Consider a gathering algorithm $\mathcal{A}$ and an initial setting in which the three robots R_1, R_2, and R_3 are collinear, with R_2 in the middle. If the algorithm instructs R_2 to remain stationary, then it is nonhyperactive and by Theorem 4.1 will not achieve gathering. From Observation 4.3 it follows that if $0 \leq \mu_1, \mu_2, \mu_3 \leq \pi$, then in order to avoid being 3-diverging, necessarily $\mu_3 > \mu_2 > \mu_1$ (see Figure 15). But under this assumption, if $\mu_2 \geq \pi/2$, then applying Case (C4) of Observation 4.3 with respect to p_2 and p_3 yields that $\mathcal{A}$ is 3-diverging. If, on the other hand, $\mu_2 \leq \pi/2$, then applying Case (C4) of Observation 4.3 with respect to p_1 and p_2 yields the same conclusion.

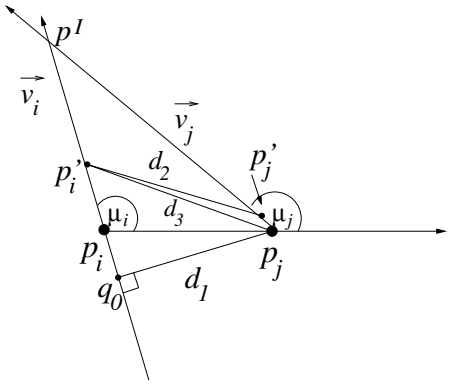


FIG. 14. *Observation* 4.3*, Case* (C4).

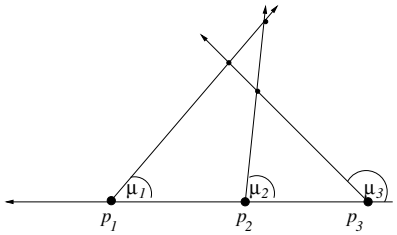


FIG. 15. *Illustration of proof of Theorem* 4.4.

A similar argument applies in case $\pi \leq \mu_1, \mu_2, \mu_3 \leq 2\pi$. Finally, if $\mu_1 > \pi$ and $\mu_2, \mu_3 < \pi$ or $\mu_1, \mu_2 > \pi$ and $\mu_3 < \pi$, then algorithm $\mathcal{A}$ is 3-diverging by Case (C3) of Observation 4.3. Thus, by Lemma 4.2, algorithm $\mathcal{A}$ fails to achieve gathering or convergence. □

We remark that in the $\mathcal{FSYNC}$ model, an N-diverging algorithm for $N > 3$ will not necessarily fail. In particular, the algorithms suggested in subsection 5.3 for the $\mathcal{FSYNC}$ model might be diverging yet still achieve gathering. Also, in the $\mathcal{FSYNC}$ model, a nonhyperactive algorithm will not necessarily fail. In particular, the gathering algorithm for $N = 3$ suggested in section 5 for the $\mathcal{FSYNC}$ model is not always hyperactive (for example, when the three robots are collinear, the robot lying in the middle is instructed to remain still). In fact, the converse may hold; namely, in the $\mathcal{FSYNC}$ model, a *hyperactive* algorithm might be problematic. For example, it is shown in the following lemma that in a one-dimensional setting, the adversary can cause failure of every hyperactive algorithm.

LEMMA 4.5. *In the $\mathcal{FSYNC}$ model, a hyperactive algorithm for a one-dimensional $(3,1)$-Byzantine system will fail to achieve gathering or convergence.*

Proof. Consider a $(3,1)$-Byzantine system on the line. Consider an arbitrary configuration C in which all robots are instructed by the algorithm to move. Without loss of generality suppose that at least two of the robots, say R_1 and R_2, are instructed to move to the right. Let ϵ_i denote the distance traversed by the robot R_i in the current round, and without loss of generality suppose $\epsilon_1 \leq \epsilon_2$. Then the same behavior will occur in a robot system in which $S \leq \epsilon_1$. In such a system, the adversary can stop R_2 after traversing a distance of only ϵ_1. The adversary can also fail the third robot R_3,

making it move to the right to distance ϵ_1. The resulting configuration is identical to the original C, implying that the adversary can keep the system at this configuration indefinitely. □

4.2. Intuition: Problems with previous approaches. The difficulty of handling a system of autonomous mobile robots with Byzantine faults is due to, among other reasons, the conventions regarding multiplicities on which most existing algorithms rely. In particular, these algorithms are based on enforcing the following conventions: (a) No more than one multiplicity point is created throughout the execution of the algorithm until successful gathering is achieved. (b) All robots lying in a multiplicity point remain stationary. (c) Robots lying in a multiplicity point are never separated again. These conventions are used for both gathering algorithms and other pattern formation algorithms; see [9].

All of the above assumptions no longer hold in a system where Byzantine faults might occur. First, the adversary could create a second multiplicity point as soon as it detects one such point, by "failing" a robot that does not lie in the multiplicity point and sending it to the location of yet another currently single robot. As a result, in the gathering problem assumption (b) cannot be relied on. Assumption (c) is violated even if the algorithm instructs all robots lying on the same point to move towards the *same* destination point, as the adversary could stop their movement in different locations.

Since all known algorithms rely on conventions (a)–(c) listed above, which can be violated in a system consisting of N robots with even one Byzantine faulty robot, it is clear that those algorithms fail to achieve gathering.

To get a feel for the possible complications that may occur in this model, let us consider some simple solutions one might propose for the problem. One natural general approach for attacking the problem is to try to gradually reduce the number of distinct points where the robots reside, by gathering partial subsets of robots at different points. A possible algorithm attempting to achieve that is one that requires each robot, in each cycle, to move towards its *closest* neighbor. This may lead to deadlocks once the robots pair up, since each one's closest neighbor already resides at the same location. Therefore the algorithm should instruct each robot to move towards the closest robot among those currently residing at locations *other* than its own. One problem that arises is that sets of robots that have already met might break up again; hence "progress" is hard to measure. Another obvious problem is that of symmetry breaking. Even ignoring this problem, this approach can still lead to nonconverging scenarios. For instance, suppose that the N robots are located on a straight line, with R_i at location $x_i = i(i-1)/2$. Then the algorithm requires R_1 to move towards R_2 and R_i to move towards R_{i-1} for every $2 \le i \le N$. However, if R_1 is faulty and chooses to move *away* from R_2, and all robots traverse exactly a distance S, then the configuration is translated by S in the $-x$ direction and is otherwise unchanged. (See Figure 16.)

Another natural algorithm is based on computing the center of gravity p_G of the configuration and going to p_G. This algorithm can be failed by the adversary in

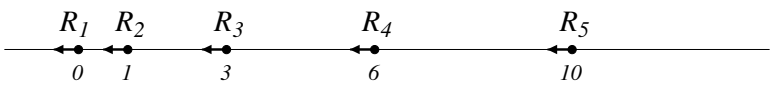


FIG. 16. *Bad scenario for nearest-neighbor algorithm.*

similar manner, by failing a robot located far from the rest and taking it to a walk towards infinity, forcing the entire pack of nonfaulty robots to be dragged along.

5. Fault-tolerant gathering in the $\mathcal{FSYNC}$ model.

5.1. Preliminaries. We now discuss the problem of gathering N autonomous mobile robots in an (N, f)-Byzantine system under the fully synchronous model. We use the following notation. Denote the *geometric span* (or *diameter*) of the set of points P by

$$\mathrm{Span}(P) \;=\; \max\{\mathrm{dist}(p,q) \mid p,q \in P\}.$$

Denote the convex hull of a multiset of points P by $H(P)$, and the set of vertices of $H(P)$ by $V_{\mathrm{H}}(P)$. Denote the set of (N to $N-f$) nonfaulty robots by $\mathcal{R}_{NF}$. Denote the *center of gravity* (or barycenter) of a multiset P of $n \geq 3$ points $p_i = (x_i, y_i), i = 1, \ldots, N$ by

$$C_{\mathrm{grav}}(P) = \left(\frac{\sum_{i=i}^{N} x_i}{N} \;,\; \frac{\sum_{i=i}^{N} y_i}{N} \right).$$

Define the sum of distances between all pairs of nonfaulty robots as

$$D_{\mathrm{tot}}(P) = \sum_{R_i, R_j \in \mathcal{R}_{NF}} \mathrm{dist}(R_i, R_j).$$

Note that for any set of points P, while the center of gravity $C_{\mathrm{grav}}(P)$ is defined in terms of the point coordinates in some specific coordinate system, the resulting point is independent of the particular coordinate system in use. Hence for a set of robots in some arbitrary configuration C in the plane, whenever each of the robots computes $C_{\mathrm{grav}}(P)$, the resulting point computed by the different robots is the same, even if each robot has its own coordinate system.

Definition. A robot algorithm is *concentrating* if it satisfies the following properties:

1. It is nondiverging; i.e., no two nonfaulty robots will increase the distance between them in any round.
2. There exists a constant $c > 0$ such that at each step, at least one pair of nonfaulty robots that are at different locations either meets or decreases the distance between them by at least c.

LEMMA 5.1. *Let $\mathcal{A}$ be a concentrating algorithm. Then in a $(3,1)$-Byzantine system under the $\mathcal{FSYNC}$ model, $\mathcal{A}$ achieves gathering.*

Proof. If in each cycle D_{tot} decreases by a constant amount c, then within a finite number of cycles $\mathcal{A}$ achieves gathering of all nonfaulty robots (since D_{tot} must reach 0). If there is indeed one faulty robot, then there may be only one pair of nonfaulty robots. The algorithm $\mathcal{A}$ ensures that the distance between the two nonfaulty robots decreases by at least a constant c in each cycle; hence $D_{\mathrm{tot}}^{new} \leq D_{\mathrm{tot}}^{old} - c$ and therefore these two robots will eventually meet. If all three robots are nonfaulty, then $\mathcal{A}$ ensures that the distance between at least one pair, say R_1 and R_2, decreases by at least c while $\mathrm{dist}(R_1, R_3)$ and $\mathrm{dist}(R_2, R_3)$ do not increase (since $\mathcal{A}$ is nondiverging); hence $D_{\mathrm{tot}}^{new} \leq D_{\mathrm{tot}}^{old} - c$ and $\mathcal{A}$ achieves gathering. ∎

Definition. A distributed robot algorithm $\mathcal{A}$ is said to *dictate* 2*-pair convergence* in a given cycle if in that cycle it instructs two distinct pairs of robots to decrease the distance between them by a constant amount. A distributed robot algorithm $\mathcal{A}$

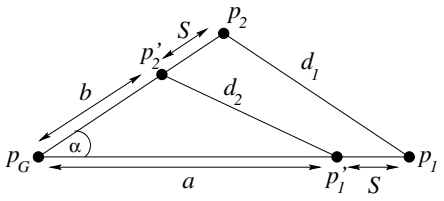


FIG. 17. *Proof of Lemma* 5.2.

is said to *dictate triple convergence* in a given cycle if in that cycle it instructs three robots to decrease by a constant amount the distance between every pair of them.

Note that for $N = 3$, triple convergence implies also nondivergence and hence also concentration. Note also that these conditions do not require that the robots involved be nonfaulty; in particular, one of them may fail and disobey the algorithm's instructions, in which case its distances will not decrease as needed. Nevertheless, these conditions turn out to be sufficient for gathering in certain settings.

LEMMA 5.2. *Consider two robots R_1 and R_2, initially located at the points p_1 and p_2, which traverse the same distance S' towards a common meeting point p_G, and let $\alpha = \angle p_1 p_G p_2$. If $\alpha \le \pi/2$, then the distance between them decreases by at least $S'(1 - \cos\alpha)$.*

Proof. Let p_1' and p_2' denote the new location of R_1 and R_2 after moving a distance S' towards p_G, and let $d_1 = \mathrm{dist}(p_1, p_2)$, $d_2 = \mathrm{dist}(p_1', p_2')$, $a = \mathrm{dist}(p_1', p_G)$, and $b = \mathrm{dist}(p_1', p_G)$ (see Figure 17).

By the cosine theorem on the triangles $\triangle p_1 p_G p_2$ and $\triangle p_1' p_G p_2'$, it follows that

$$d_1^2 = (a + S')^2 + (b + S')^2 - 2(a + S')(b + S')\cos\alpha,$$
$$d_2^2 = a^2 + b^2 - 2ab\cos\alpha.$$

Therefore

$$d_1 - d_2 = \frac{2a + 2b + S'}{d_1 + d_2} \cdot S'(1 - \cos\alpha).$$

By the triangle inequality on triangles $\triangle p_1 p_G p_2$ and $\triangle p_1' p_G p_2'$ it follows that $a + b + 2S' > d_1$ and $a + b > d_2$. Therefore $\frac{2a+2b+S'}{d_1+d_2} > 1$, so $d_1 - d_2 > S'(1 - \cos\alpha)$. In the range $(0, \pi/2]$ this value is always greater than 0 and is equal to 0 if and only if $\alpha = 0$. □

5.2. Gathering on a (3, 1)-Byzantine system in the $\mathcal{FSYNC}$ model. Let us next describe a gathering algorithm for three robots in an $\mathcal{FSYNC}$ model with at most one Byzantine fault. The input to this procedure is a configuration $P = \{p_1, p_2, p_3\}$.

Procedure 3-Gather$_{\mathrm{Byz}}(P)$

1. **State [Collinear]:** p_1, p_2, p_3 are collinear with p_2 in the middle:
   Set $p_G \leftarrow p_2$.
2. **State [Triangle]:** The three points form a triangle:
   Set p_G to be the intersection point of the three angle bisectors of the triangle $\triangle p_1 p_2 p_3$.

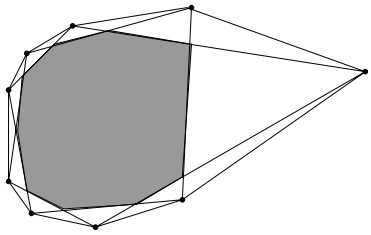


FIG. 18. *Illustration of* H^1_{int}.

Observe that the case of two robots residing at the same point, say $p_1 = p_2$, is handled by step 1 of the procedure. In this case, $p_G = p_1$; thus R_1 and R_2 stay in place and R_3 is required to move towards them.

Analysis.

THEOREM 5.3. *Algorithm* 3-Gather$_{\text{Byz}}$ *solves the gathering problem in a* $(3,1)$*-Byzantine system under the* $\mathcal{FSYNC}$ *model.*

Proof. Let us first consider the case when R_1, R_2, and R_3 are collinear, say, with R_2 in the middle. Since both extreme robots are instructed to move towards R_2, and R_2 is instructed to stay in place, it is clear that the instructions of the algorithm ensure that $\text{dist}(R_1, R_3)$ decreases in each cycle by at least S (or they meet), and also that $\text{dist}(R_1, R_2)$ and $\text{dist}(R_2, R_3)$ decrease by at least S (or they meet). Hence the algorithm dictates triple convergence in each cycle.

Next, suppose that R_1, R_2, and R_3 are not collinear. By Lemma 3.2, the angle between every two robots and p_G is greater than $\pi/2$. Therefore, by Lemma 3.3, we again have triple convergence.

Therefore in each cycle, whether the robots are collinear or not, triple convergence is ensured. Since for $N = 3$ triple convergence implies nondivergence as well, the algorithm achieves gathering by Lemma 5.1. ∎

5.3. Gathering for $f \geq 1$ and $N \geq 3f + 1$ in the $\mathcal{FSYNC}$ model. In this section we propose an algorithm for solving the gathering problem in an (N, f)-Byzantine system, where $N \geq 3f + 1$ in the $\mathcal{FSYNC}$ model. The main idea of the algorithm is to ensure that the goal point selected in each cycle falls in the convex hull of the *nonfaulty* robot locations. As shown later, this ensures that the geometric span of the set of locations of the nonfaulty robots decreases by at least $0.25S$; thus the robots will meet within a finite number of cycles. Due to its high complexity, this algorithm is only of theoretical merit, except for small values of f.

Definition. The *hull intersection* $H^k_{\text{int}}(P)$ is the convex set created as the intersection of all $\binom{N}{k}$ sets $H(P \setminus \{p_{i_1}, \ldots, p_{i_k}\})$ for $1 \leq k \leq N$, $p_{i_j} \in P$. (See Figure 18 for $k = 1$.)

The algorithm. Consider the following Procedure Gather$_{\text{Byz}}$ for determining the goal point p_G in each cycle. The input to this procedure is a configuration $P = \{p_1, \ldots, p_N\}$, and f is the maximum number of faulty robots.

Procedure Gather$_{\text{Byz}}(P)$

1. Compute $Q \leftarrow V_{\text{H}}(H^f_{\text{int}}(P))$.
2. Set $p_G \leftarrow C_{\text{grav}}(Q)$.

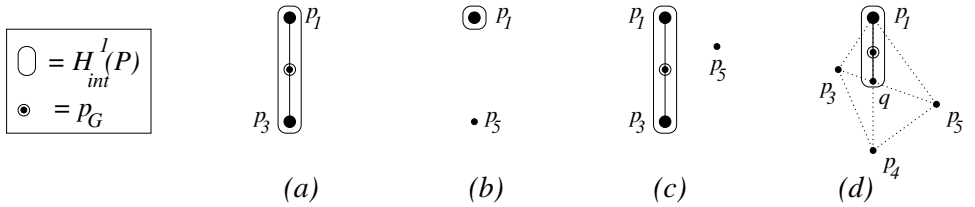


FIG. 19. *Illustration of Procedure* $\mathsf{Gather}_{\mathrm{Byz}}$ *in a* $(5,1)$*-Byzantine system.*

To illustrate the algorithm, let us consider a number of possible configurations of a $(5,1)$-Byzantine system (see Figure 19).

(a) $p_1 = p_2$, and $p_3 = p_4 = p_5$. Then $H^1_{\mathrm{int}}(P)$ is the segment $\overline{p_1p_3}$, and p_G is its midpoint.
(b) $p_1 = p_2 = p_3 = p_4 \neq p_5$. Then $H^1_{\mathrm{int}}(P) = \{p_1\}$, and $p_G = p_1$.
(c) $p_1 = p_2$, $p_3 = p_4$, and p_5 is distinct. Then $H^1_{\mathrm{int}}(P)$ is the segment $\overline{p_1p_3}$, and p_G is its midpoint.
(d) $p_1 = p_2$, and p_3, p_4, p_5 are distinct. Then $H^1_{\mathrm{int}}(P)$ is some segment $\overline{qp_1}$, and p_G is its midpoint.

Analysis. Let us first prove that the algorithm is well defined. For this we have to show that the set Q is nonempty.

Helly's theorem for $d = 2$ (cf. [28, Theorem 4.1.1]). Let $\mathcal{S}$ be a finite family of at least three convex sets in $\mathbb{R}^2$. If every three members of $\mathcal{S}$ have a point in common, then there is a point common to all members of $\mathcal{S}$.

LEMMA 5.4. *For a multiset* $P = \{p_1, \ldots, p_N\}, N \geq 3k+1$, $H^k_{\mathrm{int}}(P)$ *is convex and nonempty.*

Proof. $H^k_{\mathrm{int}}(P)$ is convex as it is the intersection of $\binom{N}{k}$ convex sets. We prove that it is nonempty by Helly's theorem. Consider three arbitrary sets $P^l = \{p^l_1, \ldots, p^l_k\} \subseteq P, 1 \leq l \leq 3$, and let $Q^l = H(P \setminus P^l)$, $1 \leq l \leq 3$. Then $Q^1 \cap Q^2 \cap Q^3$ contains at least $P' = P \setminus (P^1 \cup P^2 \cup P^3)$. As $|P| \geq 3k+1$, $|P'| \geq 1$. It follows that the intersection of every three such sets is nonempty, and by Helly's theorem $V_{\mathrm{H}}(H^k_{\mathrm{int}}(P))$ is nonempty as well. ∎

The analysis of Procedure $\mathsf{Gather}_{\mathrm{Byz}}$ is based on showing that if a set of K robots $R_1, \ldots, R_K$ initially located at the points $P = \{p_1, \ldots, p_K\}$ move towards a point p_G in their convex hull $H(P)$ and their new positions are at the points $P' = \{p'_1, \ldots, p'_K\}$, then their geometric span decreases by at least cS for some constant $c \geq 1/4$; i.e., $\mathrm{Span}(P') \leq \mathrm{Span}(P) - cS$. Consequently, the robots will meet within a finite number of cycles.

LEMMA 5.5. *Let* $P = \{p_1, \ldots, p_k\}$ *be a set of points in the plane.*

1. $\mathrm{Span}(P) \geq \mathrm{dist}(p, p')$ *for every two points* p, p' *in the convex hull* $H(P)$.
2. *The geometric span is attained by two points* $p_a, p_b \in P$ *that occur as vertices in* $H(P)$.
3. *Moreover, for every point* p_G *in* $H(P)$, $\angle p_a p_G p_b \geq \pi/4$.

Proof. Consider two arbitrary points p, p' inside $H(P)$. By the definition of the convex hull, it is clear that the segment $\overline{pp'}$ falls entirely within the convex hull of P. Therefore this segment can be extended in both directions towards the circumference of $H(P)$, hitting it at the points q, q'. Hence $\mathrm{dist}(p, p') \leq \mathrm{dist}(q, q')$. If the points q and q' are vertices of the convex hull, then $q, q' \in P$, and we are done. So now suppose this is not the case. If q is not a vertex of $H(P)$, then it occurs on an edge $\overline{p_i p_{i+1}}$. In this case, at least one of the two adjacent vertices of the convex hull,

without loss of generality p_i, satisfies that $\mathrm{dist}(q', p_i) > \mathrm{dist}(q', q)$. Similarly, if q' is not a vertex, then it occurs on an edge $\overline{p_j p_{j+1}}$, and again without loss of generality $\mathrm{dist}(p_j, p_i) > \mathrm{dist}(q', p_i)$. Hence combined, $\mathrm{Span}(P) \geq \mathrm{dist}(p, p')$. Therefore for each such segment $\overline{pp'}$ there exists a segment $\overline{p_i p_j}$, $p_i, p_j \in P$, whose length is greater than or equal to $\mathrm{dist}(q, q')$.

The proof used to establish the first claim of the lemma also yields the second claim, as it shows that for any two points q, q' that are not both vertices of $H(P)$, there exist two vertices $p_i, p_j \in P$ of $H(P)$ satisfying $\mathrm{dist}(p_i, p_j) > \mathrm{dist}(q, q')$; hence $\mathrm{Span}(P)$ cannot be attained by those points q, q'.

The third claim of the lemma is proved as follows. Let p_a, p_b be the two vertices of $H(P)$ attaining $\mathrm{Span}(P)$, and suppose, towards contradiction, that there exists a point p_G in $H(P)$ such that $\alpha = \angle p_a p_G p_b < \pi/4$. Consider the triangle $\triangle p_a p_G p_b$. Let $\beta = \angle p_a p_b p_G$ and $\gamma = \angle p_b p_a p_G$. Without loss of generality assume that $\beta \geq \gamma$. Then

$$\alpha < \pi/4 < 3\pi/8 < (\pi - \alpha)/2 = (\beta + \gamma)/2 < \beta < \beta + \gamma = \pi - \alpha.$$

Hence $\sin\beta > \sin\alpha$. Also, by the sine theorem on the triangle $\triangle p_a p_G p_b$,

$$\frac{\mathrm{dist}(p_a, p_b)}{\mathrm{dist}(p_a, p_G)} = \frac{\sin\alpha}{\sin\beta}.$$

It follows that $\mathrm{dist}(p_a, p_G) > \mathrm{dist}(p_a, p_b)$. By part 1 of the lemma, $\mathrm{Span}(P) \geq \mathrm{dist}(p_a, p_G) > \mathrm{dist}(p_a, p_b)$, contradicting the assumption. □

LEMMA 5.6. *For every two sets of points P and Q, if $H(P) \subseteq H(Q)$, then* $\mathrm{Span}(P) \leq \mathrm{Span}(Q)$.

Proof. Let $p_a, p_b \in P$ be the vertices attaining the geometric span of P. As $P \subseteq H(P) \subseteq H(Q)$, also $p_a, p_b \in H(Q)$. Thus by Lemma 5.5, $\mathrm{Span}(P) = \mathrm{dist}(p_a, p_b) \leq \mathrm{Span}(Q)$. □

LEMMA 5.7. *If a set of K robots $R_1, \ldots, R_K$ initially located at the points $P = \{p_1, \ldots, p_K\}$ traverse the same distance S towards a point p_G in the convex hull $H(P)$ and their new positions are at the points $P' = \{p'_1, \ldots, p'_K\}$, then their geometric span decreases by at least cS for some constant $c \geq 1/4$; i.e.,* $\mathrm{Span}(P') \leq \mathrm{Span}(P) - cS$.

Proof. Let p_a, p_b be the two vertices of $H(P)$ attaining $\mathrm{Span}(P)$, and let p'_i, p'_j be the two vertices of $H(P')$ attaining $\mathrm{Span}(P')$. Note that p_G is internal also to $H(P')$. Hence, by part 3 of Lemma 5.5, it follows that $\alpha = \angle p_a p_G p_b \geq \pi/4$. If $\pi/4 \leq \alpha < \pi/2$, then according to Lemma 5.2, $\mathrm{dist}(p'_a, p'_b) \leq \mathrm{dist}(p_a, p_b) - (1 - \cos\alpha) \leq \mathrm{dist}(p_a, p_b) - 0.25S$. Also, if $\alpha \geq \pi/2$, then by Lemma 3.3, $\mathrm{dist}(p'_a, p'_b) \leq \mathrm{dist}(p_a, p_b) - 0.7S$. Therefore, in any case $\mathrm{dist}(p'_a, p'_b) \leq \mathrm{dist}(p_a, p_b) - 0.25S$. Also, since p_a, p_b attains $\mathrm{Span}(P)$, it follows that $\mathrm{dist}(p_a, p_b) \geq \mathrm{dist}(p_i, p_j)$; therefore

$$\begin{aligned}\mathrm{Span}(P') &= \mathrm{dist}(p'_i, p'_j) \leq \mathrm{dist}(p_i, p_j) - S/4 \leq \mathrm{dist}(p_a, p_b) - S/4 \\ &= \mathrm{Span}(P) - S/4. \qquad \square\end{aligned}$$

COROLLARY 5.8. *If a set of K robots $R_1, \ldots, R_K$ initially located at the points $P = \{p_1, \ldots, p_K\}$ move towards a point p_G in the convex hull $H(P)$ and their new positions are at the points $P' = \{p'_1, \ldots, p'_K\}$, then their geometric span decreases by at least cS for some constant $c \geq 1/4$; i.e.,* $\mathrm{Span}(P') \leq \mathrm{Span}(P) - cS$.

Proof. By the model assumption, each robot traverses a distance of at least S towards p_G. Let p''_i denote the point at a distance *exactly* S from p_i in the direction of p_G, and let $P'' = \{p''_1, \ldots, p''_K\}$. Clearly $H(P') \subseteq H(P'')$. By Lemma 5.6, $\mathrm{Span}(P') \leq \mathrm{Span}(P'')$. By Lemma 5.7, $\mathrm{Span}(P'') \leq \mathrm{Span}(P) - cS$. The claim follows. □

LEMMA 5.9. *If a set of K robots $R_1, \ldots, R_K$ move in every cycle t towards a point p_G in their convex hull, then the robots will meet within a finite number of cycles.*

Proof. For $t \geq 1$, denote by H_t the convex hull of the robot configuration at the beginning of cycle t. In each cycle, the robots move a distance of at least S towards a point p_G in the convex hull. Thus, by Corollary 5.8, $\mathrm{Span}(H_{t+1}) \leq \mathrm{Span}(H_t) - 0.25S$ for every t. Therefore within at most $4 \cdot \mathrm{Span}(H_1)/S$ cycles, the geometric span of the robot configuration will be 0; thus all robots meet. ∎

THEOREM 5.10. *Algorithm* $\mathsf{Gather}_{\mathrm{Byz}}$ *solves the gathering problem in an (N, f)-Byzantine system under the $\mathcal{FSYNC}$ model for any $N \geq 3f + 1$.*

Proof. By Lemma 5.9 it is sufficient to show that the goal point p_G selected in the cycle falls in $H(\mathcal{R}_{NF})$, the convex hull of the nonfaulty robots. To prove this, we check the goal point p_G determined in each cycle.

By definition, the set $H^f_{\mathrm{int}}(P)$ is contained in its entirety in $H(P)$ as well as in the convex hull of every $N - f$ points of P; thus, in particular, it falls in $H(\mathcal{R}_{NF})$. Since the center of gravity of a set of points is inside its convex hull, it follows that p_G is in $H(\mathcal{R}_{NF})$. By Lemma 5.4, it follows that $H^f_{\mathrm{int}}(P)$ is nonempty; thus the center of gravity of the set $V_{\mathrm{H}}(H^f_{\mathrm{int}}(P))$ is well defined. ∎

6. Open problems. The design of fault-tolerant distributed control algorithms for multiple robot systems is still far from being fully explored. Directions for future research include the following. To begin with, it may be useful to study other kinds of fault models in addition to the crash and Byzantine models, such as a model in which the robots might lose some of their movement control (for instance, lose control of their movement length), or a model in which robots might diverge from their original movement direction up to a certain percentage of error. It is also necessary to develop fault-tolerant algorithms for tasks other than gathering (e.g., formation of geometric patterns). The maximum number of faults under which a solution is still feasible, for gathering and other tasks, has yet to be established. Finally, it would be interesting to examine the effect of changes in some initial assumptions on the model's fault tolerance properties. Examples for possible model changes include partial nonobliviousness of the robots (e.g., robots equipped with a small amount of memory, say, allowing them to remember the subsequence of X most recent cycles), robots capable of partial agreement on their orientation, or robots capable of explicit communication (perhaps under certain limitations, e.g., only with nearby robots).

REFERENCES

[1] H. ANDO, Y. OASA, I. SUZUKI, AND M. YAMASHITA, *A distributed memoryless point convergence algorithm for mobile robots with limited visibility*, IEEE Trans. Robotics and Automation, 15 (1999), pp. 818–828.

[2] H. ANDO, I. SUZUKI, AND M. YAMASHITA, *Formation and agreement problems for synchronous mobile robots with limited visibility*, in Proceedings of the IEEE International Symposium on Intelligent Control, 1995, pp. 453–460.

[3] E. M. ARKIN, M. A. BENDER, S. P. FEKETE, J. S. B. MITCHELL, AND M. SKUTELLA, *The freeze-tag problem: How to wake up a swarm of robots*, in Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms, San Francisco, 2002, pp. 568–577.

[4] T. BALCH AND R. ARKIN, *Behavior-based formation control for multirobot teams*, IEEE Trans. Robotics and Automation, 14 (1998), pp. 926–939.

[5] Y. U. CAO, A. S. FUKUNAGA, AND A. B. KAHNG, *Cooperative mobile robotics: Antecedents and directions*, Autonomous Robots, 4 (1997), pp. 7–23.

[6] M. CIELIEBAK, P. FLOCCHINI, G. PRENCIPE, AND N. SANTORO, *Solving the robots gathering problem*, in Automata, Languages and Programming, Lecture Notes in Comput. Sci. 2719, Springer-Verlag, Berlin, 2003, pp. 1181–1196.

[7] M. Cieliebak and G. Prencipe, *Gathering autonomous mobile robots*, in Proceedings of the 9th International Colloquium on Structural Information and Communication Complexity, 2002, pp. 57–72.

[8] R. Cohen and D. Peleg, *Convergence properties of the gravitational algorithm in asynchronous robot systems*, SIAM J. Comput., 34 (2005), pp. 1516–1528.

[9] X. Defago and A. Konagaya, *Circle formation for oblivious anonymous mobile robots with no common sense of orientation*, in Proceedings of the 2nd ACM Workshop on Principles of Mobile Computing, ACM Press, New York, 2002, pp. 97–104.

[10] M. Erdmann and T. Lozano-Perez, *On multiple moving objects*, Algorithmica, 2 (1987), pp. 477–521.

[11] M. Erdmann and T. Lozano-Perez, *On multiple moving objects*, in Proceedings of the IEEE International Conference on Robotics and Automation, 1986, pp. 1419–1424.

[12] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer, *Hard tasks for weak robots: The role of common knowledge in pattern formation by autonomous mobile robots*, in Algorithms and Computation, Lecture Notes in Comput. Sci. 1741, Springer-Verlag, Berlin, 1999, pp. 93–102.

[13] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer, *Distributed coordination of a set of autonomous mobile robots*, in Proceedings of the IEEE Intelligent Vehicles Symposium (IV 2000), 2000, pp. 480–485.

[14] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer, *Gathering of autonomous mobile robots with limited visibility*, in Proceedings of the 18th Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 2010, Springer-Verlag, Berlin, 2001, pp. 247–258.

[15] D. Jung, G. Cheng, and A. Zelinsky, *Experiments in realising cooperation between autonomous mobile robots*, in Proceedings of the International Symposium on Experimental Robotics, 1997, pp. 609–620.

[16] Y. Kuniyoshi, S. Rougeaux, M. Ishii, N. Kita, S. Sakane, and M. Kakikura, *Cooperation by observation: The framework and basic task patterns*, in Proceedings of the IEEE International Conference on Robotics and Automation, 1994, pp. 767–774.

[17] L. E. Parker, *Designing control laws for cooperative agent teams*, in Proceedings of the IEEE International Conference on Robotics and Automation, 1993, pp. 582–587.

[18] L. E. Parker and C. Touzet, *Multi-robot learning in a cooperative observation task*, in Distributed Autonomous Robotic Systems 4, Springer-Verlag, New York, 2000, pp. 391–401.

[19] L. E. Parker, C. Touzet, and F. Fernandez, *Techniques for learning in multirobot teams*, in Robot Teams: From Diversity to Polymorphism, T. Balch and L. E. Parker, eds., A K Peters, Natick, MA, 2002, pp. 191–236.

[20] G. Prencipe, *CORDA: Distributed coordination of a set of atonomous mobile robots*, in Proceedings of the 4th European Research Seminar on Advances in Distributed Systems, 2001, pp. 185–190.

[21] G. Prencipe, *Instantaneous actions vs. full asynchronicity: Controlling and coordinating a set of autonomous mobile robots*, in Theoretical Computer Science, Lecture Notes in Comput. Sci. 2202, Springer-Verlag, Berlin, 2001, pp. 185–190.

[22] G. Prencipe, *Distributed Coordination of a Set of Atonomous Mobile Robots*, Ph.D. thesis, Universita Degli Studi Di Pisa, Pisa, Italy, 2002.

[23] K. Sugihara and I. Suzuki, *Distributed algorithms for formation of geometric patterns with many mobile robots*, J. Robotic Systems, 13 (1996), pp. 127–139.

[24] I. Suzuki and M. Yamashita, *Agreement on a common x-y coordinate system by a group of mobile robots* in Proceedings of the Dagstuhl Seminar on Modeling and Planning for Sensor-Based Intelligent Robots, 1996, pp. 305–321.

[25] I. Suzuki and M. Yamashita, *Distributed anonymous mobile robots: Formation and agreement problems*, in Proceedings of the 3rd Colloquium on Structural Information and Communication Complexity, 1996, pp. 313–330.

[26] I. Suzuki and M. Yamashita, *Distributed anonymous mobile robots: Formation of geometric patterns*, SIAM J. Comput., 28 (1999), pp. 1347–1363.

[27] M. Sztainberg, E. Arkin, M. Bender, and J. Mitchell, *Analysis of heuristics for the freeze-tag problem*, in Algorithm Theory—SWAT 2002, Lecture Notes in Comput. Sci. 2368, Springer-Verlag, Berlin, 2002, pp. 270–279.

[28] R. Wenger, *Helly-type theorems and geometric transversals*, in Handbook of Discrete and Computational Geometry, J. E. Goodman and J. O'Rourke, eds., CRC, Boca Raton, FL, 1997, pp. 63–82.

[29] D. Yoshida, T. Masuzawa, and H. Fujiwara, *Fault-tolerant distributed algorithms for autonomous mobile robots with crash faults*, Systems and Computers in Japan, 28 (1997), pp. 33–43.

SIAM J. COMPUT.
Vol. 36, No. 1, pp. 83–98

DISPROVING THE SINGLE LEVEL CONJECTURE*

STASYS JUKNA†

Abstract. We consider the size of monotone circuits for quadratic Boolean functions, that is, disjunctions of length-2 monomials. Our motivation is that a good (linear in the number of variables) lower bound on the monotone circuit size for a certain type of quadratic function would imply a good (even exponential) lower bound on the general nonmonotone circuit size.

To get more insight into the structure of monotone circuits for quadratic functions, we consider the so-called single level conjecture posed explicitly in the early 1990s. The conjecture claims that monotone single level circuits, that is, circuits which have only one level of AND gates, for quadratic functions are not much larger than arbitrary monotone circuits. In this paper we disprove the conjecture as follows: there exist quadratic functions whose monotone circuits have linear size but whose monotone single level circuits require almost quadratic size.

Key words. quadratic functions, monotone circuits, Boolean sums, graph complexity, clique covering number, Kneser graph, Sylvester graph

AMS subject classifications. 05C35, 05C60, 68Q17, 68R10, 94C30

DOI. 10.1137/S0097539705447001

1. Introduction.

A quadratic Boolean function is a monotone Boolean function in which all prime implicants have length 2. There is an obvious correspondence between such functions and graphs: every graph $G = (V, E)$ defines a natural quadratic function

$$f_G(X) = \bigvee_{uv \in E} x_u x_v, \tag{1.1}$$

and every quadratic function defines a unique graph. We consider the complexity of computing such functions by monotone circuits, that is, by circuits over the standard monotone basis $\{\vee, \wedge, 0, 1\}$ of fanin-2 AND and OR gates. Single level circuits are circuits in which every path from an input to an output gate contains at most one AND gate. Note that every quadratic Boolean function f_G in n variables can be computed by a trivial monotone single level circuit with at most $n-1$ AND gates using the form

$$\bigvee_{u \in S} x_u \wedge \left(\bigvee_{v : uv \in E} x_v \right), \tag{1.2}$$

where $S \subseteq V$ is an arbitrary vertex cover of G, that is, a set of vertices such that every edge of G is incident with a vertex in S.

SINGLE LEVEL CONJECTURE. *For quadratic functions, single level circuits are almost as powerful as unrestricted ones.*

Here "almost" means "up to a constant factor." This conjecture—first explicitly framed as the "single level conjecture" by Lenz and Wegener in [14]—was considered by several authors; see [12, 4, 5, 17, 14, 2] among others. That the conjecture holds

*Received by the editors February 17, 2005; accepted for publication (in revised form) November 23, 2005; published electronically May 3, 2006. This research was supported by DFG grant SCHN 503/2-2.

http://www.siam.org/journals/sicomp/36-1/44700.html

†Institute of Mathematics and Informatics, Akademijos 4, LT-08663 Vilnius, Lithuania.

for *almost all* quadratic functions was shown by Bloniarz [4] more than 25 years ago and, so far, no (even constant) gap between the size of general and single level circuits for quadratic functions is known.

In this paper we disprove the single level conjecture in a strong sense as follows: There exist quadratic functions in n variables whose monotone circuits have linear size but whose monotone single level circuits require size $\Omega(n^2/\log^3 n)$. A similar gap is also shown for Boolean formulas. We also discuss the single level conjecture in the case of monotone circuits with unbounded fanin gates.

Why should we care about monotone circuits for quadratic functions when we already can prove high (even exponential) lower bounds for monotone circuits? There are several reasons for this, as follows:

1. Any explicit n-vertex graph G that cannot be represented (in a sense described later in section 3) by a monotone circuit using fewer than cn gates for a sufficiently large constant $c > 0$ would give us an explicit *exponential* lower bound for general (nonmonotone) circuits. Let us briefly sketch how this happens. Every bipartite $n \times n$ graph $G \subseteq U \times W$ with $n = 2^m$ and $U = W = \{0,1\}^m$ gives us a Boolean function f (the characteristic function of G) in $2m$ variables such that $f(uv) = 1$ iff $uv \in G$. Suppose now that we have a *nonmonotone* circuit $F(y_1, \ldots, y_{2m})$ computing f whose inputs are variables y_i and negated variables $\overline{y}_i$; the rest of the circuit is monotone (consists of AND and OR gates). Then, according to the so-called "magnification lemma" [10], it is possible to replace its $4m = 4\log n$ input literals (both positive and negative) with appropriate Boolean sums (ORs) of variables in $X = \{x_v : v \in U \cup W\}$ so that the resulting *monotone* circuit $F_+(X)$ in $|X| = 2n$ variables represents G. It can be shown (see [21] or Lemma 3.6 below) that all $4\log n$ Boolean sums can be simultaneously computed by a monotone circuit of size cn for a constant c. Therefore, the size of F cannot be much smaller than that of F_+: $\text{size}(F) \geq \text{size}(F_+) - cn$. Hence, a lower bound $cn + n^\epsilon$ on the size of *monotone* circuits representing G would yield a lower bound $n^\epsilon = 2^{\epsilon m}$ on the *nonmonotone* circuit size of an explicit Boolean function f in $2m$ variables.

2. Better yet, for some graphs G, f_G is the *only* monotone Boolean function representing G. Such graphs are, in particular, complements of triangle-free graphs (see Observation 3.5 below). Hence, one could obtain large (even exponential) lower bounds for general nonmonotone circuits by proving a good (but only linear) lower bound on the monotone circuit size of such quadratic functions.

3. Unlike Boolean functions, graphs have been studied for a long time, and explicit constructions of graphs with very special properties are already known. We therefore hope to design a lower bound proof that is highly specialized for some particular graph or some small class of graphs. This could (probably) lead to a lower bound proof, which will not fulfill the "largeness" condition in the notion of "natural proofs" [23].

4. When applied to quadratic functions, known lower bound arguments for monotone circuits, such as Razborov's method of approximations [22] and its modifications, cannot yield lower bounds larger than n. The reason for this is that these arguments are lower bounding the *minimum* of AND gates and of OR gates needed to compute the function, and (as we already noted above) every quadratic Boolean function f_G in n variables can be computed by a trivial monotone single level circuit with at most n AND gates.

We therefore need entirely new lower bound arguments for monotone circuits

computing quadratic functions. For this, it is important to better understand the *structure* of such circuits. The (long-studied) single level conjecture seems to be a good starting point in this direction.

2. Results. Let us first introduce some notation. By the *size* of a circuit we will always mean the number of gates in it. For a monotone Boolean function f, let $C(f)$ denote the minimum number of gates and $C_{\&}(f)$ the minimum number of AND gates in a monotone circuit computing f. Let also $C^1(f)$ and $C^1_{\&}(f)$ denote the single level counterparts of these measures. Further, let $L(f)$ and $L^1(f)$ denote the minimum length of a monotone (resp., of a monotone single level) formula computing f. Recall that a *formula* is a circuit in which all gates have fanout-1, i.e., the underlying graph is a tree; the *length* of the formula is the number of leaves of this tree.

In Table 2.1 we summarize known upper and lower bounds on the *maximum* possible complexity of quadratic functions f_G over all n-vertex graphs; the upper bounds here hold for all graphs and the lower bounds for almost all graphs.

TABLE 2.1
Known bounds on the maximum complexity of quadratic functions.

Upper bounds	Lower bounds
$C^1(n) = O(n^2/\log n)$ [4]	$C(n) = \Omega(n^2/\log n)$ [4]
$L^1(n) = O(n^2/\log n)$ [26, 5, 21]	$L(n) = \Omega(n^2/\log n)$
$C^1_{\&}(n) \le n - \lfloor \log n \rfloor + 1$ [26, 14]	$C^1_{\&}(n) \ge n - c\log n$ [24]
	$C_{\&}(n) = \Omega(n/\log n)$ [14]
	$C_{\&}(n) = \Omega(n)$ [2]

In this paper we are interested in the corresponding gaps between general and single level complexities for *individual* graphs, as follows:

1. circuit gap $\mathrm{Gap}(G) = C^1(f_G)/C(f_G)$;
2. multiplicative gap $\mathrm{Gap}_{\mathrm{mult}}(G) = C^1_{\&}(f_G)/C_{\&}(f_G)$;
3. formula gap $\mathrm{Gap}_{\mathrm{form}}(G) = L^1(f_G)/L(f_G)$.

Note that the single level conjecture claims that $\mathrm{Gap}(G) = O(1)$ for all graphs G. Table 2.1 shows that, for almost all graphs, the conjecture is indeed true.

An even stronger support for the single level conjecture was given by Mirwald and Schnorr [17] as follows: if we consider circuits over the basis $\{\oplus, \wedge, 1\}$ computing (algebraic) quadratic forms $\sum_{uv\in E} x_u x_v$ over GF(2) and if we count only AND gates, then *every* optimal (with respect to the number of AND gates) circuit is a single level circuit. But the case of circuits over the basis $\{\vee, \wedge, 0, 1\}$ has remained unclear.

In the case of formulas, Krichevski [12] proved that $\mathrm{Gap}_{\mathrm{form}}(K_n) = 1$ for the complete graph K_n on n vertices, even if negation is allowed as an operation. A graph with $\mathrm{Gap}_{\mathrm{form}}(G) \ge 8/7$ was given by Bublitz [5]. In the case of multiplicative complexity, a graph with $\mathrm{Gap}_{\mathrm{mult}}(G) \ge 4/3$ was given by Lenz and Wegener [14]. Recently, this gap was substantially enlarged to $\mathrm{Gap}_{\mathrm{mult}}(G) = \Omega(n/\log n)$ by Amano and Maruoka in [2]; this was implicit also in [10]. Using a construction of Tarjan [25] (which, in its turn, was used by Tarjan for disproving that AND gates are powerless for computing Boolean sums), Amano and Maruoka [2] have also shown the gap $\mathrm{Gap}(\mathcal{F}) \ge 29/28$ for circuits computing a *set* $\mathcal{F}$ of quadratic functions. However, even the existence of a *single* graph G with $\mathrm{Gap}(G) > 1$ was not known.

Our main result is the following.

THEOREM 2.1. *There exist n-vertex graphs G such that $C(f_G) = O(n)$ but $C^1(f_G) = \Omega(n^2/\log^3 n)$. Hence,* $\mathrm{Gap}(G) = \Omega(n/\log^3 n)$.

The graphs used in Theorem 2.1 are saturated extensions of Sylvester-type graphs,

that is, of bipartite graphs whose vertices are particular vectors in $\mathrm{GF}(2)^r$, and where two vertices are adjacent iff their scalar product over $\mathrm{GF}(2)$ is 1. The *saturated extension* of a bipartite graph $H \subseteq U \times W$ is a (nonbipartite) graph $G = (V, E)$ with $V = U \cup W$ such that $E \cap (U \times W) = H$ and the induced subgraphs of G on U as well as on W are complete graphs. The reason for considering graphs of this special form lies in the simple fact (see Lemma 3.8 below) that by having a small circuit representing H we can construct a small circuit computing f_G.

To disprove the single level conjecture for formulas, we consider a bipartite version of graphs introduced by Lovász [15] in his famous proof of Kneser's conjecture [11]. A bipartite *Kneser* $n \times n$ graph is a bipartite graph $K \subseteq U \times W$, where U and W consist of all $n = 2^r$ subsets u of $\{1, \ldots, r\}$, and $uv \in K$ iff $u \cap v = \emptyset$.

THEOREM 2.2. *If G is the saturated extension of a bipartite Kneser $n \times n$ graph, then $L(f_G) = O(n \log n)$, but $L^1(f_G) \geq n^{1+c}$ for a constant $c > 0$. Hence,* $\mathrm{Gap}_{\mathrm{form}}(G) = n^{\Omega(1)}$.

Next, we consider the single level conjecture for monotone *unbounded fanin* circuits and formulas. Note that in this case single level circuits are precisely the Σ_3 circuits: the bottom (next to the inputs) level consists of OR gates, the middle level consists of AND gates, and the top level consists of a single OR gate. For a monotone Boolean function f, let $C_*(f)$ (resp., $L_*(f)$) be the minimum size of a monotone unbounded fanin circuit (resp., formula) computing f. Let also $C^1_*(f)$ and $L^1_*(f)$ denote the corresponding measures in a class of monotone Σ_3 circuits (i.e., the single level versions of these measures). Note that, also in the case of formulas, we now count the number of gates, not the number of leaves.

Single level circuits of unbounded fanin are interesting for at least two reasons, as follows:

1. The presence of unbounded fanin gates may *exponentially* increase the power of single level circuits: if, say, G is the saturated extension of an n to n matching, then $C^1(f_G) = \Omega(n)$ but $C^1_*(f_G) = O(\log n)$ (by Lemmas 3.8, 3.10, and 3.13 below).

2. By the reduction due to Valiant [27], a lower bound of the form $n^{\Omega(1)}$ on the size of a monotone Σ_3 formula representing an explicit n-vertex graph would give us a *superlinear* lower bound on nonmonotone (fanin-2) circuits of logarithmic depth, and thus, would resolve an old and widely open problem in circuit complexity (see [10] for details).

The form (1.2) implies that $C^1_*(f_G) = O(n)$ for all n-vertex graphs. On the other hand, easy counting shows $C_*(f_G) = \Omega(n)$ for almost all n-vertex graphs: every gate in a circuit of size t can have at most 2^t possible sets of immediate predecessors, implying an upper bound $2^{O(t^2)}$ on the total number of such circuits. Hence, also in the case of unbounded fanin circuits, the single level conjecture holds for *almost all* quadratic functions. The following theorem gives a stronger result: the conjecture holds for *explicit* (and large) classes of quadratic functions.

Recall that a set $S \subseteq V$ is a *vertex cover* of $G = (V, E)$ if every edge of G is incident with a vertex in S. Let $\tau(G)$ denote the minimum cardinality of a vertex cover of G. Note that for every n-vertex graph $G = (V, E)$ of maximum degree d we have $|E|/d \leq \tau(G) \leq n - 1$. Let also $\mathrm{m}(G)$ denote the maximum possible number m such that G contains a matching with m edges as an *induced* subgraph. Representation (1.2) gives us the upper bound $C^1_*(f_G) \leq L^1_*(f_G) \leq 2\tau(G) + 1$. On the other hand, we have the following lower bounds.

THEOREM 2.3. *For every graph G we have $C_*(f_G) \geq \mathrm{m}(G) + 1$. Moreover,*

$L^1_*(f_G) \geq \tau(G)/d$ and $C^1_*(f_G) \geq \sqrt{\tau(G)/d}$, where d is the maximum degree of G.

Hence, if we consider circuits with *unbounded* fanin gates, then the single level conjecture is true for all n-vertex graphs containing an induced matching with $\Omega(n)$ edges: for all such graphs we have $C_*(f_G) = \Omega(n)$ and $C^1_*(f_G) = O(n)$.

In the case of multiplicative complexity (where we count only AND gates) we have the following gap.

THEOREM 2.4. *If G is the saturated extension of an n to n matching, then $C_\&(f_G) = O(\log n)$, but $C^1_\&(f_G) = \Omega(n)$. Hence,* $\mathrm{Gap}_{\mathrm{mult}}(G) = \Omega(n/\log n)$.

This result was implicit in [10] (and even in [1]; cf. Lemma 3.10 below), where it was shown that an n to n matching M (a bipartite $n \times n$ graph consisting of n vertex disjoint edges) can be represented by a monotone conjunctive normal form (CNF) with $O(\log n)$ clauses. The proof in this case is particularly simple, and we include it just for completeness. Amano and Maruoka [2] have used a somewhat different argument to show the same gap.

TABLE 2.2
Summary of results concerning the single level conjecture.

	Known	This paper
Circuits (all gates)	$\mathrm{Gap}(\mathcal{F}) \geq 29/28$ [2] (for a set of graphs; no known gap for a single graph)	$\mathrm{Gap}(G) = \Omega(n/\log^3 n)$ Sylvester-type graphs (main result)
Formulas (all gates)	$\mathrm{Gap}_{\mathrm{form}}(K_n) = 1$ [12] $\mathrm{Gap}_{\mathrm{form}}(G) \geq 8/7$ [5]	$\mathrm{Gap}_{\mathrm{form}}(G) = n^{\Omega(1)}$ Kneser-type graphs
Circuits (AND gates)	No gap over GF(2) [17] $\mathrm{Gap}_{\mathrm{mult}}(G) \geq 4/3$ [14] $\mathrm{Gap}_{\mathrm{mult}}(G) = O(n/\log\log n)$ [2]	$\mathrm{Gap}_{\mathrm{mult}}(G) = \Omega(n/\log n)$ Perfect matchings (also in [2]; implicit in [1, 10])
Unbounded fanin (all gates)		$C_*(f_G) \geq \mathrm{m}(G) + 1$ $\tau(G)/d \leq L^1_*(f_G) \leq 2\tau(G) + 1$ $C^1_*(f_G) \geq \sqrt{\tau(G)/d}$ $\forall$ G of maximal degree d

The rest of the paper is organized as follows. In the next section we collect some preliminary definitions and technical facts. We then use these facts to prove Theorems 2.1–2.4 in sections 4–7. We conclude with several open problems.

3. Preliminaries. In this section we first recall from [10] the notion of graph representation, expose some properties of quadratic functions of saturated graphs, and recall some results about Boolean sums. We then prove some general (graph theoretic) bounds on the circuit complexity of quadratic functions.

We shall use standard graph theory notation. A set of vertices is *independent* if no two of its vertices are adjacent. A *nonedge* is a pair of nonadjacent vertices; if the graph is bipartite, then a nonedge is a pair of nonadjacent vertices from different parts (color classes); that is, pairs of vertices in one color class are neither edges nor nonedges. A *subgraph* (or a *spanning subgraph*) of a graph is obtained by deleting its edges. An *induced subgraph* is obtained by deleting vertices (together with all the edges incident to them). The main difference between these two types of subgraphs is that every nonedge of an induced subgraph is also a nonedge of the original graph. A bipartite clique $K_{a,b}$ is a complete bipartite graph with color classes of sizes a and b.

3.1. Graph representation. Every graph $G = (V, E)$ gives us a set of Boolean functions "representing" this graph in the following sense. We associate with each vertex v a Boolean variable x_v, and consider Boolean functions $f(X)$ with $X =$

$\{x_v : v \in V\}$. Such a function accepts or rejects a subset of vertices $S \subseteq V$ if it accepts or rejects the incidence vector of S. We are interested in the behavior of such functions on edges and nonedges of G, viewed as 2-element sets of their endpoints.

DEFINITION 3.1 (see [10]). *A Boolean function* represents *a given graph if it accepts all edges and rejects all nonedges.*

Hence, $f(X)$ represents the graph G if for every input vector $a \in \{0,1\}^X$ with precisely two 1's in, say, positions u and v, $f(a) = 1$ if uv is an edge, and $f(a) = 0$ if uv is a nonedge of G. If uv is neither an edge nor a nonedge (in the bipartite case) or if a contains more or less than two 1's, then the value $f(a)$ may be arbitrary.

Note that the quadratic function f_G represents the graph G in a strong sense: for every subset $S \subseteq V$, $f_G(S) = 0$ iff S is an independent set of G. But, in general, there may be many other Boolean functions representing the same graph, because they do not need to reject independent sets with more than two vertices. Hence, there are more chances to design a small circuit representing a given graph than to (directly) design a small circuit computing its quadratic function. We will use this possibility later to upper bound the circuit size of quadratic functions.

A *complete star* around a vertex u in a graph with n vertices is a set of $n-1$ edges sharing u as one of their endpoints. If the graph is bipartite, then a complete star is a set of edges joining all vertices of one part with a fixed vertex of the other part. A graph is *star-free* if it contains no complete stars. The only property of star-free graphs we will use later is given by the following simple observation.

Observation 3.2. Any monotone Boolean function representing a star-free graph must reject all its single vertices.

This is true because $f(\{u\}) = 1$, together with the monotonicity of f, implies that f must accept all edges of a complete star around u.

3.2. Saturated graphs. As noted above, besides the quadratic function f_G, there may be many other monotone Boolean functions representing G—these functions may "wrongly" accept some independent sets of G of cardinality larger than 2. The simplest way to exclude this possibility is to "kill off" all such independent sets by "saturating" the graph, i.e., by adding new edges. This way we come to the following definition.

DEFINITION 3.3. *A graph G is* saturated *if it has no independent sets with more than two vertices, that is, if the complement of G is a triangle-free graph.*

The first interesting property of quadratic functions of saturated graphs is that these functions belong to a fundamental class of so-called "slice functions" for which negation is almost powerless (see, e.g., [29, sections 6.13–6.14]). Recall that a *k-slice function* is a monotone Boolean function f such that $f(a) = 0$ for inputs a with less than k 1's, and $f(a) = 1$ for inputs a with more than k 1's, that is, $f = f \wedge T_k^n \vee T_{k+1}^n$.

Observation 3.4. If G is a saturated graph, then f_G is a 2-slice function.

Proof. Let $G = (V, E)$ be a saturated graph, and let $S \subseteq V$. If $|S| < 2$, then $f_G(S) = 0$ by the definition of quadratic functions (they cannot have prime implicants shorter than 2). If $|S| > 2$, then S cannot be an independent set since G is saturated; hence, $f_G(S) = 1$. □

The next interesting property of saturated graphs is their unique function representation.

Observation 3.5. If G is a saturated star-free graph, then f_G is the only monotone Boolean function representing G.

Proof. Let f be an arbitrary monotone Boolean function representing G. We have

to show that $f(S) = f_G(S)$ for all subsets $S \subseteq V$. If $f_G(S) = 1$, then S contains both endpoints of some edge. This edge must be accepted by f and, since f is monotone, $f(S) = 1$. If $f_G(S) = 0$, then S is an independent set of G, and $|S| \leq 2$ since G is saturated. Hence, S is either a single vertex or a nonedge. In both cases we have that $f(S) = 0$ because f must reject all nonedges and, by Observation 3.2, must also reject all single vertices. □

3.3. Boolean sums. We shall also use the following two facts about the monotone complexity of Boolean sums. The *disjunctive complexity* of a collection of Boolean sums $\bigvee_{i\in S_1} x_i, \ldots, \bigvee_{i\in S_m} x_i$ (or of the corresponding family of sets $S_1, \ldots, S_m$) is the minimum size of a circuit consisting solely of fanin-2 OR gates and simultaneously computing all these m Boolean sums.

LEMMA 3.6 (Pudlák–Rödl–Savický [21]). *For every $m \geq k \geq 1$, the disjunctive complexity of any family of m subsets of $\{1, \ldots, n\}$ does not exceed*

$$kn + k2^{\lceil m/k\rceil+1}.$$

In particular, any collection of $k \log n$ Boolean sums in n variables can be simultaneously computed by a circuit consisting solely of at most $3kn$ fanin-2 OR gates.

By this lemma, Boolean sums may not necessarily be computed separately: one partial sum computed at some OR gate may be used many times. Still, the overlap of gates cannot be too large if the sums are "disjoint enough." A family of sets is (h, k)-*disjoint* if no $h + 1$ of its members share more than k elements in common.

LEMMA 3.7 (Wegener [28]; Mehlhorn [16]). *Any (h, k)-disjoint family $S_1, \ldots, S_m$ has disjunctive complexity at least*

$$\frac{1}{kh}\sum_{i=1}^{m} |S_i| - \frac{m}{h}.$$

Proof sketch. At least $|S_i| - 1$ gates are necessary for computation of the ith sum and at least $|S_i|/k - 1$ of the functions computed at these gates are Boolean sums of more than k summands. We count only these gates. Since the family is (h, k)-disjoint, each of these gates can be useful for at most h outputs. Hence, we need at least $\sum_{i=1}^{m}(|S_i|/k - 1)/h$ gates to compute all m sums. □

3.4. Upper bounds for general circuits. An *extension* of a bipartite graph $H \subseteq U \times W$ is a (nonbipartite) graph $G = (V, E)$ with $V = U \cup W$ such that $E \cap (U \times W) = H$. The *saturated extension* is an extension whose induced subgraphs on U as well as on W are complete graphs. That is, saturated extensions consist of two disjoint cliques with some edges between these cliques. A useful property of such graphs (besides that they are saturated) is that the complexity of *computing* f_G cannot be much larger than the complexity of *representing* H: to determine the value $f_G(S)$, it is enough to additionally test whether S has more than two elements.

By the *length* of a CNF we mean the number of clauses in it.

LEMMA 3.8. *Let $H \subseteq U \times W$ be a bipartite $n \times n$ graph, G the saturated extension of H, and f a monotone Boolean function representing H. Then $f_G = (f \wedge g) \vee h$, where g is a monotone CNF of length 2 and h is an OR of $O(\log n)$ monotone CNFs of length 2. Moreover, if H is star-free, then $f_G = f \vee h$.*

Remark. Note that $C_{\&}(h) = O(\log n)$, $L(h) = O(n \log n)$, and $C(h) = O(n)$. The first two upper bounds are obvious. The third follows from Lemma 3.6.

Proof. Let $g = \left(\bigvee_{u\in U} x_u\right) \wedge \left(\bigvee_{w\in W} x_w\right)$ and $h = K_U \vee K_W$, where $K_U(S) = 1$ iff $|S \cap U| \geq 2$; that is, K_U is the quadratic function of a complete graph on U. Since

the edges of a complete graph n-vertex graph can be covered by $m \leq \lceil \log n \rceil$ bipartite cliques, each of the functions K_U and K_W has the form

$$\bigvee_{i=1}^{m} \left(\bigvee_{u \in A_i} x_u \right) \wedge \left(\bigvee_{v \in B_i} x_v \right) \tag{3.1}$$

with $m \leq \lceil \log n \rceil$ and $A_i \cap B_i = \emptyset$ for all $i = 1, \ldots, m$. Hence, h can be computed by an OR of m monotone CNFs of length 2. It remains to show that $(f \wedge g) \vee h$ coincides with f_G.

If $f_G(S) = 1$, then S contains both endpoints of some edge uv of G. This edge must be accepted either by $f \wedge g$ (if $uv \in H$) or by h (if both u and v are in the same color class). Since both $f \wedge g$ and h are monotone, the function $(f \wedge g) \vee h$ accepts S.

If $f_G(S) = 0$, then S is an independent set of G; that is, S is either a single vertex or a nonedge of H. In both cases, $h(S) = 0$ because none of the color classes can contain more than one vertex from S. Moreover, $g(S) = 0$ if S is a single vertex, and $f(S) = 0$ if S is a nonedge of H. Hence, the function $(f \wedge g) \vee h$ rejects S.

If H is star-free, then the function f alone must reject all single vertices, implying that in this case $f_G = f \vee h$. □

Lemma 3.8 gives us a simple (but useful) tool to show that a quadratic function f_G of the saturated extension of a bipartite graph H can be *computed* by a small monotone circuit: it is enough to *represent* H by a small circuit. To achieve this last goal, it is often enough to show that H has small "intersection representation."

Say that a graph G admits an *intersection representation of size* r if it is possible to associate with every vertex u a subset A_u of $\{1, \ldots, r\}$ so that $A_u \cap A_v = \emptyset$ if uv is an edge, and $A_u \cap A_v \neq \emptyset$ if uv is a nonedge of G. Let $\mathrm{int}(G)$ denote the smallest r for which G admits such a representation.

Let $\mathrm{cnf}(G)$ denote the minimum length of a monotone CNF representing the graph G, and let $\mathrm{cov}(G)$ denote the minimum number of independent sets of G covering all nonedges of G.

LEMMA 3.9 (see [9, 10]). *For every graph G, $\mathrm{cnf}(G) = \mathrm{int}(G) = \mathrm{cov}(G)$.*

The first equality was observed in [10], and the second in [9]. Both are easy to verify. If a graph $G = (V, E)$ can be represented by a CNF $\bigwedge_{i=1}^{r} \bigvee_{v \in S_i} x_v$, then the sets $A_u = \{i : u \notin S_i\}$ give the desired intersection representation of G, the r sets $I_i = \{u \in V : i \in A_u\}$ are independent and cover all nonedges of G, and the CNF of the form above with $S_i = V \setminus I_i$ represents the graph G.

Alon [1] used probabilistic arguments to prove that $\mathrm{cov}(G) = O(d^2 \log n)$ for every n-vertex graph G of maximum degree d. Hence, we have the following general upper bound.

LEMMA 3.10 (Alon [1]). *For every n-vertex graph G of maximum degree d, we have $\mathrm{cnf}(G) = O(d^2 \log n)$.*

Another possible way to show that a graph H can be represented by a small monotone circuit is to design a small nonmonotone circuit representing H, and then use the fact that negation is (almost) powerless in the context of graph representation.

LEMMA 3.11. *Let H be a bipartite $n \times n$ graph. If H can be represented by a circuit of size L over the basis $\{\vee, \wedge, \neg\}$, then H can be represented by a monotone circuit of size at most $2L + O(n)$.*

Proof. The proof is reminiscent of the proof, due to Berkowitz [3], that negation is (almost) powerless for slice functions (see also Theorem 13.1 in [29]).

Let F be a circuit of size L over the basis $\{\vee, \wedge, \neg\}$ representing a bipartite graph $H \subseteq U \times W$. Using DeMorgan rules we can transform this circuit into an equivalent

circuit F' of size at most $2L$ such that negation is used only on inputs. We then replace each negated input $\overline{x}_u$ with $u \in U$ by a Boolean sum $g_u = \bigvee_{v \in U \setminus \{u\}} x_v$, and replace each negated input $\overline{x}_w$ with $w \in W$ by a Boolean sum $h_w = \bigvee_{v \in W \setminus \{w\}} x_v$. Since all these Boolean sums can be simultaneously computed by a trivial circuit consisting of $O(n)$ OR gates (see, e.g., [29, p. 198] for a more general result), the size of the new circuit F_+ does not exceed $2L + O(n)$. Since the only difference of F_+ from the original circuit F is that negated inputs are replaced by Boolean sums, it remains to show that on arcs $ab \in U \times W$ these sums take the same values as the corresponding inputs.

Take an arbitrary set $S = \{a, b\}$ with $a \in U$ and $b \in W$. The incidence vector of this set has precisely two 1's in positions a and b. Hence, $g_u(S) = 1$ iff $a \neq u$ iff $x_u(S) = 0$ iff $\overline{x}_u(S) = 1$. Similarly, $h_w(S) = 1$ iff $b \neq w$ iff $x_w(S) = 0$ iff $\overline{x}_w(S) = 1$. Hence, on edges and nonedges of H the functions g_u and h_w take the same values as the negated variables $\overline{x}_u$ and $\overline{x}_w$, implying that F_+ represents H. □

3.5. Lower bounds for single level circuits. Given a covering $E = \bigcup_{i=1}^m A_i \times B_i$ of the edges of a graph $G = (V, E)$ by bipartite cliques, its *size* is the number m of cliques, and its *weight* is the total number $\sum_{i=1}^m (|A_i| + |B_i|)$ of vertices in these cliques. Let $\mathrm{cc}(G)$ denote the minimum size and $\mathrm{cc_w}(G)$ the minimum weight of a bipartite clique covering of G. These measures were first studied by Erdős, Goodman, and Pósa in [9], and now are the subject of an extensive literature. In particular, it is known that the maximum of $\mathrm{cc}(G)$ over all n-vertex graphs is $n - \Theta(\log n)$ [6, 26, 24], and that the maximum of $\mathrm{cc_w}(G)$ is $\Theta(n^2/\log n)$ [4, 7, 5].

For a graph G, let $\mu(G)$ be the minimum of $(a+b)/ab$ over all pairs $a, b \geq 1$ such that G contains a copy of a complete bipartite $a \times b$ graph $K_{a,b}$.

LEMMA 3.12. *For every graph G, $C^1_{\&}(f_G) = \mathrm{cc}(G)$ and $L^1(f_G) \geq \mu(G) \cdot |E|$. Moreover, if G is an extension of a bipartite graph H, then $\mathrm{cc}(G) \geq \mathrm{cc}(H)/2$ and $\mathrm{cc_w}(G) \geq \mathrm{cc_w}(H)$.*

Proof. The equalities $C^1_{\&}(f_G) = \mathrm{cc}(G)$ and $L^1(f_G) = \mathrm{cc_w}(G)$ follow immediately from the fact (shown in [4, 14]) that monotone single level circuits for quadratic functions have the form (3.1), where m is the number of AND gates in the circuit.

To show that $\mathrm{cc_w}(G) \geq \mu(G) \cdot |E|$, let $E = A_1 \times B_1 \cup \cdots \cup A_m \times B_m$ be a bipartite clique covering of $G = (V, E)$ of minimal weight. Select subsets $E_i \subseteq A_i \times B_i$ so that the E_i's are disjoint and cover the same set E of edges. Then

$$\mathrm{cc_w}(G) = \sum_{i=1}^m (|A_i| + |B_i|) = \sum_{i=1}^m \sum_{e \in E_i} \frac{|A_i| + |B_i|}{|E_i|} \geq \sum_{i=1}^m \sum_{e \in E_i} \mu(G) = \mu(G) \cdot |E|.$$

To prove the last claim, let $G = (V, E)$ be an extension of $H \subseteq U \times W$; hence, $E \cap (U \times W) = H$. If $A_i \times B_i$, $i = 1, \ldots, m$, is a bipartite clique covering of G, then $(A_i \cap U) \times (B_i \cap W)$, $(B_i \cap U) \times (A_i \cap W)$, $i = 1, \ldots, m$, is a bipartite clique covering of H. The number of bipartite cliques in this new covering is at most twice that in the original covering, and the total number of vertices in the new covering does not increase at all. □

The case of circuits, in which we count all gates (not just AND gates), is a bit more complicated because Boolean sums (entering AND gates) may not necessarily be computed separately: one partial sum computed at some OR gate may be used many times. Still, by Lemma 3.7, we know that the overlap of gates cannot be too large if the sums are disjoint enough. The disjointedness of a collection of sums $\bigvee_{i \in S_1} x_i, \ldots, \bigvee_{i \in S_m} x_i$ is naturally related to the absence of large cliques in the

incidence $m \times n$ graph of this collection, where i and j are adjacent iff $j \in S_i$: the collection of sums is (h,k)-disjoint precisely when this graph has no copies of $K_{h+1,k+1}$. Amano and Maruoka [2] used this relation to show that $C^1(f_G) \geq |E|$ for any graph $G = (V,E)$ with no copies of $K_{2,2}$; in this case the corresponding sums are $(1,1)$-disjoint. Their argument can be easily extended to yield a lower bound of the form $C^1(f_G) \geq |E|/t^{O(1)}$ for $K_{t,t}$-free graphs. However, we need superlinear lower bounds on $C^1(f_G)$ for graphs G, which are *saturated extensions* of bipartite $n \times n$ graphs H, and such graphs already have copies of $K_{t,t}$ with $t = n/4$, even if the graph H itself is $K_{2,2}$-free.

To get rid of this problem, we use a tighter analysis of single level circuits to prove a stronger result, namely, a lower bound on the minimum size $C^1(H)$ of monotone single level circuits *representing* H (recall that such a circuit must behave correctly only on edges and nonedges of H; on other inputs it may take arbitrary values). If G is an extension of H, then nonedges of H are also nonedges of G, and hence must be rejected by f_G. This means that every circuit *computing* f_G must also represent H, implying that $C^1(f_G) \geq C^1(H)$ for every extension G of H.

LEMMA 3.13. *Let $H \subseteq U \times W$ be a bipartite star-free $n \times n$ graph with no copies of $K_{t,t}$. Then $C^1(H) = \Omega(|H|/t^3)$.*

Proof. Take a minimal monotone single level circuit F representing H. The circuit F has the form $\bigvee_{i=1}^m g_i \wedge h_i$, where

$$g_i = \bigvee_{u \in S_i} x_u \text{ and } h_i = \bigvee_{v \in T_i} x_v$$

with $S_i, T_i \subseteq U \cup W$ are Boolean sums computed at the inputs of the ith AND gate. Our goal is to show that we need many OR gates to compute these sums. We cannot apply Lemma 3.7 directly to these sums because the corresponding families may not be disjoint enough. Still, we can use the absence of $K_{t,t}$ in H to show that the restriction of these families to the left part U or to the right part W of the bipartition must contain a large enough (t,t)-disjoint subfamily.

First, observe that $S_i \cap T_i = \emptyset$ because the graph H is star-free (single variables represent complete stars). Also, if for some i, both S_i and T_i lie entirely in the same part of the bipartition, then we could just remove the ith AND gate—the resulting circuit would still represent H (recall that on pairs of vertices within one part of the bipartition the circuit can take arbitrary values). Thus, we may assume that this does not happen. Hence, H is the union of bipartite cliques

$$A_i \times B_i = (S_i \cap U) \times (T_i \cap W),$$
$$A'_i \times B'_i = (T_i \cap U) \times (S_i \cap W)$$

for $i = 1, \ldots, m$. We may assume w.l.o.g. that the union H' of cliques $A_i \times B_i$, $i = 1, \ldots, m$, contains at least $|H'| \geq |H|/2$ edges of H (if not, then take the remaining bipartite cliques).

Since H' has no copies of $K_{t,t}$, for every $i = 1, \ldots, m$, at least one of the sets A_i and B_i must have fewer than t elements. Hence, if we set $I = \{i : |A_i| < t\}$, then $|B_i| < t$ for all $i \notin I$. We may assume that the bipartite graph

$$H_1 = \bigcup_{i \in I} A_i \times B_i$$

contains at least $|H_1| \geq |H'|/2 \geq |H|/4$ edges of H (if not, then let H_1 be the union of bipartite cliques $A_i \times B_i$ with $i \notin I$ and replace the roles of the A_i's and the B_i's).

This way we obtain a bipartite $K_{t,t}$-free graph $H_1 \subseteq A \times B$ with parts $A = \bigcup_{i \in I} A_i$ and $B = \bigcup_{i \in I} B_i$, and with $|H_1| \geq |H|/4$ edges. We are going to represent this graph by a monotone (single level) circuit F_1 of size not much larger than that of F, and to apply Lemma 3.7 in order to show that the size of F_1 must be large; this will yield the desired lower bound on size(F).

To achieve the first goal, we collect the Boolean sums $h_i, i \in I$, computed in F into a circuit F_1, by the construction

$$F_1(X) = \bigvee_{u \in A} x_u \wedge \left(\bigvee_{i \in I_u} h_i \right) = \bigvee_{u \in A} x_u \wedge \left(\bigvee_{i \in I_u} \bigvee_{v \in T_i} x_v \right),$$

where $I_u = \{i \in I : u \in A_i\}$. For every vertex $u \in A$, the circuit F_1 accepts an arc $uv \in A \times B$ iff $v \in T_i \cap W = B_i$ for some $i \in I$ such that $u \in A_i$. Hence, F_1 represents the graph H_1. Since all Boolean sums h_i with $i \in I$ are already computed in F, we need at most

$$\sum_{u \in A} |I_u| = \sum_{i \in I} |A_i| \leq t \cdot |I|$$

new gates to compute all functions $x_u \wedge \left(\bigvee_{i \in I_u} h_i\right)$ with $u \in A$. To compute the disjunction of these functions, we need at most $|A| \leq \sum_{i \in I} |A_i| \leq t \cdot |I|$ additional OR gates. Hence, size$(F_1) \leq$ size$(F) + 2t \cdot |I| \leq 3t \cdot$ size(F).

On the other hand, by the construction, the circuit F_1 simultaneously computes all Boolean sums $\bigvee_{i \in I_u} h_i = \bigvee_{v \in T_u} x_v$ with $u \in A$ and $T_u = \bigcup_{i \in I_u} T_i$ using only fanin-2 OR gates. Hence, size(F_1) is at least the disjunctive complexity of the family $\mathcal{T} = \{T_u : u \in A\}$. This, in its turn, is at least the disjunctive complexity of the restriction $\mathcal{T}' = \{T_u \cap W : u \in A\}$ of $\mathcal{T}$ to the set W: having a circuit for $\mathcal{T}$ we can get a circuit for $\mathcal{T}'$ just by setting to 0 all variables x_u with $u \notin W$. Observe that for every $u \in A$,

$$T_u \cap W = \bigcup_{i : u \in A_i} T_i \cap W = \bigcup_{i : u \in A_i} B_i$$

is the set of all neighbors of u in H_1. Since H_1 has no copies of $K_{t,t}$, no t vertices in A can have t common neighbors. This means that the family $\mathcal{T}'$ must be (t,t)-disjoint (in fact, even $(t-1, t-1)$-disjoint). Since $|H_1| = \sum_{u \in A} |T_u \cap W|$, Lemma 3.7 yields

$$\text{size}(F_1) \geq \frac{1}{t^2} \sum_{u \in A} |T_u \cap W| - \frac{|A|}{t} = \frac{|H_1|}{t^2} - \frac{|A|}{t}.$$

Together with the previous estimate size$(F_1) \leq 3t \cdot$ size(F) and an obvious estimate $|A| \leq t \cdot |I| \leq t \cdot$ size(F), this yields

$$\text{size}(F) \geq \frac{1}{3t} \cdot \text{size}(F_1) \geq \frac{|H_1|}{3t^3} - \frac{|A|}{3t^2} \geq \frac{|H_1|}{3t^3} - \text{size}(F).$$

Since $|H_1| \geq |H|/4$, the desired lower bound size$(F) = \Omega(|H|/t^3)$ follows. □

Now we turn to the actual proofs of Theorems 2.1–2.4.

4. Circuits: Proof of Theorem 2.1. In order to prove the gap claimed in Theorem 2.1, we need (by Lemma 3.13) a bipartite $n \times n$ graph which

1. is dense, i.e., has $\Omega(n^2)$ edges,
2. has no copies of $K_{t,t}$ with t about $\log n$,
3. can be represented by a small (linear size) monotone circuit.

The existence of graphs, satisfying the first two conditions, is a classical result of Erdős [8]. However, its proof is probabilistic and gives no idea of how to ensure the third condition. To get rid of this problem, we just reverse the order of the argument: we first choose an appropriate graph G whose induced subgraphs satisfy the third condition. Then we use the probabilistic argument to show that G must contain a sufficiently large induced subgraph satisfying the first two conditions.

Let $\mathbb{F} = \mathrm{GF}(2)$, and let r be a sufficiently large even integer. With every subset $S \subseteq \mathbb{F}^r$ we associate a bipartite graph $H_S \subseteq S \times S$ such that two vertices u and v are adjacent iff $u \cdot v = 1$, where $u \cdot v$ is the scalar product over $\mathbb{F}$. We will need the following Ramsey-type property of such graphs.

LEMMA 4.1 (Pudlák–Rödl [20]). *Suppose every vector space $V \subseteq \mathbb{F}^r$ of dimension $\lfloor (r+1)/2 \rfloor$ intersects S in less than t elements. Then neither H_S nor the bipartite complement $\overline{H}_S$ contains $K_{t,t}$.*

Proof sketch. The proof is based on the observation that any copy of $K_{t,t}$ in H_S would give us a pair of subsets X and Y of S of size t such that $x \cdot y = 1$ for all $x \in X$ and $y \in Y$. Viewing the vectors in X as the rows of the coefficient matrix and the vectors in Y as unknowns, we obtain that the sum $\dim(X') + \dim(Y')$ of the dimensions of vector spaces X' and Y', spanned by X and Y, respectively, cannot exceed $r+1$. Hence, at least one of these dimensions is at most $(r+1)/2$, implying that either $|X' \cap S| < t$ or $|Y' \cap S| < t$. However, this is impossible because both X' and Y' contain subsets X and Y of S of size t. □

In the next lemma we use the following versions of Chernoff's inequality (see, e.g., [18], section 4.1): If X is the sum of n independent Bernoulli random variables with the success probability p, then $\Pr(|X| \le (1-c)pn) \le e^{-c^2 pn/2}$ for $0 < c \le 1$, and $\Pr(|X| \ge cpn) \le 2^{-cpn}$ for $c > 2e$.

LEMMA 4.2. *There exists a subset $S \subseteq \mathbb{F}^r$ of size $|S| = 2^{r/2}$ such that neither H_S nor the bipartite complement $\overline{H}_S$ contains a copy of $K_{r,r}$.*

Proof. Let $N = 2^r$, and let $\mathbf{S} \subseteq \mathbb{F}^r$ be a random subset where each vector $u \in \mathbb{F}^r$ is included in $\mathbf{S}$ independently with probability $p = 2^{1-r/2} = 2/\sqrt{N}$. By Chernoff's inequality, $|\mathbf{S}| \ge pN/2 = 2^{r/2}$ with probability at least $1 - e^{-\Omega(pN)} = 1 - o(1)$.

Now let $V \subseteq \mathbb{F}^r$ be a subspace of $\mathbb{F}^r$ of dimension $\lfloor (r+1)/2 \rfloor = r/2$ (remember that r is even). Then $|V| = 2^{r/2} = \sqrt{N}$ and we may expect $p|V| = 2$ elements in $|\mathbf{S} \cap V|$. By Chernoff's inequality, $\Pr(|\mathbf{S} \cap V| \ge 2c) \le 2^{-2c}$ holds for any $c > 2e$. The number of vector spaces in $\mathbb{F}^r$ of dimension $r/2$ does not exceed $\binom{r}{r/2} \le 2^r/\sqrt{r}$. We can therefore take $c = r/2$ and conclude that the set $\mathbf{S}$ intersects some $(r/2)$-dimensional vector space V in $2c = r$ or more elements with probability at most $2^{r-(\log r)/2-r} = r^{-1/2} = o(1)$. Hence, with probability $1 - o(1)$, the set $\mathbf{S}$ has cardinality at least $2^{r/2}$ and $|\mathbf{S} \cap V| < r$ for every $(r/2)$-dimensional vector space V. Fix such a set S' and take an arbitrary subset $S \subseteq S'$ of cardinality $|S| = 2^{r/2}$. By Lemma 4.1, neither H_S nor $\overline{H}_S$ contains a copy of $K_{r,r}$. □

Now we turn to the actual proof of Theorem 2.1.

Proof of Theorem 2.1. Let $S \subseteq \mathbb{F}^r$ be a subset of cardinality $|S| = n = 2^{r/2}$ guaranteed by Lemma 4.2. We may assume that $u \cdot v = 1$ holds for at least half of the pairs in S (otherwise take the bipartite complement of H_S). Hence, $H = H_S$ is a

bipartite $n \times n$ graph with $n = |S|$ vertices in each part and with $|H| \geq |S|^2/2 = n^2/2$ edges. Moreover, this graph contains no copy of $K_{r,r}$ where $r = 2 \log n$.

Now let G be the saturated extension of H. By removing the centers of complete stars, we obtain an *induced* star-free subgraph H' of H. Since the graph H has no copies of $K_{r,r}$, it can have at most $2(r-1)$ complete stars, implying that the resulting subgraph H' still has $|H'| \geq |H| - 2(r-1)n = \Omega(n^2)$ edges. Moreover, every circuit representing H must also represent H' because edges/nonedges of H' are also edges/nonedges of H (this is a property of *induced* subgraphs that is not shared by spanning subgraphs) and every circuit for H must correctly accept/reject them. Therefore, $C^1(f_G) \geq C^1(H) \geq C^1(H')$ where, by Lemma 3.13, $C^1(H') = \Omega(|H'|/r^3) = \Omega(n^2/\log^3 n)$. Hence, $C^1(f_G) = \Omega(n^2/\log^3 n)$.

To get an upper bound on $C(f_G)$, let us identify each vector $w \in S$ with the set of 1-coordinates of w. Hence, two vertices u and v are adjacent in H iff $|u \cap v|$ is odd. It is not difficult to verify that (for even r) the graph H can be represented by a depth-2 formula $F(X) = \bigoplus_{i=1}^{r} \bigvee_{w \in S_i} x_w$ with $S_i = \{w \in S : i \notin w\}$. Indeed, the ith clause $\bigvee_{w \in S_i} x_w$ accepts an arc $uv \in S \times S$ iff $u \in S_i$, or $v \in S_i$ iff $i \notin u \cap v$. Hence, the formula F accepts uv iff uv is accepted by an odd number of clauses iff $|\{i : i \notin u \cap v\}| = r - |u \cap v|$ is odd iff $|u \cap v|$ is odd iff $uv \in H$.

By Lemma 3.6, all $r = 2 \log n$ Boolean sums in the formula $F(X)$ above can be simultaneously computed by a circuit of linear (in n) size. Hence, the graph H can be represented by a linear size circuit over the basis $\{\vee, \wedge, \neg\}$ and, by Lemma 3.11, can be represented by a *monotone* circuit of linear size. Since G is the saturated extension of H, Lemma 3.8 implies that $C(f_G) = O(n)$. Hence, $\mathrm{Gap}(G) = C^1(f_G)/C(f_G) = \Omega\left(n/\log^3 n\right)$. □

5. Formulas: Proof of Theorem 2.2. Let G be the saturated extension of the bipartite Kneser $n \times n$ graph $K \subseteq U \times V$. Recall that in this case U and W consist of all $n = 2^r$ subsets u of $\{1, \ldots, r\}$, and $uv \in K$ iff $u \cap v = \emptyset$. Since $\log_2 3 > 1.58$, the graph K has $|K| = \sum_{u \in U} 2^{r-|u|} = 3^r \geq n^{3/2+c}$ edges with $c \geq 0.08$. Moreover, the graph K can contain a complete bipartite $a \times b$ subgraph $\emptyset \neq A \times B \subseteq K$ only if $a \leq 2^k$ and $b \leq 2^{r-k}$ for some $0 \leq k \leq r$, because then it must hold that $\left(\bigcup_{u \in A} u\right) \cap \left(\bigcup_{v \in B} v\right) = \emptyset$. Since $a \leq a'$ and $b \leq b'$ imply $(a+b)/ab \geq (a'+b')/a'b'$, we have $\mu(K) \geq (2^k + 2^{r-k})/2^r \geq 2^{-r/2} = n^{-1/2}$. By Lemma 3.12, $L^1(f_G) = \mathrm{cc_w}(G) \geq \mathrm{cc_w}(K) \geq \mu(K) \cdot |K| \geq n^{1+c}$.

On the other hand, by its definition the graph K admits an intersection representation of size r and, by Lemma 3.9, can be represented by a monotone CNF with $\mathrm{int}(K) \leq r = \log n$ clauses, and hence, by a monotone formula with $O(n \log n)$ fanin-2 AND and OR gates. Together with Lemma 3.8, this implies that $L(f_G) = O(n \log n)$. Hence, $\mathrm{Gap}_{\mathrm{form}}(G) = L^1(f_G)/L(f_G) = \Omega(n^c/\log n)$. □

6. Unbounded fanin circuits: Proof of Theorem 2.3. To prove the lower bound $C_*(f_G) \geq \mathrm{m}(G) + 1$ we use the communication complexity argument. By an observation due to Nisan (see [19] or [13, Lemma 11.2]), $C_*(f_G)$ is at least the deterministic two-party communication complexity of f_G under the worst-case partition of its input variables (this holds for arbitrary, not necessarily quadratic, functions and for arbitrary, not necessarily monotone, circuits). Now let M be an induced matching in G with $|M| = \mathrm{m}(G)$ edges. By setting to 0 all the variables corresponding to vertices outside this matching, we obtain that $C_*(f_G) \geq C_*(f_M)$ (recall that M is an *induced* subgraph of G). The function f_M itself has the form $f_M = \bigvee_{i=1}^{|M|} x_i y_i$, i.e., is

the negation of the set disjointedness function, and its deterministic communication complexity under the natural partition, where one player gets all x_i's and the other gets all y_i's, is well known to be $|M|+1$. Hence, $C_*(f_G) \geq C_*(f_M) \geq |M|+1 = \mathrm{m}(G)+1$.

For the proof of the second part of Theorem 2.3 we need the following fact. Let $\mathrm{cnf}(f_G)$ denote the minimum length of (i.e., the number of clauses in) a monotone CNF computing f_G.

LEMMA 6.1. *For every graph G of maximum degree d, $\mathrm{cnf}(f_G) \geq \tau(G)/d$.*

Proof. Let F be a monotone CNF of length $t = \mathrm{cnf}(f_G)$ computing f_G. Since f_G has no prime implicants of length 1 (by its definition (1.1)), this CNF must contain at least two clauses. Take any of these clauses $C = \bigvee_{u \in S} x_u$ and consider the shrunken CNF $F' = F \setminus \{C\}$. Since C must accept all edges of G, each of these edges must have at least one endpoint in S. Hence, S must be a vertex cover of G, implying that $|S| \geq \tau(G)$.

Since F is a shortest CNF computing f_G, the shrunken CNF F' must make an error; i.e., it must (wrongly) accept some independent set of G. That is, there must be an independent set I such that every clause of F' contains a variable x_v with $v \in I$. Since F' has only $t-1$ clauses, we may assume that $|I| \leq t-1$. This error must be corrected by the clause C, implying that every vertex $u \in S$ must be adjacent (in G) to at least one vertex in I, for otherwise F would wrongly accept the independent set $I \cup \{u\}$ of G. Hence, at least one vertex $v \in I$ must have at least $|S|/|I| \geq \tau(G)/t$ neighbors in S. Since the degree of v cannot exceed d, the desired lower bound $t \geq \tau(G)/d$ follows. □

Now take an arbitrary graph $G = (V, E)$ of maximum degree d, and let F be a smallest monotone Σ_3 circuit computing f_G. We first consider the case when F is a formula, i.e., all gates have fanout-1. This formula is an OR $F = F_1 \vee \cdots \vee F_s$ of monotone CNFs, and $\mathrm{size}(F) \geq \sum_{i=1}^{s} r_i$, where r_i is the length of the ith CNF F_i. The CNFs F_i, $i = 1, \ldots, s$, compute quadratic functions of subgraphs $G_i = (V, E_i)$ of G such that $E_1 \cup \cdots \cup E_s = E$. Note that $\tau(G) \leq \sum_{i=1}^{s} \tau(G_i)$. Since each of these subgraphs has maximum degree at most d, Lemma 6.1 implies that the entire formula F must have size at least $\sum_{i=1}^{s} r_i \geq \sum_{i=1}^{s} \tau(G_i)/d \geq \tau(G)/d$. If F is not a formula (i.e., some OR gates on the bottom level have fanout larger than 1), then we still have that $\mathrm{size}(F) \geq t = \max\{s, r_1, \ldots, r_s\}$. Take a CNF F_i for which $\tau(G_i) \geq \tau(G)/s$. By Lemma 6.1, F_i has length $r_i \geq \tau(G_i)/d \geq \tau(G)/sd$. Since both r_i and s do not exceed t, this yields $t^2 \geq \tau(G)/d$, and the desired lower bound $t \geq \sqrt{\tau(G)/d}$ on the number of gates in F follows. □

7. Multiplicative complexity: Proof of Theorem 2.4. Let G be the saturated extension of an n to n matching M. Then, by Lemma 3.12, $C^1_{\&}(f_G) = \mathrm{cc}(G) \geq \mathrm{cc}(M)/2 = n/2$. On the other hand, M can be represented by a monotone CNF of length $O(\log n)$. This follows from the more general Lemma 3.10, but can also be shown directly as follows (see [10]): Let $r = 2\log n$, and associate with each vertex u_i on the left side its *own* $(r/2)$-element subset A_i of $\{1, \ldots, r\}$, and assign to the unique matched vertex v_i on the right side the complement B_i of A_i. It is clear then that $A_i \cap B_j = \emptyset$ iff $i = j$. Hence, $\mathrm{cnf}(M) = \mathrm{int}(M) \leq r = 2\log n$. Together with Lemma 3.8, this implies that $C_{\&}(f_G) = O(\log n)$. Hence, $\mathrm{Gap}_{\mathrm{mult}}(G) = \Omega(n/\log n)$. □

8. Concluding remarks and open problems. As we mentioned in section 2, the unbounded fanin version of the single level conjecture is true for almost all graphs.

Better yet, Theorem 2.3 implies that the conjecture is true for all n-vertex graphs containing an induced matching with $\Omega(n)$ edges. Still, it seems very unlikely that the conjecture is true for all graphs.

PROBLEM 8.1. *Does there exist n-vertex graphs G of maximal degree d with $L_*(f_G) = o(\tau(G)/d)$ or $C_*(f_G) = o(\sqrt{\tau(G)/d})$?*

Another open problem is to prove *superlinear* lower bounds on the size of monotone (fanin-2) circuits computing explicit quadratic functions in n variables. For formulas (fanout-1 circuits) lower bounds $L(f_G) = \Omega(n^{3/2})$ can be proved using the rank argument [10]. However, the case of *circuits* is more complicated because (as mentioned in the introduction) known lower bounds for monotone circuits—the method of approximations due to Razborov [22] and its derivatives—cannot yield lower bounds larger than n.

PROBLEM 8.2. *Prove $C(f_G) \geq n^{1+\epsilon}$ for an explicit n-vertex graph G.*

What can be said about the single level conjecture in the context of graph representation, that is, if we consider circuits *representing* graphs G instead of circuits *computing* their quadratic functions f_G? For circuits with fanin-2 gates the question is already answered in section 4: the gap between single level and general circuits is $\Omega(n/\log^3 n)$ also in this context. But what about circuits with unbounded fanin gates? For a graph G, let $C_*(G)$ be the minimum size of a monotone unbounded fanin circuit representing G, and let $C^1_*(G)$ be the single level version of this measure. Note that, for some graphs G, circuits representing G may be exponentially smaller than circuits computing the quadratic function f_G. If, say, M_n is a matching with n edges, then $\mathrm{cnf}(M_n) = O(\log n)$ (by Lemma 3.10) but $C_*(f_{M_n}) = \Omega(n)$ (by Theorem 2.3). This also shows that, in the context of graph representation, Lemma 6.1 no longer holds.

PROBLEM 8.3 (Pudlák–Rödl–Savický [21]). *Prove that $C^1_*(G)$ may be much larger than $C_*(G)$.*

Easy counting shows that $C^1_*(G) = \Omega(n)$ for almost all n-vertex graphs. On the other hand, as mentioned in section 2, a lower bound $n^{\Omega(1)}$ for an explicit graph G would yield a superlinear lower bound for nonmonotone log-depth circuits. Actually, even a much more moderate lower bound $2^{\alpha\sqrt{\log n}}$ with $\alpha \to \infty$ would have interesting consequences (see [10]).

PROBLEM 8.4. *Prove $C^1_*(G) \geq 2^{\alpha\sqrt{\log n}}$ for an explicit n-vertex graph G.*

Although, as mentioned above, we already can prove lower bounds $L(f_G) = \Omega(n^{3/2})$ for some explicit graphs G, doing this for *saturated* graphs is a much more difficult task. Bloniarz [4] used counting arguments to show that $C(f_G) = \Omega(n^2/\log n)$ for almost all n-vertex graphs G; this remains true also in the class of saturated graphs. The problem, however, is the *explicitness*: we want a lower bound for explicitly constructed graphs. As mentioned in the introduction, a lower bound $C(f_G) \geq cn$ for a sufficiently large constant $c > 0$ would have great consequences in circuit complexity. A (potentially) less ambitious problem is to do this for formulas.

PROBLEM 8.5. *Exhibit an explicit saturated star-free graph on n-vertices with $L(f_G) = \Omega(n \log^k n)$.*

Since, by Observation 3.5, for such graphs we have the equality $L(G) = L(f_G)$, this would yield an explicit Boolean function in $m = \Theta(\log n)$ variables requiring nonmonotone formulas of size $\Omega(m^k)$ (see [10] for details).

Acknowledgments. I am grateful to Georg Schnitger and Ingo Wegener for interesting discussions, and to the referees for numerous and very helpful suggestions concerning the presentation.

REFERENCES

[1] N. Alon, *Covering graphs by the minimum number of equivalence relations*, Combinatorica, 6 (1986), pp. 201–206.

[2] M. Amano and A. Maruoka, *On the monotone circuit complexity of quadratic Boolean functions*, in Proceedings of the 5th International Symposium on Algorithms and Computation, Lecture Notes in Comput. Sci. 3341, Springer, Berlin, 2004, pp. 28–40.

[3] S. J. Berkowitz, *On Some Relations between Monotone and Non-monotone Circuit Complexity*, Tech. rep., Comput. Sci. Dept., University of Toronto, Toronto, Ontario, Canada, 1982.

[4] P. A. Bloniarz, *The Complexity of Monotone Boolean Functions and an Algorithm for Finding Shortest Paths in a Graph*, Ph.D. Dissertation, Tech. rep. 238, Lab. Comput. Sci., MIT, Cambridge, MA, 1979.

[5] S. Bublitz, *Decomposition of graphs and monotone size of homogeneous functions*, Acta Inform., 23 (1986), pp. 689–696.

[6] F. R. K. Chung, *On the covering of graphs*, Discrete Math., 30 (1980), pp. 89–93.

[7] F. R. K. Chung, P. Erdős, and J. Spencer, *On the decomposition of graphs into complete bipartite subgraphs*, in Studies in Pure Mathematics, Birkhäuser, Basel, 1983, pp. 95–101.

[8] P. Erdős, *Some remarks on the theory of graphs*, Bull. Amer. Math. Soc., 53 (1947), pp. 292–294.

[9] P. Erdős, A. W. Goodman, and L. Pósa, *The representation of a graph by set intersections*, Canad. J. Math., 18 (1966), pp. 106–112.

[10] S. Jukna, *On graph complexity*, Combin. Probab. Comput., to appear.

[11] M. Kneser, *Aufgabe* 300, Jahresber. Dtsch. Math.-Ver., 58 (1955), p. 27.

[12] R. E. Krichevski, *Complexity of contact circuits realizing a function of logical algebra*, Sov. Phys. Dokl., 8 (1964), pp. 770–772.

[13] E. Kushilevitz and N. Nisan, *Communication Complexity*, Cambridge University Press, Cambridge, MA, 1997.

[14] K. Lenz and I. Wegener, *The conjunctive complexity of quadratic boolean functions*, Theoret. Comput. Sci., 81 (1991), pp. 257–268.

[15] L. Lovász, *Kneser's conjecture, chromatic numbers and homotopy*, J. Combin. Theory Ser. A, 25 (1978), pp. 319–324.

[16] K. Mehlhorn, *Some remarks on Boolean sums*, Acta Inform., 12 (1979), pp. 371–375.

[17] R. Mirwald and C. P. Schnorr, *The multiplicative complexity of quadratic boolean forms*, Theoret. Comput. Sci., 102 (1992), pp. 307–328.

[18] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.

[19] N. Nisan, *The communication complexity of threshold gates*, in Combinatorics, Paul Erdős Is Eighty, Vol. 1, D. Miklós, V. T. Sós, and T. Szőni, eds., János Bolyai Math. Soc., Budapest, Hungary, 1993, pp. 301–315.

[20] P. Pudlák and V. Rödl, *Pseudorandom sets and explicit constructions of Ramsey graphs*, in Complexity of Computations and Proofs, J. Krajiček, ed., Quad. Mat. 13, Dept. Math., Seconda Univ., Napoli, Caserta, 2004, pp. 327–346.

[21] P. Pudlák, V. Rödl, and P. Savický, *Graph complexity*, Acta Inform., 25 (1988), pp. 515–535.

[22] A. Razborov, *Lower bounds on the monotone complexity of some Boolean functions*, Sov. Math. Dokl., 31 (1985), pp. 354–357.

[23] A. Razborov and S. Rudich, *Natural proofs*, J. Comput. System Sci., 55 (1997), pp. 24–35.

[24] V. Rödl and A. Ruciński, *Bipartite coverings of graphs*, Combin. Probab. Comput., 6 (1997), pp. 349–352.

[25] R. Tarjan, *Complexity of monotone networks for computing conjunctions*, Ann. Discrete Math., 2 (1978), pp. 121–133.

[26] Z. Tuza, *Covering of graphs by complete bipartite subgraphs: Complexity of* 0-1 *matrices*, Combinatorica, 4 (1984), pp. 111–116.

[27] L. Valiant, *Graph-theoretic arguments in low-level complexity*, in Proceedings of the 6th Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 53, Springer, Berlin, 1977, pp. 162–176.

[28] I. Wegener, *A new lower bound on the monotone network complexity of Boolean sums*, Acta Inform., 15 (1980), pp. 147–152.

[29] I. Wegener, *The Complexity of Boolean Functions*, John Wiley, Chichester; Teubner, Stuttgart, 1987.

SIAM J. COMPUT.
Vol. 36, No. 1, pp. 99–131

RECONSTRUCTING CHROMOSOMAL EVOLUTION*

LI-SAN WANG† AND TANDY WARNOW‡

Abstract. Chromosomes evolve through genome rearrangement events, including inversions, transpositions, and inverted transpositions, that change the order and strandedness of genes within chromosomes. In this paper we present a method for estimating evolutionary histories for chromosomes based upon such events. The fundamental mathematical challenge of our approach is to estimate the *true evolutionary distance* between every pair of chromosomes, where the true evolutionary distance is the number of rearrangement events that took place in the evolutionary history between the chromosomes. We present two techniques, `Exact-` and `Approx-IEBP`, for estimating true evolutionary distances and prove guarantees about the accuracy of these techniques under a very general stochastic model of chromosomal evolution. We then show how we can use these estimated distances to obtain highly accurate estimates of chromosomal evolutionary history, significantly improving upon the previous best techniques.

Key words. phylogeny reconstruction, genome rearrangements, Markov chain, distance correction, neighbor joining, Nadeau–Taylor model, inversions, transpositions

AMS subject classifications. 05E25, 60J27, 65C50, 92B05, 92B10

DOI. 10.1137/S0097539701397229

1. Introduction.

Evolutionary tree reconstructions have for the most part been based upon analyses of biomolecular sequences evolving through *site substitutions* (also called *point mutations*). Yet, point mutations can accumulate quickly, and extensive research (both theoretical and experimental) has shown that when enough point mutations have occurred, existing approaches for estimating evolutionary histories will generally produce highly inaccurate trees [1, 8, 15, 19]. Thus, obtaining a good estimate of the evolutionary history of some datasets can be very difficult when using point mutations alone as the source of evolutionary signal.

In the last two decades, biologists have become increasingly interested in reconstructing evolution using large-scale features of whole chromosomes. The hope is that if the events that led these chromosomes to change were sufficiently rare, it might be possible to obtain better (i.e., more accurate) evolutionary history reconstructions through a proper analysis of these data.

Two of the more promising types of mutational events that have been considered by biologists are inversions and transpositions (defined later); these events affect chromosomes in two ways: they can *rearrange* the order of genes within chromosomes, and they can change the strand on which these genes appear. Since these events are much less frequent than point mutations, biologists were hopeful that evolutionary histories estimated using these rearrangement events might be more accurate than histories estimated from point mutations [24].

*Received by the editors November 1, 2001; accepted for publication (in revised form) December 21, 2005; published electronically May 26, 2006. Parts of this article appeared in the Thirty-Third Symposium on Theory of Computing (STOC'01) and the First Workshop for Algorithms on Biocomputing (WABI'01).

http://www.siam.org/journals/sicomp/36-1/39722.html

†Corresponding author. 203 Goddard Labs, Department of Biology, University of Pennsylvania, Philadelphia, PA 19104 (lswang@mail.med.upenn.edu). This author's research was supported by the National Science Foundation and the National Institutes of Health.

‡Department of Computer Sciences, Taylor Hall 2.124, University of Texas, Austin, TX 78712-1188 (tandy@cs.utexas.edu). This author's research was supported by the David and Lucile Packard Foundation and by the National Science Foundation (0121680 and 0331453).

In this paper we show how to estimate evolutionary histories for chromosomes that have evolved under inversions and transpositions. Our approach is "distance-based," which means that we first estimate a matrix of "evolutionary distances" between every pair of chromosomes in the input and then apply a tree reconstruction method which uses distances (such as neighbor joining [25]). This is a standard approach in phylogenetics, and it has been used very successfully for phylogenetic analyses from molecular sequences. However, this approach requires a mathematical technique for estimating evolutionary distances.

Techniques for estimating evolutionary distances between chromosomes have been developed prior to this work, but all have been restricted to inversion-only evolution. For example, several polynomial time methods for computing the inversion distance (see [2, 10, 14]) have been developed. However, the inversion distance is an edit distance, and hence it may underestimate the actual number of events; that is, the actual transformation of one chromosome into another may not take the shortest path. Using the inversion distance as a proxy for the evolutionary distance can thus lead in turn to inaccurate phylogenies. In addition, Sankoff and Blanchette [26] developed a technique which enabled the estimation of evolutionary distances under inversion-only models of evolution; this technique, however, was never used in phylogeny reconstruction. No technique before this work has addressed the problem of estimating evolutionary distances between chromosomes when inversions and transpositions are both allowed.

This paper makes the following progress towards the challenge of estimating evolutionary histories for chromosomes:

- We present a statistical model of the evolutionary process which allows for inversions, transpositions, and inverted transpositions; this model generalizes the *Nadeau–Taylor* model [18], and thus is called the generalized Nadeau–Taylor (GNT) model.
- We present mathematical techniques for estimating evolutionary distances and prove theorems about the accuracy of these estimations under the GNT model.
- We present a simulation study evaluating the relative performance of our new distance estimators by comparison to earlier distance estimators, as well as evaluating the accuracy of phylogenies reconstructed by neighbor joining (the benchmark distance-based method) using different ways of estimating evolutionary distances. This study shows how our new distance estimation techniques dramatically improve upon the estimation of true evolutionary distances, and also improve the accuracy of phylogeny reconstruction from chromosomes when used with neighbor joining, by comparison to early distance estimation techniques.

The organization of this paper is as follows. We define our notation in section 2 and give an overview of our mathematical technique for estimating evolutionary distances in section 3. Fundamental to our estimation technique is the ability to compute the expected "breakpoint distance" produced by a sequence of k random events in the GNT model; techniques (both exact and approximate) for these computations are presented in sections 4 and 5. In section 6, we then show how to use these computations to estimate evolutionary distances under the GNT model from whole chromosomes; part of this requires that we establish theoretical results about the fast mixing of a Markov chain we define. We provide a simulation study in section 7, evaluating both the accuracy of new distance estimators in comparison to earlier techniques and how these new distance estimation techniques impact the accuracy of phylogenies estimated using the neighbor joining [25] method. We summarize our results in section 9.

2. Definitions.

2.1. Representations of chromosomes. The genomes of some organisms have a single chromosome or contain single chromosome organelles (such as mitochondria [4, 21] or chloroplasts [20, 23]). These chromosomes can be linear or circular, depending upon their type. Whether from gene maps or from whole chromosome sequencing projects, it is possible to obtain highly accurate information about the ordering and strandedness (since chromosomes are double-stranded) of the genes within the chromosome.

In this paper, we will explore evolutionary history reconstruction for single chromosome genomes, under the assumption that the chromosomes have evolved under inversions, transpositions, and inverted transpositions. Since genes are never lost or added under these processes, our inputs are sets of chromosomes in which each gene appears once in each chromosome.

We now describe how we can represent each chromosome as a signed permutation of the integers $1, 2, \ldots, n$, where there are n genes. We assign a number to the same gene in each chromosome, arbitrarily pick one strand out of each chromosome to be the positive orientation, and thus assign each gene either a positive or negative sign to indicate its strand. In this way, each chromosome can be represented by a signed permutation of $\{1, \ldots, n\}$. Note that this representation is not unique. Each linear chromosome can be represented in two ways, so that $(1, 2, \ldots, 9, 10)$ and $(-10, -9, \ldots, -2, -1)$ represent the same linear chromosome. To represent a circular permutation, we simply break the circular permutation at an arbitrary gene and pick one of the two directions to follow; thus, circular permutations admit many representations. For example, $(1, 2, 3), (2, 3, 1)$, and $(-1, -3, -2)$ are representations for the same circular chromosome. The *canonical representation* for a circular chromosome is obtained by having gene 1 at the first position in its positive sign. Thus, the first representation in the previous example is the canonical representation.

2.2. Chromosomal rearrangement events. We now define inversions, transpositions, and inverted transpositions. Starting with a chromosome $G = (g_1, g_2, \ldots, g_n)$ an *inversion* between indices a and b, $1 \leq a < b \leq n + 1$, produces the chromosome with linear ordering

$$(g_1, g_2, \ldots, g_{a-1}, -g_{b-1}, \ldots, -g_a, g_b, \ldots, g_n).$$

If $b < a$, we can still apply an inversion to a circular (but not linear) chromosome by simply rotating the circular ordering until g_a precedes g_b in the representation, since we consider all rotations of the complete circular ordering of a circular chromosome as equivalent.

A *transposition* on the (linear or circular) chromosome G acts on three indices, a, b, c, with $1 \leq a < b \leq n$ and $2 \leq c \leq n + 1$, $c \notin [a, b]$, and operates by picking up the interval $g_a, g_{a+1}, \ldots, g_{b-1}$ and inserting it immediately after g_{c-1}. Thus, the chromosome G above (with the additional assumption of $c > b$) is replaced by

$$(g_1, \ldots, g_{a-1}, g_b, g_{b+1}, \ldots, g_{c-1}, g_a, g_{a+1}, \ldots, g_{b-1}, g_c, \ldots, g_n).$$

An *inverted transposition* is the combination of a transposition and an inversion on the transposed substring so that G is replaced by

$$(g_1, \ldots, g_{a-1}, g_b, g_{b+1}, \ldots, g_{c-1}, -g_{b-1}, -g_{b-2}, \ldots, -g_a, g_c, \ldots, g_n).$$

Inversions, transpositions, and inverted transpositions are particular examples of *chromosomal rearrangements*, which are events that change the order and strandedness of genes within chromosomes. Thus, any function that maps signed gene orders

to signed gene orders on the same set of genes is a chromosomal rearrangement. We may also consider rearrangements that change the number of copies of each gene within a chromosome; such events include duplications, insertions, and deletions of strings. However, in this paper, we will be most interested in studying just inversions, transpositions, and inverted transpositions.

2.3. Breakpoint distances. A standard way of computing a distance between chromosomes is the *breakpoint distance* [3], which we now define. Let G and G' be two chromosomes on the same set of genes, and assume each gene appears exactly once in each of the two chromosomes. Two genes x and y are *adjacent* in chromosome G if x is immediately followed by y in G or, equivalently, if $-y$ is immediately followed by $-x$ (recall that chromosomes have several equivalent representations). We then define a *breakpoint* in G with respect to G' to be an ordered pair of signed genes (x, y) such that x and y are adjacent in G but are not adjacent in G'. Just as we consider chromosomes to have equivalent representations, we consider the ordered pair (x, y) to be equivalent to $(-y, -x)$. The *breakpoint distance* between two chromosomes G and G' is the number of nonequivalent breakpoints in G with respect to G'. Therefore, the breakpoint distance is a metric since it is the cardinality of the symmetric difference between two sets. In particular, it is a symmetric function: the number of breakpoints in G with respect to G' is the same as the number of breakpoints in G' with respect to G.

An example should make this clear. Let G and G' be circular chromosomes defined by $G = (1, 2, 3, 4)$ and $G' = (1, -3, -2, 4)$. There are two breakpoints in G' with respect to G, and these are $(1, -3)$ and $(-2, 4)$. There are also two breakpoints in G with respect to G', namely $(1, 2)$ and $(3, 4)$. Thus, the breakpoint distance between G and G' is two. Note also that the pair $(2, 3)$ is not a breakpoint in either chromosome with respect to the other chromosome.

We will let $d_{BP}(G, G')$ denote the breakpoint distance between G and G'.

2.4. The Nadeau–Taylor model. The Nadeau–Taylor model was introduced in [18] and was the first stochastic model of chromosome evolution. In the Nadeau–Taylor model there is a fixed binary evolutionary tree, and the ancestral chromosome (that is, the chromosome at the root of the tree) has a single copy of each gene. This chromosome evolves down the tree solely through inversions, and any two inversions are equiprobable. The number of inversions that takes place on an edge e is a random variable which is Poisson distributed with mean λ_e. (Note that λ_e depends upon the edge e, and so the expected number of inversions can differ between the different edges.) A Nadeau–Taylor model tree is fully specified by the chromosome at the root, the tree, and the parameters λ_e (one for each edge).

2.5. The generalized Nadeau–Taylor model. The *generalized Nadeau–Taylor (GNT) model* is obtained by generalizing the Nadeau–Taylor model [18] to allow for transpositions and inverted transpositions, in addition to inversions. As in the Nadeau–Taylor model, we assume that there is a fixed binary evolutionary tree T, and the ancestral chromosome has a single copy of each gene. In the GNT model, inversions, transpositions, and inverted transpositions can occur on each edge, but any two events of the same type (two inversions, two transpositions, or two inverted transpositions) are equiprobable. We fix the relative probabilities of the three types of events across the tree. Thus, we have three parameters w_I, w_T, and w_{IT}, which denote the probability that a rearrangement is an inversion, a transposition, and an

inverted transposition, respectively. Thus, $w_I + w_T + w_{IT} = 1$. As in the Nadeau–Taylor model, we assume the number of events on each edge e is Poisson distributed with mean λ_e. Thus, the tree T, the parameters λ_e on each edge, and the three parameters w_I, w_T, and w_{IT} define the model exactly. We let $GNT(w_I, w_T, w_{IT})$ denote the set of GNT model trees with the triplet (w_I, w_T, w_{IT}).

2.6. Additive distances and trees. Let T be a binary tree with leaves labeled $s_1, s_2, \ldots, s_m$. Assume that every edge e in T has a positive length w_e. Then the *additive distance* matrix corresponding to the edge-weighted tree (T, w) is

$$D_{ij} = \sum_{e \in P_{ij}} w(e),$$

where P_{ij} is the path in T between leaves s_i and s_j.

It is well known that given the matrix $[D_{ij}]$ it is possible to reconstruct T and its edge weights w_e but not the location of the root of T (so that only the unrooted underlying tree can be reconstructed from its additive distance matrix) [29]. In fact, given a matrix $[d_{ij}]$ such that $\max_{ij} |d_{ij} - D_{ij}| < f/2$, where $f = \min\{w_e : e \in E(T)\}$, the tree T (but not its edge weights) can be reconstructed exactly [1, 15].

2.7. True evolutionary distances. Let T be the true tree for a set of chromosomes, let e be any edge in T, and let k_e be the actual number of events (whether inversions, transpositions, or inverted transpositions) that took place on edge e. Then the matrix $D_{ij} = \sum_{e \in P_{ij}} k_e$ is the matrix of *true evolutionary distances*. By construction, the matrix of true evolutionary distances is additive, and thus defines the true evolutionary tree.

3. An overview of our technique for estimating true evolutionary distances. By construction, given the matrix of true evolutionary distances we can compute the true tree and the number of events on each edge of the tree in polynomial time. Furthermore, by our earlier discussion, given a bounded inaccurate estimation of true evolutionary distances, we can still compute the true tree in polynomial time. Because highly accurate estimations of true evolutionary distances enable accurate estimations of phylogenies, statisticians have developed techniques for estimating true evolutionary distances under various stochastic models of evolution (see [15, 17] for more on this). It is for this reason that we are interested in estimating true evolutionary distances for chromosomes that have evolved under the GNT model. We now describe our basic technique for estimating true evolutionary distances between chromosomes that have evolved under the GNT model.

Suppose we are given the triplet (w_I, w_T, w_{IT}) and a positive integer k. Let G be an arbitrary (but fixed) chromosome on genes $1, 2, \ldots, n$, and let G_k be the chromosome that is the result of applying k random events to G under the $GNT(w_I, w_T, w_{IT})$ model. Note that $d_{BP}(G, G_k)$ is a random variable that depends only upon n, k, and the triplet (w_I, w_T, w_{IT}). We will show that we can compute the expected value of this random variable (i.e., $E[d_{BP}(G, G_k)]$) in polynomial time, and we can also compute a highly accurate approximation of $E[d_{BP}(G, G_k)]$ somewhat faster. Once we have an estimate of this expected breakpoint distance, we can then estimate the true evolutionary distance between an arbitrary pair of chromosomes as follows.

We begin by making a simple but significant observation. Given any pair G, G' of chromosomes, we can always relabel the genes so that under this relabeling G becomes G_0 (the unrearranged chromosome) and G' becomes G''. Because all inversions (as well as transpositions and inverted transpositions) are equally likely, we

know $E[d_{BP}(G, G')]$ and $E[d_{BP}(G_0, G'')]$ will have the same distribution; this was first mentioned explicitly in [5]. In subsequent sections, we assume one of the two input chromosomes is always the unrearranged chromosome.

- *Input*: G and G', two chromosomes on n genes, and the triplet (w_I, w_T, w_{IT}).
- *Output*: An estimation of the actual number of events (inversions, transpositions, and inverted transpositions) that took place in the evolutionary history between G and G', under the $GNT(w_I, w_T, w_{IT})$ model.
- *Method*:
  - Compute the breakpoint distance $y = d_{BP}(G, G')$ between G and G'.
  - Return the integer k that minimizes $|E[d_{BP}(G, G_k)] - y|$.

Thus, our approach for estimating true evolutionary distances relies explicitly upon techniques for estimating (or computing exactly) the expected breakpoint distance produced by k random events under the $GNT(w_I, w_T, w_{IT})$ model. Our exact calculation for the expected breakpoint distance produced by k random events takes $O(kn^2)$ time for signed circular chromosomes and $O((k+n)n^4)$ time for signed linear chromosomes. We also provide an approximation algorithm for computing the expected breakpoint distance and prove that it has very low error; furthermore, it is a simple closed form formula which can be computed in constant time, and hence it is much faster than our exact algorithm. In addition, our approximation algorithm can be applied to circular/linear and signed/unsigned chromosomes easily, and thus is a more flexible analytical tool. Both techniques, therefore, are of interest.

4. Exact calculation of the expected breakpoint distance. We show how to calculate exactly the expected breakpoint distance produced by k random events in the $GNT(w_I, w_T, w_{IT})$ model. Here we will assume that we know the triplet (w_I, w_T, w_{IT}) and thus do not need to estimate this from the data. Later in the paper we will investigate the empirical performance of using our techniques for estimating evolutionary distances and examine the consequences of using incorrect values for the triplet. We will show that even under these conditions, the trees we obtain using these distance estimates will still improve significantly upon trees obtained using standard distance estimation techniques.

4.1. Preliminaries.

Circular chromosomes. Let $G_0 = (1, 2, \ldots, n)$ be the "unrearranged" chromosome of n genes at the beginning of the evolutionary process. For any $k \geq 1$, let $\rho_1, \rho_2, \ldots \rho_k$ be k random events drawn from the $GNT(w_I, w_T, w_{IT})$ model, and let $G_k = \rho_k \rho_{k-1} \cdots \rho_1 G_0$ (i.e., G_k is the result of applying these k rearrangements to G_0). We define the function $B_i(G), 1 \leq i \leq n-1$, by setting $B_i(G) = 0$ if G has the form $(\ldots, i, i+1, \ldots)$ after some rotation and/or flipping and $B_i(G) = 1$ if not; in other words, $B_i(G) = 1$ if and only if the unrearranged chromosome G_0 has a breakpoint between i and $i+1$ with respect to G. We also set $B_n(G)$ to reflect a breakpoint between n and 1; i.e., $B_n(G) = 0$ if and only if gene 1 follows gene n in G. Let $P_{i,k} = \Pr(B_i(G_k) = 1)$; then $E[d_{BP}(G_0, G_k)] = \sum_{i=1}^{n} P_{i,k}$.

Linear chromosomes. First, we modify the representation of the linear chromosome by adding "sentinel" genes 0 and $n+1$ at the beginning and end of the ancestral chromosome G_0; these sentinel genes exist just to make the discussion easier and are never moved during the evolution of the chromosome. Thus, every chromosome we consider will begin with gene 0 and end with gene $n+1$.

Given a linear chromosome $G = (g_0 = 0, g_1, g_2, \ldots, g_n, g_{n+1} = n+1)$, we define the function $B_i(G), 0 \leq i \leq n$, as we did for circular chromosomes; thus, $B_i(G) = 1$ if and only if the unrearranged chromosome G_0 has a breakpoint between i and $i+1$

with respect to G. We define $P_{i,k}$ identically as in the case of circular chromosomes and obtain $E[d_{BP}(G_0, G_k)] = \sum_{i=0}^{n} P_{i,k}$.

4.2. Calculating the expected breakpoint distance for inversion-only evolution (i.e., under the Nadeau–Taylor model). We begin by showing how the calculation proceeds when $w_I = 1$, so that only inversions occur. This calculation was first obtained by Sankoff and Blanchette in [26].

THEOREM 1 (see [26]). *Let* $\Pr((G_k)_i = h)$ *be the probability gene* h *(with the given sign) at the* ith *position after* k *random inversions. Then*

$$\Pr((G_k)_i = h) = \Pr((G_{k-1})_i = h)\left(1 - \binom{n}{2}^{-1}(i-1)(n+1-i)\right) + \binom{n}{2}^{-1}\sum_{j=2}^{n}(\Pr((G_{k-1})_j = h)\min\{i-1, n+1-j, j-1, n+1-j\}).$$

By setting $i = 2$ and $h = 2$, and using $\Pr((G_k)_2 \neq 2) = 1 - \Pr((G_k)_2) = 2)$, we can compute the probability that a breakpoint occurs between genes 1 and 2 after k random inversions for any $k \geq 0$.

4.3. Exact calculation of the expected breakpoint distance for circular chromosomes. The previous section presented a technique for calculating the expected breakpoint distance after k random equiprobable inversions. Here we show how to use the same idea in calculating the expected breakpoint after k random events in the GNT model for circular chromosomes. We begin with some notation.

Notation. Under the GNT model, $P_{i,k}$ has the same distribution for all i, $1 \leq i \leq n$, and so $E[d_{BP}(G_0, G_k)] = nP_{1,k}$. Therefore, we will focus on the breakpoint between genes 1 and 2. Let $\mathcal{G}_n^C$ be the set of all signed circular chromosomes with n genes, and let $W_n^C = \{\pm 1, \pm 2, \ldots, \pm(n-1)\}$.

The *state representation* $L(G)$ of chromosome G is an element of W_n^C defined as follows:

- In G, do genes 1 and 2 have the same sign? If the signs are the same, then $L(G) > 0$; otherwise, $L(G) < 0$.
- To determine $|L(G)|$, we transform G into the canonical representation (so gene 1 is at the first position with positive sign). Then $|L(G)|$ is one less than the position where gene 2 is found (either in its positive or negated form) in G.

For example, for the case where $G = (1, 3, 5, 4, -2, 6)$ we have $L(G) = -4$, since gene 2 appears (in its negated form) in position 5 in the canonical representation for G.

Since $P_{1,k}$ is the probability that $B_1 = 1$ (i.e., that there is a breakpoint between g_1 and g_2) after k random events, the sign and the position of gene 2 uniquely determine $P_{1,k}$. Hence, $\{L(G_k) : k \geq 0\}$ is a homogeneous Markov chain whose state space is W_n^C. We will use these states for indexing elements in the transition matrix and the distribution vectors. For example, if M is the transition matrix for $\{L(G_k) : k \geq 0\}$, then for all i, j in W_n^C, $M_{i,j}$ is the probability of jumping to state i from state j in one step in the Markov chain.

The transition matrix for signed circular chromosomes. Let R_I, R_T, and R_{IT} be the set of all inversions, transpositions, and inverted transpositions, respectively. For every rearrangement $\rho \in R_I \cup R_T \cup R_{IT}$, we construct the 0, 1-matrix M_ρ as follows:

- For every i and j in W_n^C, $(M_\rho)_{i,j} = 1$ if $L(G) = j$ implies $L(\rho G) = i$ (i.e., ρ corresponds to a transition from state j to state i).

We then let

$$M_I = \frac{1}{|R_I|} \sum_{\rho \in R_I} M_\rho,$$

$$M_T = \frac{1}{|R_T|} \sum_{\rho \in R_T} M_\rho,$$

and

$$M_{IT} = \frac{1}{|R_{IT}|} \sum_{\rho \in R_{IT}} M_\rho.$$

The transition matrix M for $\{L(G_k) : k \geq 0\}$ is therefore

$$M = (1 - w_T - w_{IT})M_I + w_T M_T + w_{IT} M_{IT}.$$

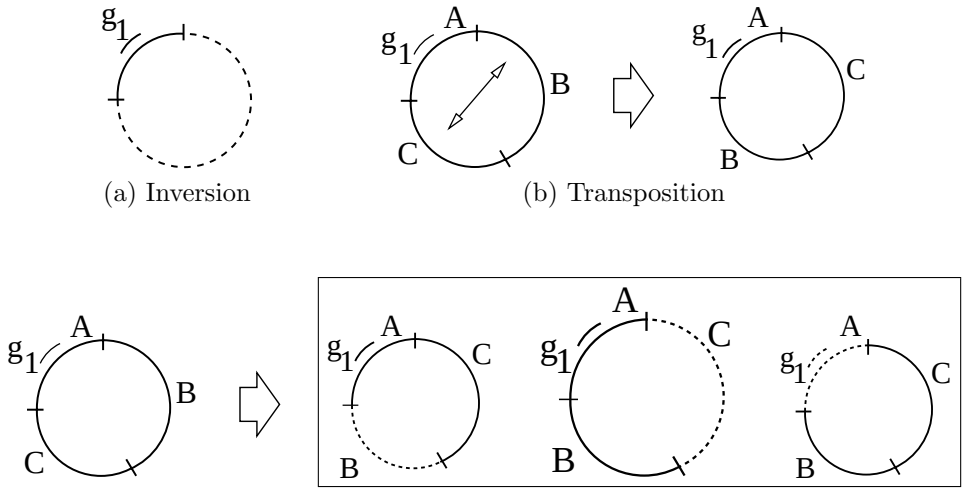
Let x_k be the distribution vector for $L(G_k)$. The following can then be easily established:

$$\begin{aligned} (x_0)_2 &= 1, \\ (x_0)_i &= 0, \quad i \in W_n^C,\ i \neq 2, \\ x_k &= M^k x_0, \\ E[d_{BP}(G_0, G_k)] &\triangleq nP_{1,k} = n(1 - (x_k)_2). \end{aligned}$$

Note that the result in Theorem 1 is just this result for the case in which only inversions occur, i.e., for which $w_I = 1$ and $w_T = w_{IT} = 0$.

We now derive closed-form formulas for the transition matrix M for the $GNT(w_I, w_T, w_{IT})$ model on signed circular chromosomes with n genes. Let $\binom{a}{b}$ denote the binomial coefficient, with $\binom{a}{b} = 0$ if $b > a$. First, consider the number of rearrangement events in each class:

1. *Inversions.* By symmetry of circular chromosomes and the model, each inversion has a corresponding inversion that inverts the complementary substring (the solid vs. the dotted arc in Figure 1(a)); thus we need only to consider the $\binom{n}{2}$ inversions that do not invert gene 1.
2. *Transpositions.* In Figure 1(b), given the three indices in a transposition, the chromosome is divided into three substrings, and the transposition swaps two substrings without changing the signs. Let the three substrings be A, B, and C, where A contains gene 1. A takes the form $(A_1, 1, A_2)$, where A_1 and A_2 may be empty. In the canonical representation there are only two possible unsigned permutations: $(1, A_2, B, C, A_1)$ and $(1, A_2, C, B, A_1)$. This means we need only to consider transpositions that swap the two substrings not containing gene 1.
3. *Inverted transpositions.* There are $3\binom{n}{3}$ inverted transpositions. In Figure 1(c), given the three endpoints in an inverted transposition, exactly one of the three substrings changes signs. Using the canonical representation, we interchange the two substrings that do not contain gene 1 and invert one of them (the first two chromosomes right of the arrow in Figure 1(c)), or we invert both substrings without swapping (the rightmost chromosome in Figure 1(c)).



(a) Inversion (b) Transposition

(c) Inverted transposition

FIG. 1. *The three types of rearrangement events in the GNT model on a signed circular chromosome.* (a) *We need only to consider inversions that do not invert gene* 1. (b) *A transposition corresponds to swapping two substrings.* (c) *The three types of inverted transpositions. Starting from the left chromosome, the three distinct results are shown here; the broken arc represents the substring being transposed and inverted.*

For all u and v in W_n^C, let $\iota_n(u,v)$, $\tau_n(u,v)$, and $\nu_n(u,v)$ be the numbers of inversions, transpositions, and inverted transpositions that bring a gene in state u to state v. Then

$$M_{u,v} = (1 - w_T - w_{IT})(M_I)_{u,v} + w_T(M_T)_{u,v} + w_{IT}(M_{IT})_{u,v}$$

$$= \frac{1 - w_T - w_{IT}}{\binom{n}{2}}\iota_n(u,v) + \frac{w_T}{\binom{n}{3}}\tau_n(u,v) + \frac{w_{IT}}{3\binom{n}{3}}\nu_n(u,v).$$

The following lemma gives formulas for $\iota_n(u,v)$, $\tau_n(u,v)$, and $\nu_n(u,v)$.

LEMMA 1. *For all u and v in W_n^C, let $\iota_n(u,v)$, $\tau_n(u,v)$, and $\nu_n(u,v)$ be the numbers of inversions, transpositions, and inverted transpositions that bring a gene in state u to state v (n is the number of genes in each chromosome). Then*

$$\text{(a) } \iota_n(u,v) = \begin{cases} \min\{|u|, |v|, n-|u|, n-|v|\} & \text{if } uv < 0, \\ 0 & \text{if } u \neq v, uv > 0, \\ \binom{|u|}{2} + \binom{n-|u|}{2} & \text{if } u = v, \end{cases}$$

$$\text{(b) } \tau_n(u,v) = \begin{cases} 0 & \text{if } uv < 0, \\ (\min\{|u|, |v|\})(n - \max\{|u|, |v|\}) & \text{if } u \neq v, uv > 0, \\ \binom{|u|}{3} + \binom{n-|u|}{3} & \text{if } u = v, \end{cases}$$

$$\text{(c) } \nu_n(u,v) = \begin{cases} (n-2)\iota_n(u,v) & \text{if } uv < 0, \\ \tau_n(u,v) & \text{if } u \neq v, uv > 0, \\ 3\tau_n(u,v) & \text{if } u = v. \end{cases}$$

Proof. The proof of (a) was provided in [26] and is omitted.

We now prove (b). Consider the gene with state u. Let v be the new state of that gene after the transposition with indices (a, b, c), $2 \le a < b < c \le n+1$. Since transpositions do not change the sign, $\tau_n(u, v) = \tau_n(-u, -v)$, and $\tau_n(u, v) = 0$ if $uv < 0$. Therefore we need only to analyze the case where $u, v > 0$.

We first analyze the case when $u = v$. Assume that either $a \le u < b$ or $b \le u < c$. In the first case, from the definition in section 2 we immediately have $v = u + (c - b)$, from which it follows that $v - u = c - b > 0$. In the second case, we have $v = u + (a - b)$, so that $v - u = a - b < 0$. Both cases contradict the assumption that $u = v$, and the only remaining possibilities that make $u = v$ are when $1 \le u = v < a - 1$ or $c - 1 \le u = v \le n - 1$. This leads to the third line in the $\tau_n(u, v)$ formula. Next, the total number of solutions (a, b, c) for the following two problems is $\tau_n(u, v)$ when $u \ne v$ and $u, v > 0$:

(i) $u < v : b = c - (v - u)$, $2 \le a \le u + 1 < b < c \le n + 1, u < v \le c$.

(ii) $u > v : b = a + (u - v)$, $2 \le a < b \le u + 1 < c \le n + 1, a \le v < u$.

In the first case $\tau_n(u, v) = u(n - v)$, and in the second case $\tau_n(u, v) = v(n - u)$. The second line in the $\tau_n(u, v)$ formula follows by combining the two results.

For inverted transpositions there are three distinct subclasses of rearrangement events. The result in (c) follows by applying the above method to the three cases. ∎

We now derive the running time for this exact method.

THEOREM 2. *It takes $O(kn^2)$ time to compute the expected breakpoint distance produced by k random events under the GNT model on circular chromosomes on n genes, using the exact technique.*

Proof. We first compute the $2(n-1) \times 2(n-1)$ transition matrices M_I, M_T, M_{IT} in $O(n^2)$ time using the closed-form formulas above (each entry takes $O(1)$ time). Computing M from the three matrices takes $O(n^2)$ time. Finally, computing the expected breakpoint distance requires k matrix-vector multiplications, which takes $O(kn^2)$ time. ∎

4.4. Exact calculation of the expected breakpoint distance for linear chromosomes. When the chromosomes are linear, different breakpoints will have different distributions. Thus, to compute the distribution of a given breakpoint, we will need to consider the positions and the signs of the genes involved at the same time. We begin with some notation.

Notation. Let $\mathcal{G}_n^L$ be the set of all signed linear chromosomes, and let $W_n^L = \{(u, v) : u, v = \pm 1, \ldots, \pm n, |u| \ne |v|\}$. We define the state representations $J_i : \mathcal{G}_n^L \to W_n^L$, $i = 1, \ldots, n-1$, as follows. For any chromosome $G \in \mathcal{G}_n^L$, $J_i(G) = (u, v)$, where

- $|u|$ is the position of gene i, and the sign of u is the same as that of gene i;
- $|v|$ is the position of gene $i + 1$, and the sign of v is the same as that of gene $i + 1$.

For example, in the chromosome $G = (1, 4, 2, 5, 6, -3, -7, 8)$ we have $J_3(G) = (-6, 2)$ (based on the positions and signs of genes 3 and 4), and $J_7(G) = (-7, 8)$ (based on the positions and signs of genes 7 and 8). Thus, by examining $J_i(G)$ we can determine if the genes i and $i + 1$ are "adjacent" (meaning that gene $i + 1$ *follows* gene i) in G; thus, genes i and $i + 1$ are adjacent if and only if $J_i(G_k) = (u, u + 1)$ for some integer u (note that u could be negative).

Therefore $\{J_i(G_k) : k \ge 0\}, 1 \le i \le n - 1$, forms a homogeneous Markov chain with state space W_n^L. As before we use the states in W_n^L as indices to the transition matrix and the distribution vectors.

Let $x_{\langle i,k\rangle}$ be the distribution vector of $J_i(G_k)$. For every rearrangement $\rho \in R_I, R_T$, and R_{IT}, the matrix M_ρ is defined as for circular chromosomes, with different dimensions. We let

$$M_I = \frac{1}{|R_I|} \sum_{\rho \in R_I} M_\rho,$$

$$M_T = \frac{1}{|R_T|} \sum_{\rho \in R_T} M_\rho,$$

and

$$M_{IT} = \frac{1}{|R_{IT}|} \sum_{\rho \in R_{IT}} M_\rho.$$

The transition matrix M for linear chromosomes has the same form as the transition matrix for circular chromosomes, i.e.,

$$M = (1 - w_T - w_{IT})M_I + w_T M_T + w_{IT} M_{IT}.$$

Let e be a vector where $e_{(u,v)} = 1$ whenever $v = u + 1$, and 0 otherwise. Then

$$e^T x_{\langle i,k\rangle} = \sum_{u=v+1} (x_{\langle i,k\rangle})_{(u,v)}$$

is the probability there is no breakpoint between genes i and $i+1$ in G_0 with respect to G_k. Therefore

$$\begin{aligned}
(x_{\langle i,0\rangle})_{(i,i+1)} &= 1, \\
(x_{\langle i,0\rangle})_{(u,v)} &= 0, \quad (u,v) \in W_n^L,\ (u,v) \neq (i,i+1), \\
x_{\langle i,k\rangle} &= M^k x_{\langle i,0\rangle}, \\
P_{i,k} &= 1 - e^T x_{\langle i,k\rangle} \ = \ 1 - e^T M^k x_{\langle i,0\rangle}.
\end{aligned}$$

Since the two sentinel genes 0 and $n+1$ never change their positions and signs, the distributions of the two breakpoints B_0 and B_n depend on the state of one gene each (1 and n, respectively). Hence we can use the results from circular chromosomes to estimate $P_{0,k}$ and $P_{n,k}$. Since these two quantities have the same distribution under the GNT model, the expected breakpoint distance after k events is

$$\begin{aligned}
E[d_{BP}(G_0, G_k)] &= \sum_{i=0}^{n} P_{i,k} = 2P_{0,k} + \sum_{i=1}^{n-1} P_{i,k} = 2P_{0,k} + \sum_{i=1}^{n-1} (1 - e^T M^k x_{\langle i,0\rangle}) \\
&= 2P_{0,k} + (n-1) - e^T M^k \sum_{i=1}^{n-1} x_{\langle i,0\rangle}. \qquad (1)
\end{aligned}$$

Constructing transition matrices. We do not have a simple closed-form formula to construct M_I, M_T, or M_{IT} as in the circular-chromosome case. Instead, we compute M_ρ for every rearrangement ρ and sum over them to obtain the three matrices according to their definition. We use the following lemma to simplify the computation.

LEMMA 2. *For every rearrangement ρ transforming $G_0 = (0, 1, 2, \ldots, n, n+1)$ into $G = (0, g_1, g_2, \ldots, g_n, n+1)$, we have $(M_\rho)_{(x,x'),(y,y')} = 1$ (i.e., ρ transforms state (y, y') into (x, x')) if*

1. *gene $|y|$ is the $|x|$th gene in G (g_1 is the first gene), and the sign of gene $|y|$ in G is the same as the sign of xy;*
2. *gene $|y'|$ is the $|x'|$th gene in G (g_1 is the first gene), and the sign of gene $|y'|$ in G is the same as the sign of $x'y'$.*

$(M_\rho)_{(x,x'),(y,y')} = 0$ otherwise.

Proof. Consider any chromosome G'. If $L_i(G') = (y, y')$, then gene i is at position $|y|$ having the same sign as the number y. Assume in ρG_0 that gene $|y|$ is at position $|x|$, and the sign of gene $|y|$ is the same as the sign of number x; this implies that gene i is in position $|x|$ in $\rho G'$. Moreover, gene i changes sign if and only if $x < 0$; so the sign of gene i is the same as xy. □

The lemma above provides us with an efficient algorithm for constructing M_I, M_T, and M_{IT}. We give the details in the next theorem.

THEOREM 3. *Assume the chromosome is linear with n distinct genes. It takes $O((k+n)n^4)$ time to compute the expected breakpoint distance produced by k random events under the GNT model, using the exact technique.*

Proof. By (1), we need to compute $M^k \sum_{i=1}^{n-1} x_{\langle i,0\rangle}$ and $P_{0,k}$. $P_{0,k}$ is easy because it is equal to $P_{1,k}$ for circular chromosomes. On the other hand, computing $M^k \sum_{i=1}^{n-1} x_{\langle i,0\rangle}$ means we have to compute M_I, M_T, and M_{IT}, for which we do not have simple closed-form formulas (unlike for the case of circular chromosomes). Moreover, the number of states (and the number of rows or columns for the transition matrix) is $O(n^2)$.

To compute M_I, M_T, and M_{IT}, we need only to apply every rearrangement to the unrearranged chromosome and check which entry is set to 1 for each column in the matrix. We use M_I as an example:

1. We first initialize every entry in M_I to 0.
2. For every inversion ρ, we apply it to the unrearranged chromosome G_0 to obtain $G = \rho G_0$ and do the following:
   (a) For each column with corresponding state (y, y'), we use the lemma above together with G to find the resulting state (x, x'); we then increase $(M_I)_{(x,x'),(y,y')}$ by 1.
3. We divide M_I by $\binom{n}{2}$ to obtain the final matrix.

We now analyze the running time for the algorithm above. Initializing M_I takes $O(n^4)$ time, since there are $2^2\binom{n}{2}$ distinct states. There are $\binom{n}{2}$ distinct inversions; for each inversion ρ, computing ρG_0 takes $O(n)$ time. Line 2(a) takes $O(n^2)$ time for each ρ, since there are $2^2\binom{n}{2}$ columns in M_I. So the total running time is $O(n^4 + \binom{n}{2} \times (n + 2^2\binom{n}{2})) = O(n^4)$. Similar arguments show the running times for constructing M_T and M_{IT} are both $O(n^5)$, since there are $O(n^3)$ distinct transpositions and inverted transpositions.

Computing $P_{0,k}$ and $P_{n,k}$ takes $O(kn^2)$ time using the circular-chromosome algorithm. To compute $P_{i,k}$, $1 \le i \le n-1$, we first compute M_I in $O(n^4)$ time and M_T and M_{IT} in $O(n^5)$ time. Computing $\sum_{i=1}^{n-1} x_{\langle i,0\rangle}$ takes $O(n^3)$ time. We then evaluate k matrix-vector multiplications to obtain $M^k \sum_{i=1}^{n-1} x_{\langle i,0\rangle}$; this step takes $O(kn^4)$ time. It then takes $O(n^2)$ time to obtain $E[d_{BP}(G_0, G_k)]$ from these results. □

Note that if only inversions are present, the running time becomes $O(kn^4)$.

5. Approximating the expected breakpoint distance.

5.1. Introduction. In this section we present a faster, but approximate rather than exact, technique, $\mathcal{F}_k$, for estimating the expected breakpoint distance produced by k random events under the GNT model. Our estimation is obtained through averaging tight lower and upper bounds, and we show the relative and absolute error of the approximation is small.

5.2. Extending the model. We use a model more general than the GNT model for the derivation to allow more general results for our approximation. We formulate this more general model as follows. A rearrangement class $\mathcal{E}$ acting on $\mathcal{G}_n$ (depending on the context of the problem, $\mathcal{G}_n$ can be either circular or linear) is a pair $(A(\mathcal{E}), f_{\mathcal{E}})$, where $A(\mathcal{E})$ is a set of rearrangements with nonzero probability of taking place, and $f_{\mathcal{E}}(\rho|G)$ is the probability that rearrangement ρ takes place on chromosome G, for a given $\rho \in A(\mathcal{E})$ and $G \in \mathcal{G}_n$. We say the random variable ρ is of rearrangement class $\mathcal{E}$ acting on chromosome G if ρ is in $A(\mathcal{E})$ and has distribution $f_{\mathcal{E}}(\rho|G)$.

Following the notation in section 2, we now present the derivation of our approximation algorithm. Assume the rearrangement to act on G is ρ. We recall the definitions of $B_i, 1 \le i \le n$, and $P_{i,k}$: $B_i(G) = 0$ if genes i and $i+1$ are adjacent in G_0 (with respect to G) and $B_i(G) = 1$ if not. G_k is the result of applying k random events to the unrearranged chromosome $(1, 2, \ldots, n)$. $P_{i,k}$ is defined to be $\Pr(B_i(G_k) = 1)$. Thus, $E[d_{BP}(G_0, G_k)] = \sum_{i=1}^{n} P_{i,k}$ for circular chromosomes and $E[d_{BP}(G_0, G_k)] = \sum_{i=0}^{n} P_{i,k}$ for linear chromosomes.

We make the following definitions:

- $s(i|G, \mathcal{E}) = \Pr(B_i(\rho G) = 1 \mid B_i(G) = 0)$,
- $u(i|G, \mathcal{E}) = \Pr(B_i(\rho G) = 0 \mid B_i(G) = 1)$,
- $\mathrm{Sep}(i|G, \mathcal{E}) = \{\rho \in A(\mathcal{E}) : B_i(\rho G) = 1\}$, and
- $\mathrm{Uni}(i|G, \mathcal{E}) = \{\rho \in A(\mathcal{E}) : B_i(\rho G) = 0\}$.

We focus on rearrangement classes $\mathcal{E}$ where $f_{\mathcal{E}}$ is independent of k and G, and $s(i|G, \mathcal{E})$ is independent of G. All three rearrangement classes in the GNT model, namely the class of random inversions, the class of random transpositions, and the class of random inverted transpositions, satisfy these requirements.

We now show the derivation and properties of our true evolutionary distance estimator. We start in section 5.3 with the simple case of rearrangement event classes where the breakpoints satisfy the Markov property and find the expected number of breakpoints after k random rearrangements. The result is extended in section 5.4, where the requirement on the Markov property is relaxed; the result is an approximation to the expected number of breakpoints. The error bounds on the approximation are shown in section 5.5. The main result is in section 5.6, where we develop the technique for rearrangement classes that are mixtures of other rearrangement classes. The technique is then applied to the GNT model of chromosome evolution in section 5.7.

5.3. Single rearrangement class models where the breakpoints satisfy the Markov property. We start with a simple case by considering any rearrangement class $\mathcal{E}$ for which $s(i|G, \mathcal{E})$ and $u(i|G, \mathcal{E})$ are independent of the past history and the current chromosome G to be acted upon. Then $\{B_i(G_k), k \ge 0\}$ is a Markov process (see Figure 2), as is shown in the following theorem.

THEOREM 4. *Assume $\mathcal{E}$ is a class of rearrangements such that $s(i|G, \mathcal{E})$ and $u(i|G, \mathcal{E})$ do not depend upon chromosome G. Let their common values be $s(i|\mathcal{E})$ and*

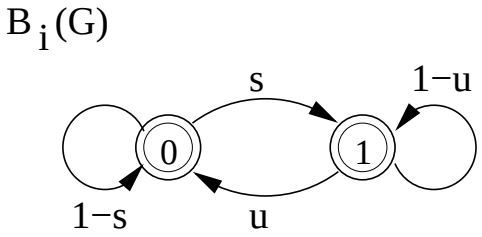


FIG. 2. *Each breakpoint can be regarded as a two-state stochastic process with two parameters s and u (see section 5.4).*

$u(i|\mathcal{E})$, *respectively. Then*

$$P_{i,k} = s(i|\mathcal{E}) \left(\frac{1 - (1 - s(i|\mathcal{E}) - u(i|\mathcal{E}))^k}{1 - (1 - s(i|\mathcal{E}) - u(i|\mathcal{E}))} \right).$$

Proof. We have the following recurrence:

$$\begin{aligned}
s(i|\mathcal{E}) &= \Pr(\rho_k \in \mathrm{Sep}(i|G_k, \mathcal{E}) \mid B_i(i|G_k) = 0) \\
&= \Pr(B_i(G_{k+1}) = 1 \mid B_i(G_k) = 0) \\
&= \frac{\Pr(B_i(G_{k+1}) = 1 \cap B_i(G_k) = 0)}{1 - P_{i,k}}, \\
u(i|\mathcal{E}) &= \Pr(\rho_k \in \mathrm{Uni}(i|G_k, \mathcal{E}) \mid B_i(G_k) = 1), \\
&= \Pr(B_i(G_{k+1}) = 0 \mid B_i(G_k) = 1) \\
&= \frac{\Pr(B_i(G_{k+1}) = 0 \cap B_i(G_k) = 1)}{P_{i,k}}, \\
P_{i,k+1} &= \Pr(B_1(G_{k+1}) = 1) = (1 - P_k)s(i|\mathcal{E}) + P_k(1 - u(i|\mathcal{E})), \\
&= P_k(1 - s(i|\mathcal{E}) - u(i|\mathcal{E})) + s(i|\mathcal{E}), \\
P_{i|0} &= 0.
\end{aligned}$$

The proof follows by solving the recurrence. □

COROLLARY 1. *Let G_k be the result of applying k random inversions to the unsigned linear chromosome G_0 having n genes. If G_k is linear,*

$$E[d_{BP}(G_0, G_k)] = (n-1)\left(1 - \left(\frac{n-3}{n-1}\right)^k\right),$$

and if G_k is circular,

$$E[d_{BP}(G_0, G_k)] = \frac{n(n-3)}{n-1}\left(1 - \left(\frac{n-4}{n-2}\right)^k\right).$$

Proof. The proof follows from Theorem 4, with parameters from Table 1. The linear case is originally in [5] with similar arguments, and the circular case is a simple extension. □

5.4. The lower and upper bounds technique for single rearrangement class models. For many other classes of rearrangements, the parameters regarding transitions of $B_i(G)$'s state depend not only on $B_i(G)$ but on other properties of G as

well. For example, the number of inversions that makes genes 1 and 2 adjacent depends on the number of genes between these two genes. However, for the rearrangement classes $\mathcal{E}$ where $s(i|G,\mathcal{E})$ does not depend on G, we can obtain upper and lower bounds on the expected number of breakpoints, as we now show.

Let $u_{min}(i|\mathcal{E})$ and $u_{max}(i|\mathcal{E})$ be the lower and upper bounds of $u(i|G,\mathcal{E})$ over all chromosomes G. Observe that a larger value of $u(i|G,\mathcal{E})$ means that genes i and $i+1$ are more likely to be made adjacent, given that they are currently not adjacent. This means $P_{i,k}$, the probability of having a breakpoint between gene i and $i+1$ after k rearrangements, is monotone decreasing on $u(i|G,\mathcal{E})$.

THEOREM 5. *Assume $\mathcal{E}$ is a class of rearrangements such that $s(i|\mathcal{E})$ is independent of the chromosome G currently acted upon. Let $u_{min}(i|\mathcal{E})$ and $u_{max}(i|\mathcal{E})$ be defined as in the previous paragraph. We have $P^L_{i,k} \le P_{i,k} \le P^H_{i,k}$ for all k, where*

$$P^L_{i,k} = s(i|\mathcal{E})\left(\frac{1-(1-s(i|\mathcal{E})-u_{max}(i|\mathcal{E}))^k}{1-(1-s(i|\mathcal{E})-u_{max}(i|\mathcal{E}))}\right),$$
$$P^H_{i,k} = s(i|\mathcal{E})\left(\frac{1-(1-s(i|\mathcal{E})-u_{min}(i|\mathcal{E}))^k}{1-(1-s(i|\mathcal{E})-u_{min}(i|\mathcal{E}))}\right).$$

Proof. The two recursions determined by $u_{min}(i|\mathcal{E})$ and $u_{max}(i|\mathcal{E})$ can be solved using Theorem 4. We prove the inequality bounding $P_{i,k}$ by $P^L_{i,k}$ and $P^H_{i,k}$ for all $k \ge 0$ by induction. When $k = 0$, all three quantities are 0, and so the base case holds. The induction step is as follows:

$$\begin{aligned} P^L_{i,k+1} &= P^L_{i,k}(1-s(i|\mathcal{E})-u_{max}(i|\mathcal{E})) + s(i|\mathcal{E}) \\ &\le P_{i,k}(1-s(i|\mathcal{E})-u(i|G_k,\mathcal{E})) + s(i|\mathcal{E}) = P_{i,k+1} \\ &\le P^H_{i,k}(1-s(i|\mathcal{E})-u_{min}(i|\mathcal{E})) + s(i|\mathcal{E}) = P^H_{i,k+1}. \end{aligned}$$ □

COROLLARY 2. *Given two random signed circular chromosomes G and G' on n genes, $n \ge 2$,*

$$E[d_{BP}(G,G')] = \frac{n(n-1.5)}{n-1}.$$

Proof. The expected breakpoint distance between two random chromosomes is the same as the breakpoint distance between an unrearranged chromosome G_0 and a random chromosome G. In the canonical representation, gene 1 is always positive and at the first position in both chromosomes.

Since G_0 is the unrearranged chromosome, $G_0 = (1, 2, \ldots, n)$; however, there are $2(n-1)$ equally probable choices regarding the sign and position of gene 2 in G. The probability of a breakpoint between genes 1 and 2 in G (with respect to G_0) is thus exactly $(2(n-1)-1)/(2(n-1)) = (n-1.5)/(n-1)$. The theorem follows since the other breakpoints $(i, i+1)$ have the same probability as $(1, 2)$. □

This result is apparently new; see [4] for a previous estimate, which this corrects.

DEFINITION 1. *Given any class of rearrangements $\mathcal{E}$ that satisfies the assumption in Theorem* 5, *we set*

$$\mathcal{F}_k = \sum_{i=0}^{n} \frac{P^L_{i,k} + P^H_{i,k}}{2}.$$

The function $\mathcal{F}_k$ is an approximation to the expected number of breakpoints after k random rearrangements drawn from $\mathcal{E}$.

Note that $\mathcal{F}_k$ is strictly monotone increasing with respect to k, since both $P^L_{i,k}$ and $P^H_{i,k}$ are strictly monotone increasing with respect to k. This observation guarantees that the integer k minimizing $|\mathcal{F}_k - x|$ is unique (up to a possible tie between two integers that differ by 1).

5.5. Error bounds on the technique using upper and lower bounds. In this section we bound the absolute and relative errors of the estimator $\mathcal{F}_k$ with respect to $E[d_{BP}(G_0, G_k)]$. Let

- $R^L_i = 1 - s(i|\mathcal{E}) - u_{max}(i|\mathcal{E})$, and
- $R^H_i = 1 - s(i|\mathcal{E}) - u_{min}(i|\mathcal{E})$.

Note $(R^L_i)^k \le (R^H_i)^k$ for all $k \ge 0$. We now bound the error of the estimator $\mathcal{F}_k$.

LEMMA 3.

$$\frac{1}{2}(P^H_{i,k} - P^L_{i,k}) \le \frac{u_{max}(i|\mathcal{E}) - u_{min}(i|\mathcal{E})}{2\ s(i|\mathcal{E})}.$$

Proof.

$$\frac{1}{2}(P^H_{i,k} - P^L_{i,k}) = \frac{1}{2}s(i|\mathcal{E})\left(\frac{1-(R^H_i)^k}{1-R^H_i} - \frac{1-(R^L_i)^k}{1-R^L_i}\right) = \frac{1}{2}s(i|\mathcal{E})\sum_{j=0}^{k-1}((R^H_i)^j - (R^L_i)^j)$$

$$\le \frac{1}{2}s(i|\mathcal{E})\sum_{j=0}^{\infty}((R^H_i)^j - (R^L_i)^j) = \frac{1}{2}s(i|\mathcal{E})\left(\frac{1}{1-R^H_i} - \frac{1}{1-R^L_i}\right)$$

$$= \frac{s(i|\mathcal{E})(u_{max}(i|\mathcal{E}) - u_{min}(i|\mathcal{E}))}{2(s(i|\mathcal{E}) + u_{min}(i|\mathcal{E}))(s(i|\mathcal{E}) + u_{max}(i|\mathcal{E}))}$$

$$\le \frac{u_{max}(i|\mathcal{E}) - u_{min}(i|\mathcal{E})}{2\ s(i|\mathcal{E})}. \qquad \square$$

THEOREM 6.

$$|\mathcal{F}_k - E[d_{BP}(G_0, G_k)]| \le \sum_{i=0}^{n} \frac{u_{max}(i|\mathcal{E}) - u_{min}(i|\mathcal{E})}{2s(i|\mathcal{E})} \quad \forall k \ge 0.$$

In addition, if $u_{max}(i|\mathcal{E})$ (and thus $u_{min}(i|\mathcal{E})$) is $O(s(i|\mathcal{E})/n)$, for all $i : 0 \le i \le n$ (as is the case for random inversions, transpositions, and inverted transpositions), then $|\mathcal{F}_k - E[d_{BP}(G_0, G_k)]| = O(1)$.

Proof. The error is at most one half of the maximum difference between $\sum_{i=0}^{n} P^H_{i,k}$ and $\sum_{i=0}^{n} P^L_{i,k}$; the result follows from Lemma 3.

When both $u_{min}(i|\mathcal{E})$ and $u_{max}(i|\mathcal{E})$ are $O(\frac{s(i|\mathcal{E})}{n})$, the error is at most

$$\sum_{i=0}^{n} \frac{u_{max}(i|\mathcal{E}) - u_{min}(i|\mathcal{E})}{2\ s(i|\mathcal{E})} = \sum_{i=0}^{n} O\left(\frac{1}{n}\right) = O(1). \qquad \square$$

THEOREM 7. *Let $s_l = \min_{0\le i\le n}\{s(i|\mathcal{E})\}$, $s_h = \max_{0\le i\le n}\{s(i|\mathcal{E})\}$, $r_l = \min_{0\le i\le n}\{s(i|\mathcal{E}) + u_{min}(i|\mathcal{E})\}$, and $r_h = \max_{0\le i\le n}\{s(i|\mathcal{E}) + u_{max}(i|\mathcal{E})\}$. For all $k \ge 1$,*

$$\frac{s_l r_l}{s_h r_h} \le \frac{\mathcal{F}_k}{E[d_{BP}(G_0, G_k)]} \le \frac{s_h r_h}{s_l r_l}.$$

In addition, if $s_h/s_l = 1 + \Theta(\frac{1}{n})$ and $u_{max}(i|\mathcal{E})$ (and thus $u_{min}(i|\mathcal{E})$) is $O(s(i|\mathcal{E})/n)$, for all $i : 0 \le i \le n$, then

$$\frac{\mathcal{F}_k}{E[d_{BP}(G_0, G_k)]} = 1 + O\left(\frac{1}{n}\right).$$

Proof. We prove only the upper bound, as the lower bound is the reciprocal of the upper bound and can be proved similarly. Let $w = 1 - r_l$ and $v = 1 - r_h$; we have $v \le w$, $1 - w^k \le 1 - v^k$, and

$$\frac{\mathcal{F}_k}{E[d_{BP}(G_0, G_k)]} = \frac{\sum_{i=0}^{n} P_{i,k}^H}{\sum_{i=0}^{n} P_{i,k}^L} \le \frac{\max\limits_{0 \le i \le n} P_{i,k}^H}{\min\limits_{0 \le i \le n} P_{i,k}^L} \le \frac{s_h(1 + w + w^2 + \cdots + w^{k-1})}{s_l(1 + v + v^2 + \cdots + v^{k-1})}$$

$$= \frac{s_h \dfrac{1 - w^k}{1 - w}}{s_l \dfrac{1 - v^k}{1 - v}} = \left(\frac{s_h(1 - v)}{s_l(1 - w)}\right)\left(\frac{1 - w^k}{1 - v^k}\right) \le \frac{s_h(1 - v)}{s_l(1 - w)} = \frac{s_h r_h}{s_l r_l}. \qquad \square$$

In Table 1 is a list of the parameters of the three rearrangement classes in the GNT model.

5.6. Upper and lower bounds estimation with multiple rearrangement classes. We can easily extend the results to a mixture of different rearrangement classes. Consider m classes of rearrangements, $\mathcal{E}_1, \ldots, \mathcal{E}_m$, where $\mathcal{E}_i = (A(\mathcal{E}_i), f_{\mathcal{E}_i}), 1 \le i \le m$. For any rearrangement ρ, let $\gamma_j = \Pr(\rho \in \mathcal{E}_j), 1 \le j \le m$. Assume γ_j does not depend on chromosome G, the chromosome currently acted upon. Let $s(i|\mathcal{E}_j)$, $u(i|G, \mathcal{E}_j)$, $u_{min}(i|\mathcal{E}_j)$, and $u_{max}(i|\mathcal{E}_j)$ be the parameters corresponding to $\mathcal{E}_j$ as defined in Theorem 5. Let $\mathcal{E} = (A(\mathcal{E}), f_{\mathcal{E}})$ be the rearrangement class such that $A(\mathcal{E}) = \cup_{j=1}^{m} A(\mathcal{E}_j)$, and $f_{\mathcal{E}}(r|G) = \sum_{j=1}^{m} \gamma_j \ f_{\mathcal{E}_j}(r|G)$. Then $\mathrm{Sep}(i|G, \mathcal{E}) = \cup_{j=1}^{m} \mathrm{Sep}(i|G, \mathcal{E}_j)$, and $\mathrm{Uni}(i|G, \mathcal{E}) = \cup_{j=1}^{m} \mathrm{Uni}(i|G, \mathcal{E}_j)$.

The hierarchical way of choosing rearrangements (first, choose rearrangement class, and then choose one rearrangement among others in the class chosen) during evolution allows two rearrangements in different rearrangement classes to produce the same results, while retaining the additivity of probability:

$$\begin{aligned}\Pr(\rho = \rho_0 | G = G_0) &= \sum_{j=1}^{m} \Pr(\rho = \rho_0 | G = G_0, \mathcal{E}_j \text{ is chosen}) \Pr(\mathcal{E}_j \text{ is chosen} | G = G_0) \\ &= \sum_{j=1}^{m} \gamma_j f_{\mathcal{E}_j}(\rho_0 | G_0).\end{aligned}$$

The new recurrence is

$$\begin{aligned}s(i|\mathcal{E}) = \Pr(B_i(G_{k+1}) = 1 | B_i(G_k) = 0) &= \Pr(\rho_k \in \mathrm{Sep}(i|G_k, \mathcal{E}) \mid B_i(G_k) = 0) \\ = \sum_{j=1}^{m} \Pr(\rho_k \in \mathrm{Sep}(i|G_k, \mathcal{E}_j) \mid B_i(G_k) = 0) &= \sum_{j=1}^{m} \gamma_j s(i|\mathcal{E}_j).\end{aligned}$$

Similarly,

$$\begin{aligned}u(i|G_k, \mathcal{E}) = \Pr(B_i(G_{k+1}) = 0 | B_i(G_k) = 1) &= \sum_{j=1}^{m} \gamma_j u_j(i|G_k, \mathcal{E}_j) \quad \forall k \ge 0, \\ u_{min}(i|\mathcal{E}) = \sum_{j=1}^{m} \gamma_j u_{min}(i|\mathcal{E}_j), \quad u_{max}(i|\mathcal{E}) &= \sum_{j=1}^{m} \gamma_j u_{max}(i|\mathcal{E}_j), \\ P_{i,k+1} &= (1 - P_{i,k}) s(i|\mathcal{E}) + P_{i,k}(1 - u(i|G_k, \mathcal{E})) \\ &= P_{i,k}(1 - s(i|\mathcal{E}) - u(i|G_k, \mathcal{E})) + s(i|\mathcal{E}), \\ P_{i|0} &= 0.\end{aligned}$$

TABLE 1

Recurrence parameters for rearrangement classes in the GNT model. The three rearrangement classes are inversion (Inv), transposition (Trp), and inverted transposition (ITrp). For circular chromosomes, $B_i(G_k)$ has the same distribution for $1 \leq i \leq n$, and $B_0(G_k)$ is always set to 0.

Linear chromosomes							
Signed	Rearrangement type	$s(i)$ $i \neq 0, n$	s_0, s_n	$u_{min}(i)$ $i \neq 0, n$	$u_{max}(i)$ $i \neq 0, n$	$u_{min}(0)$ $u_{min}(n)$	$u_{max}(0)$ $u_{max}(n)$
No	Inv	$\frac{n-2}{\binom{n}{2}}$	$\frac{2}{n}$	$\frac{2}{\binom{n}{2}}$	$\frac{2}{\binom{n}{2}}$	$\frac{1}{\binom{n}{2}}$	$\frac{1}{\binom{n}{2}}$
Yes	Inv	$\frac{2}{n+1}$	$\frac{2}{n+1}$	0	$\frac{1}{\binom{n+1}{2}}$	0	$\frac{1}{\binom{n+1}{2}}$
No	Trp	$\frac{3(n-2)}{n(n-1)}$	$\frac{3}{n+1}$	$\frac{6}{n(n-1)}$	$\frac{6}{n(n-1)}$	$\frac{1}{\binom{n+1}{3}}$	$\frac{6}{n(n+1)}$
Yes	Trp	$\frac{3}{n+1}$	$\frac{3}{n+1}$	0	$\frac{6}{n(n+1)}$	0	$\frac{6}{n(n+1)}$
No	ITrp	$\frac{3(n-3)}{n(n-1)}$	$\frac{3}{n}$	$\frac{6}{n(n-2)}$	$\frac{6}{n(n-2)}$	$\frac{1}{2\binom{n}{3}}$	$\frac{6}{n(n-1)}$
Yes	ITrp	$\frac{3}{n+1}$	$\frac{3}{n+1}$	0	$\frac{3}{(n-1)(n+1)}$	0	$\frac{3}{n(n+1)}$

Circular chromosomes				
Signed	Rearrangement type	$s(i)$ $1 \leq i \leq n$	$u_{min}(i)$ $1 \leq i \leq n$	$u_{max}(i)$ $1 \leq i \leq n$
No	Inv	$\frac{n-3}{\binom{n-1}{2}}$	$\frac{2}{\binom{n-1}{2}}$	$\frac{2}{\binom{n-1}{2}}$
Yes	Inv	$\frac{2}{n}$	0	$\frac{1}{\binom{n}{2}}$
No	Trp	$\frac{3(n-3)}{(n-1)(n-2)}$	$\frac{6}{(n-1)(n-2)}$	$\frac{6}{(n-1)(n-2)}$
Yes	Trp	$\frac{3}{n}$	0	$\frac{6}{n(n-1)}$
No	ITrp	$\frac{3}{n-1}$	$\frac{6}{(n-1)(n-3)}$	$\frac{6}{(n-1)(n-3)}$
Yes	ITrp	$\frac{3}{n}$	0	$\frac{4}{n(n-2)}$

Results similar to Theorems 6 and 7 on error bounds can be obtained for multiple classes. Recall that we defined $\mathcal{F}_k = \sum_{i=0}^{n} \frac{P_{i,k}^L + P_{i,k}^H}{2}$.

THEOREM 8. *Consider the estimator $\mathcal{F}_k$ with the parameters $s(i|\mathcal{E})$, $u_{min}(i|\mathcal{E})$, and $u_{max}(i|\mathcal{E})$ given in the previous paragraphs. If the assumptions in Theorems 6 and 7 regarding these parameters are satisfied, then*

$$|\mathcal{F}_k - E[d_{BP}(G_0, G_k)]| = O(1)$$

and

$$\phi^{-1} \leq \frac{\mathcal{F}_k}{E[d_{BP}(G_0, G_k)]} \leq \phi,$$

where $\phi = 1 + O(\frac{1}{n})$.

Proof. The proof follows from Theorems 6 and 7. □

5.7. Approximating the breakpoint distance under the GNT model. We now show how to compute an approximation to the expected breakpoint distance under the $GNT(w_I, w_T, w_{IT})$ model. We use the notation defined in earlier sections (i.e., $\mathcal{F}_k$,$s(i)$, $u(i)$, $u_{min}(i)$, and $u_{max}(i)$) and provide the parameters for the upper and lower bounds technique in Table 2.

Recall that $\mathcal{F}_k = \sum_{i=0}^{n} \frac{P_{i,k}^L + P_{i,k}^H}{2}$. Under the GNT model we can tighten the error bounds for $\mathcal{F}_k$ obtained in Theorem 6 as follows.

THEOREM 9. *We assume the chromosomes evolve under the GNT model. For all $k > 0$,*

$$|\mathcal{F}_k - E[d_{BP}(G_0, G_k)]| \leq 1 + \frac{1}{n-1}$$

and

$$\phi^{-1} \leq \frac{\mathcal{F}_k}{E[d_{BP}(G_0, G_k)]} \leq \phi,$$

where $\phi = 1 + \frac{2+4w_T+2w_{IT}}{2+w_T+w_{IT}} n^{-1} + O(n^{-2})$.

Proof. The proof follows from Theorems 6 and 7 with parameters $s(i)$, $u_{min}(i)$, and $u_{max}(i)$. See Table 2 for details. The relative error bound can be obtained by examining $\frac{\sum_{i=0}^{n} P_{i,k}^H}{\sum_{i=1}^{n} P_{i,k}^L}$ directly. □

6. Estimating evolutionary distances under the GNT model. In the previous two sections we presented methods (one exact and one approximate) for estimating the expected breakpoint distance produced by a sequence of random events under the GNT model. In this section we show how to use those methods to estimate evolutionary distances under the GNT model.

Recall the discussion in the overview section, in which we described the basic technique for estimating evolutionary distances. We assumed we had a method for estimating the expected breakpoint distance produced by a sequence of k random events in the GNT model (denoted by $E[d_{BP}(G, G_k)]$) and that we knew the parameters of the model (i.e., the triplet (w_I, w_T, w_{IT})). Our technique was as follows:

- Compute the breakpoint distance $y = d_{BP}(G, G')$ between G and G'.
- Return the integer k that minimizes $|E[d_{BP}(G, G_k)] - y|$.

TABLE 2

Recurrence parameters and error bounds for the GNT model. The probabilities that a rearrangement is an inversion, a transposition, or an inverted transposition are $1 - w_T - w_{IT}$, w_T, and w_{IT}, respectively, for linear (Lin) or circular (Cir) chromosomes. For circular chromosomes, the parameters $s(n)$, $u_{min}(n)$, and $u_{max}(n)$ agree with those for $i = 1, \ldots, n-1$, and $B_0(G) = 0$ for all chromosomes G.

Signed	Chr.	$s(i)$ $(1 \le i \le n-1)$	$u_{min}(i)$ $(1 \le i \le n-1)$	$u_{max}(i)$ $(1 \le i \le n-1)$
No	Lin	$\frac{2(n-2)+w_T(n-2)+w_{IT}(n-5)}{n(n-1)}$	$\frac{2(2n-4)+2w_T(n-2)+2w_{IT}(n+1)}{n(n-1)(n-2)}$	$\frac{2(2n-4)+2w_T(n-2)+2w_{IT}(n+1)}{n(n-1)(n-2)}$
Yes	Lin	$\frac{2+w_T+w_{IT}}{n+1}$	0	$\frac{2(n-1)+4w_T(n-1)+w_{IT}(n+2)}{(n+1)n(n-1)}$
No	Cir	$\frac{2(n-3)+w_T(n-3)+w_{IT}n}{(n-1)(n-2)}$	$\frac{4(n-3)+2w_T(n-3)+2w_{IT}n}{(n-1)(n-2)(n-3)}$	$\frac{4(n-3)+2w_T(n-3)+2w_{IT}n}{(n-1)(n-2)(n-3)}$
Yes	Cir	$\frac{2+w_T+w_{IT}}{n}$	0	$\frac{2(n-2)+4w_T(n-2)+2w_{IT}n}{n(n-1)(n-2)}$

Signed	Chr.	$s(0), s(n)$	$u_{min}(0), u_{min}(n)$	$u_{max}(n), u_{max}(n)$
No	Lin	$\frac{2(n+1)+w_T(n-2)+w_{IT}(n+1)}{n(n+1)}$	$\frac{2(n+1)(n-2)-2w_T(n-2)^2-w_{IT}(n+1)(2n-7)}{(n+1)n(n-1)(n-2)}$	$\frac{2(n+1)+4w_T(n-2)+4w_{IT}(n+1)}{(n+1)n(n-1)}$
Yes	Lin	$\frac{2+w_T+w_{IT}}{n+1}$	0	$\frac{2+4w_T+w_{IT}}{n(n+1)}$

Sign	Chr.	Absolute error bound $= \sum_{i=0}^{n} \frac{u_{max}(i)-u_{min}(i)}{2s(i)}$	Relative error upper bound† $= \frac{\sum_{i=0}^{n} P^H_{i,k}}{\sum_{i=0}^{n} P^L_{i,k}}$
No	Lin	$\frac{6w_T+w_{IT}\left(2+\frac{9(n-1)}{(n-2)^2}\right)}{(n-1)\left(2+w_T+w_{IT}+\frac{3(1+w_{IT})}{n-2}\right)} \le \frac{2}{n-1}$	$1 + O(n^{-2})$
Yes	Lin	$\frac{1}{2} + \frac{3w_T}{2(2+w_T+w_{IT})} + \frac{1}{n}\left(2 - \frac{3}{2+w_T+w_{IT}}\right) \le 1 + \frac{1}{n}$	$1 + \frac{2+4w_T+w_{IT}}{2+w_T+w_{IT}} n^{-1} + O(n^{-2})$
No	Cir	0	1
Yes	Cir	$\left(1 + \frac{3w_T+w_{IT}\left(\frac{n+2}{n-2}\right)}{2+w_T+w_{IT}}\right)\left(\frac{n}{2(n-1)}\right) \le 1 + \frac{1}{n-1}$	$1 + \frac{2+4w_T+2w_{IT}}{2+w_T+w_{IT}} n^{-1} + O(n^{-2})$

† See Theorems 7 and 9 for details. Only the upper bounds are shown here; the lower bounds are their reciprocals.

In the previous two sections we presented methods (one exact and one approximate) for estimating the expected breakpoint distance produced by k random events under the $GNT(w_I, w_T, w_{IT})$ model. Thus, each of these methods can be used in this general technique to estimate evolutionary distances under the GNT model. Since each involves inverting an expected breakpoint distance, we call these methods `Exact-IEBP` and `Approx-IEBP`. However, in order to precisely define these methods and compute their running time, we need to answer the following questions:

- For which values of k will we compute $E[d_{BP}(G, G_k)]$ (the expected breakpoint distance produced by k random events under the GNT model)?
- Given a set of these expected breakpoint distances, how do we find k minimizing $|E[d_{BP}(G, G_k)] - y|$?

We present the answers to these questions in this section. These answers will then let us prove our main theorem (Theorem 12), which shows that we can compute the pairwise `Exact-IEBP` distances of m circular chromosomes in $O(m^2n + \min\{m^2, n\} \log n + n^3 \log n)$ time, and $O(m^2n + \min\{m^2, n\} \log n + n^5 \log n)$ time if the chromosomes are linear, and we can compute the `Approx-IEBP` distances of all $\binom{m}{2}$ pairs of chromosomes in $O(m^2n + \min\{m^2, n\} \log n$ time for both linear and circular chromosomes. Thus, the `Approx-IEBP` method is much faster than the `Exact-IEBP` method for typical values of n and m that would be encountered in practice (where $n \gg m$ is likely). Since we will also establish that `Approx-IEBP` provides highly accurate estimates of evolutionary distances, and that phylogenies obtained using `Approx-IEBP` are comparable to phylogenies obtained using `Exact-IEBP`, `Approx-IEBP` is a competitive technique.

6.1. Fast mixing of the GNT model. Both the `Exact-IEBP` and the `Approx-IEBP` algorithms require a finite upper limit to the number of actual rearrangements allowed to occur on the chromosome. However, distance-based phylogeny reconstruction methods require finite entries in distance matrices, and so we need some upper bound on the estimated true evolutionary distance. This means we need only to compute $\mathcal{F}_i$ for all i up to the smallest k such that $E[d_{BP}(G_0, G_\infty)] - \mathcal{F}_k = o(1)$. Since the approximate expected breakpoint distance $\mathcal{F}_k$ in `Approx-IEBP` has absolute error $O(1)$, the upper limit for `Approx-IEBP` should be sufficient for `Exact-IEBP` as well.

Intuitively, the following question provides us with a reasonable upper limit: what is the number of rearrangements required for the resulting chromosome to be almost random? The answer was shown, for the inversion-only scenario, to be $\Theta(n \ln n)$ in [7].

THEOREM 10 (see [7]). *Assume we apply k random inversions to the linear unrearranged chromosome with n genes, $(1, 2, \ldots, n)$. Let P_k be the distribution of linear gene orders after k random inversions, and let $J_n = 2^n n!$ be the number of distinct linear gene orders with n genes. Let $D_k = \max_G |P_k(G) - \frac{1}{J_n}|$ be the total variational distance between P_k and the uniform distribution of gene orders. Then*

1. *if $c < 1/2$, then $\lim_{n\to\infty} D_{cn \ln n} = 1$;*
2. *if $c > 2$, then $\lim_{n\to\infty} D_{cn \ln n} = 0$.*

The result above is applicable only to the inversion-only scenario. Using results in the derivation for `Approx-IEBP`, we show in Theorem 11 that the upper limit our algorithms need is smaller than $2n \ln n$: when $k = \frac{1}{2+w_T+w_{IT}} n \ln n$, the expected breakpoint distance is already "saturated," in the sense the expected breakpoint distance reaches its maximum value.

We assume the chromosomes are signed and circular, and we fix (w_I, w_T, w_{IT}); then s, u_{min}, u_{max}, P_k^L, and P_k^H all are independent of the position of breakpoints. We set $\mathcal{F}_\infty = \lim_{k\to\infty} \mathcal{F}_k$. (More generally, given a definition for X_k, we set $X_\infty =$

$\lim_{k\to\infty} X_k$.)

LEMMA 4.

1. $\mathcal{F}_\infty = n(1 - \frac{u_{max}}{2(s+u_{max})})$.
2. *Let* $c = \frac{v}{2+w_T+w_{IT}} \ln n$ $(v > 1)$. *Then* $P^H_{cn} \geq 1 - \frac{1}{n^v}$.
3. *Let* $c = \frac{v}{3+w_T+w_{IT}} \ln n$ $(v > 1)$. *Then* $P^L_{cn} \geq (1 - \frac{u_{max}}{s+u_{max}})(1 - \frac{1}{n})$.

Proof.

1. We have

$$\begin{aligned} s &= \frac{2 + w_T + w_{IT}}{n}, \\ u_{min} &= 0, \\ u_{max} &= \frac{2(n-2) + 4w_T(n-2) + 2w_{IT}n}{n(n-1)(n-2)}. \end{aligned}$$

Since

$$\begin{aligned} P^H_k &= s\frac{1 - (1 - s - u_{min})^k}{1 - (1 - s - u_{min})}, \\ P^L_k &= s\frac{1 - (1 - s - u_{max})^k}{1 - (1 - s - u_{max})}. \end{aligned}$$

We have

$$\begin{aligned} P^H_\infty &= s\frac{1}{1 - (1 - s - u_{min})} = \frac{s}{s + u_{min}} = 1, \\ P^L_\infty &= s\frac{1}{1 - (1 - s - u_{max})} = \frac{s}{s + u_{max}} \\ \Rightarrow \mathcal{F}_\infty &= \frac{n}{2}\left(P^H_\infty + P^L_\infty\right) \\ &= \frac{n}{2}\left(\frac{2s + u_{max}}{s + u_{max}}\right) \\ &= n\left(1 - \frac{u_{max}}{2(s + u_{max})}\right). \end{aligned}$$

2.

$$\begin{aligned} P^H_{cn} &= 1 - (1 - s)^{cn} \\ &\geq 1 - \exp(-c(2 + w_T + w_{IT})) \\ &= 1 - \frac{1}{n^v}. \end{aligned}$$

3. We have

$$\begin{aligned} u_{max} &= \frac{2(n-2) + 4w_T(n-2) + 2w_{IT}n}{n(n-1)(n-2)} \leq \frac{2 + 4w_T + 2w_{IT}}{n(n-2)} \leq \frac{6}{n(n-2)}, \\ s + u_{max} &= \frac{2(n-2) + 4w_T(n-2) + 2w_{IT}n}{n(n-1)(n-2)} + \frac{2 + w_T + w_{IT}}{n} \leq \frac{3 + w_T + w_{IT}}{n}. \end{aligned}$$

So

$$(1 - (s + u_{max}))^{cn} \leq \exp(-cn(s + u_{max})) \leq \exp(-c(3 + w_T + w_{IT})).$$

The lemma follows:

$$\begin{aligned}
P_{cn}^{L} &= \frac{s}{s+u_{max}}\left(1-(1-s-u_{max})^{cn}\right)\\
&\geq \frac{s}{s+u_{max}}\left(1-\exp(-cn(s+u_{max}))\right)\\
&\geq \left(1-\frac{u_{max}}{s+u_{max}}\right)\left(1-\exp(-cn(s+u_{max}))\right)\\
&\geq \left(1-\frac{u_{max}}{s+u_{max}}\right)\left(1-\exp(-c(3+w_T+w_{IT}))\right)\\
&= \left(1-\frac{u_{max}}{s+u_{max}}\right)\left(1-\frac{1}{n^v}\right). \qquad \square
\end{aligned}$$

THEOREM 11. *Let* $k = \frac{vn}{2+w_T+w_{IT}}\ln n$ *(*$v > 1$*). Then* $\mathcal{F}_\infty - \mathcal{F}_{cn} \leq n^{1-v}$.

Proof.

$$\begin{aligned}
\mathcal{F}_k &= \frac{n}{2}(P_k^L + P_k^H)\\
&\geq \frac{n}{2}\left(1-\frac{1}{n^v}+\left(1-\frac{u_{max}}{s+u_{max}}\right)\left(1-\frac{1}{n^v}\right)\right)\\
&= \frac{n}{2}\left(1-\frac{1}{n^v}\right)\left(2-\frac{u_{max}}{s+u_{max}}\right)\\
&= \left(n-\frac{1}{n^{v-1}}\right)\left(1-\frac{u_{max}}{2(s+u_{max})}\right)\\
\Rightarrow \mathcal{F}_\infty - \mathcal{F}_k &\leq \frac{1}{n^{v-1}}\left(1-\frac{u_{max}}{2(s+u_{max})}\right) \leq n^{1-v}. \qquad \square
\end{aligned}$$

6.2. Running time analysis. We prove the following result regarding the running time.

THEOREM 12. *Let* S *be a set of* m *chromosomes, each on the same set of* n *genes, with each gene appearing once in each chromosome. If each chromosome in* S *is circular, then we can compute the* `Exact-IEBP` *distance matrix in* $O(m^2n + \min\{m^2, n\}\log n + n^3\log n)$ *time; if the chromosomes in* S *are each linear, then the* `Exact-IEBP` *distance matrix can be computed in* $O(m^2n+\min\{m^2, n\}\log n+n^5\log n)$ *time. For both linear and circular chromosomes, we can compute the* `Approx-IEBP` *distance matrix in* $O(m^2n + \min\{m^2, n\}\log n)$ *time.*

Proof. As shown in Theorem 11, we can assume that no more than $r = vn\ln n$ rearrangements occur in the evolutionary history. We will use this as an upper limit for the maximum number of rearrangement events, in order to obtain a faster algorithm for `Exact-IEBP` and `Approx-IEBP`.

We begin by deriving the running time for `Approx-IEBP`. For each pair of chromosomes we compute the breakpoint distance; this takes $O(m^2n)$ time. For each of the $\binom{m}{2}$ breakpoint distances that appears (of which there are at most $O(\min\{m^2, n\})$ distinct ones) we will need to compute the `Approx-IEBP` distance; this, we will show, can be done in $O(\min\{m^2, n\}\log n)$ time, yielding the desired result. Recall that $\mathcal{F}_k$ is the approximation of the expected breakpoint distance produced by k random events in the GNT model. For each k, the value k that minimizes $|\mathcal{F}_k - d_{BP}(G, G_k)|$ can be found in $O(\log r) = O(\log n)$ time using the binary search method; whenever a new $\mathcal{F}_k$ is required in the binary search, we compute it, which takes constant time.

To compute the `Exact-IEBP` distance matrix, we begin as with `Approx-IEBP`, by computing all pairwise breakpoint distances. We then compute the table of expected breakpoint distances after k events for every k between 1 and r. To do this, we compute Mx, $M^2x = M(Mx)$, up to M^rx, where M is the transition matrix and x is the distribution vector for G_0, the unrearranged chromosome. Then, for $k = 1$ to r, we compute $E[d_{BP}(G_0, G_k)]$ from M^kx in $O(1)$ time for each k. So the running time for building the table is dominated by r matrix-vector multiplications, which takes $O(rn^2)$ time for circular chromosomes and $O((r+n)n^4) = O(n^5 \log n)$ time for linear chromosomes. The table allows us to perform binary search in $O(\log n)$ time for each distinct breakpoint distance. □

7. Experiments. We studied four distance estimators with respect to estimating the evolutionary distances and with respect to their impact on phylogeny reconstruction. The four distance estimators we studied are

1. `BP`, the breakpoint distance between two chromosomes,
2. `INV`, the minimum number of inversions needed to transform one chromosome into another,
3. `Approx-IEBP`, and
4. `Exact-IEBP`.

7.1. Software. We use PAUP* 4.0 [27] to compute the neighbor joining (NJ) method and the topological error rate of the computed trees. We implemented a simulator [6] for the GNT model. The input consists of a rooted leaf-labeled tree and the associated parameters (i.e., edge lengths and the relative probabilities of inversions, transpositions, and inverted transpositions). On each edge, the simulator applies random rearrangement events to the circular chromosome at the ancestral node according to the model with given parameters w_T and w_{IT}. We use `tgen` [11] to generate random trees. These trees have topologies drawn from the uniform distribution, and edge lengths drawn from the discrete uniform distribution on intervals $[a, b]$, where we specify a and b.

7.2. Accuracy of the estimators. In this section we study the behavior of the `Exact-IEBP` and `Approx-IEBP` distances by comparing each to the actual number of rearrangement events. We simulate the GNT model on a circular chromosome with 37 genes (the typical number of genes in the animal mitochondrial chromosomes [4]) and 120 genes (the typical number of genes in the plant chloroplast chromosomes [13]). Starting with the unrearranged chromosome G_0, we apply k events to it to obtain the chromosome G_k for $k = 1, \ldots, 300$ when the number of genes is 120 and $k = 1, \ldots, 100$ when the number of genes is 37. For each value of k we simulate 500 runs. We then compute the four distances.

The simulation results are shown in Figures 3 and 4 for inversion-only evolution and Figure 5 when all three types of events are equiprobable. Note that both `BP` and `INV` distances underestimate the actual number of events (except for very small numbers of events) under all conditions. Note also that the error increases more rapidly with the number of events, when the model allows transpositions and inverted transpositions. When the number of genes increases, the standard deviations of both methods decrease. By contrast, `Approx-IEBP` and `Exact-IEBP` provide much better estimates of the true evolutionary distances.

We then compute the absolute error (that is, the difference between the measured distance and the actual number of events) for each of the distance estimators. Using

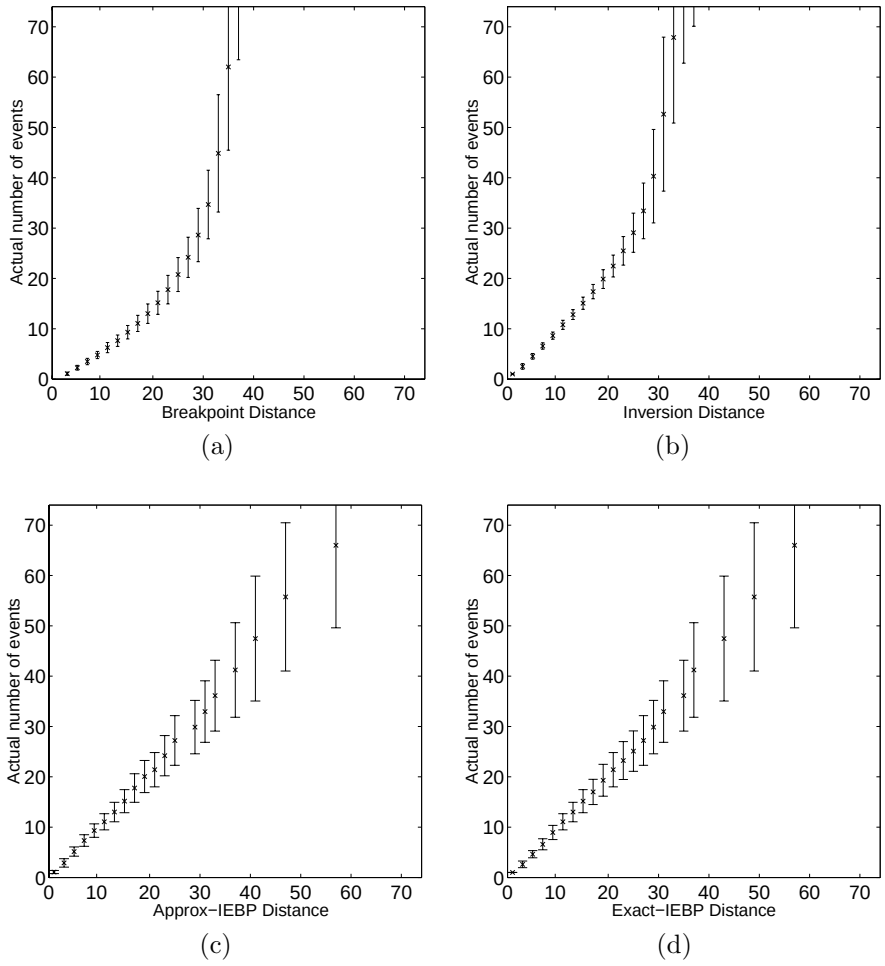


FIG. 3. *Accuracy of the estimators (see section* 7.2*). The number of genes is* 37 *(the typical value for mitochondrial chromosomes). Each plot is a comparison between some distance measures and the actual number of rearrangements. The evolutionary model is inversion-only. The x-axis is divided into* 25 *bins; the length of the vertical bars indicate the standard deviation. The distance estimators are* (a) *BP,* (b) *INV,* (c) *Approx-IEBP, and* (d) *Exact-IEBP.*

the same data in the previous experiment, we generate the plots as follows. The x-axis is the actual number of events. For each distance estimator D we plot the curve f_D, where $f_D(x)$ is the mean of the set $\{|\frac{1}{c}D(G_0, G_k) - k| : 1 \leq k \leq x\}$ over all observations G_k, where we pick a constant c for each distance estimator in order to reduce the bias. Thus, for example, we use $c = 1$ for the `Approx-IEBP` and the `Exact-IEBP` distances since they estimate the actual number of events. For the `BP` distance we let $c = 2(1 - w_T - w_{IT}) + 3(w_T + w_{IT}) = 2 + w_T + w_{IT}$ since this is the expected number of breakpoints created by each event in the model when the number of events is very low. Similarly for the `INV` distance we let $c = (1 - w_T - w_{IT}) + 3w_T + 2w_{IT} = 1 + 2w_T + w_{IT}$ since each transposition can be replaced by three inversions, and each inverted transposition can be replaced by two inversions.

The result is in Figure 6. We consistently observe that `Exact-IEBP` is the best, and that `INV` and `BP` are the worst, and the difference between `Exact-IEBP` and `BP` or `INV`

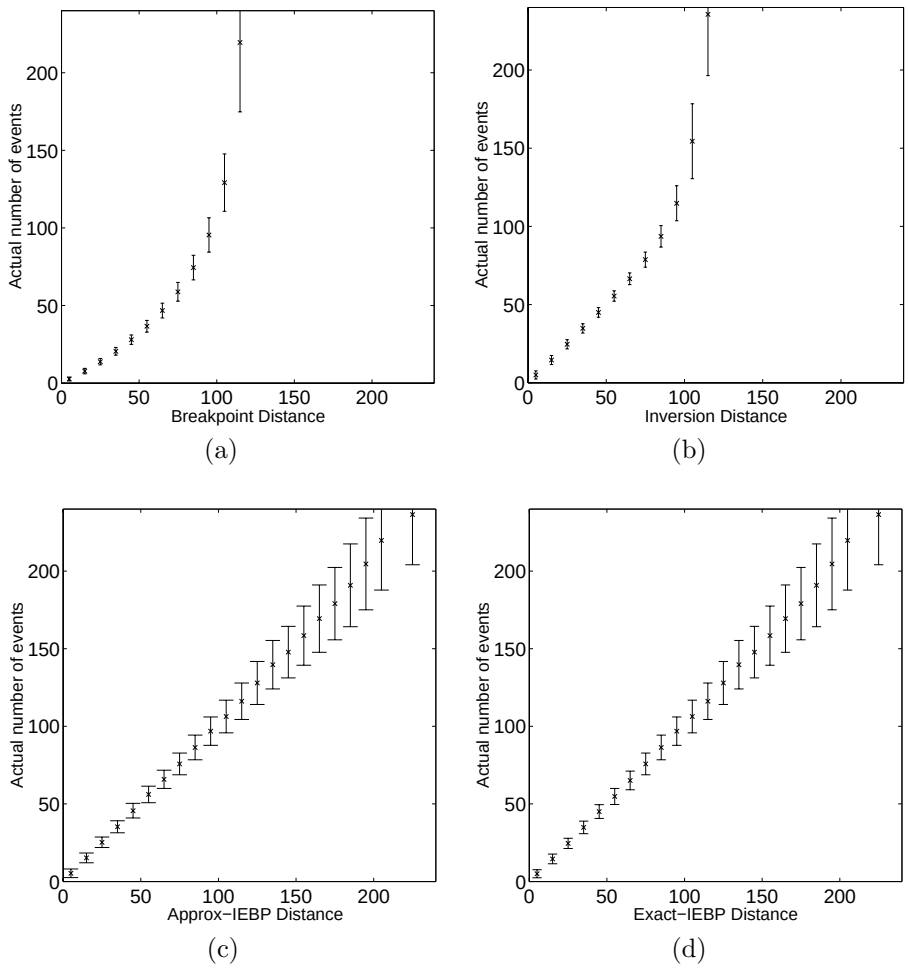


FIG. 4. *Accuracy of the estimators (see section* 7.2*). The number of genes is* 120 *(the typical value for chloroplast chromosomes). Each plot is a comparison between some distance measures and the actual number of rearrangements. The evolutionary model is inversion-only. The x-axis is divided into* 25 *bins; the length of the vertical bars indicate the standard deviation. The distance estimators are* (a) *BP,* (b) *INV,* (c) *Approx-IEBP, and* (d) *Exact-IEBP.*

is significant. The relative performance of `Approx-IEBP` is variable; in most cases it is a close second to `Exact-IEBP` (in most cases they are essentially indistinguishable), but in one case (transposition-only evolution on 37 genes) it is actually worse than all the methods for large numbers of events. Note also that all methods get worse with increasing numbers of events—accuracy is greatest for the smallest distances, with error increasing as the number of events increases.

7.3. Accuracy of NJ using different estimators. Based upon the relative performance of the distance estimators with respect to estimating true evolutionary distances, we would conjecture that phylogenies constructed using either `Approx-IEBP` or `Exact-IEBP` would be better than phylogenies constructed using either `BP` or `INV` and that, in general, phylogenies constructed using `Exact-IEBP` would be better than those constructed using `Approx-IEBP`. In this section we evaluate this conjecture. See Table 3 for the settings for the experiment.

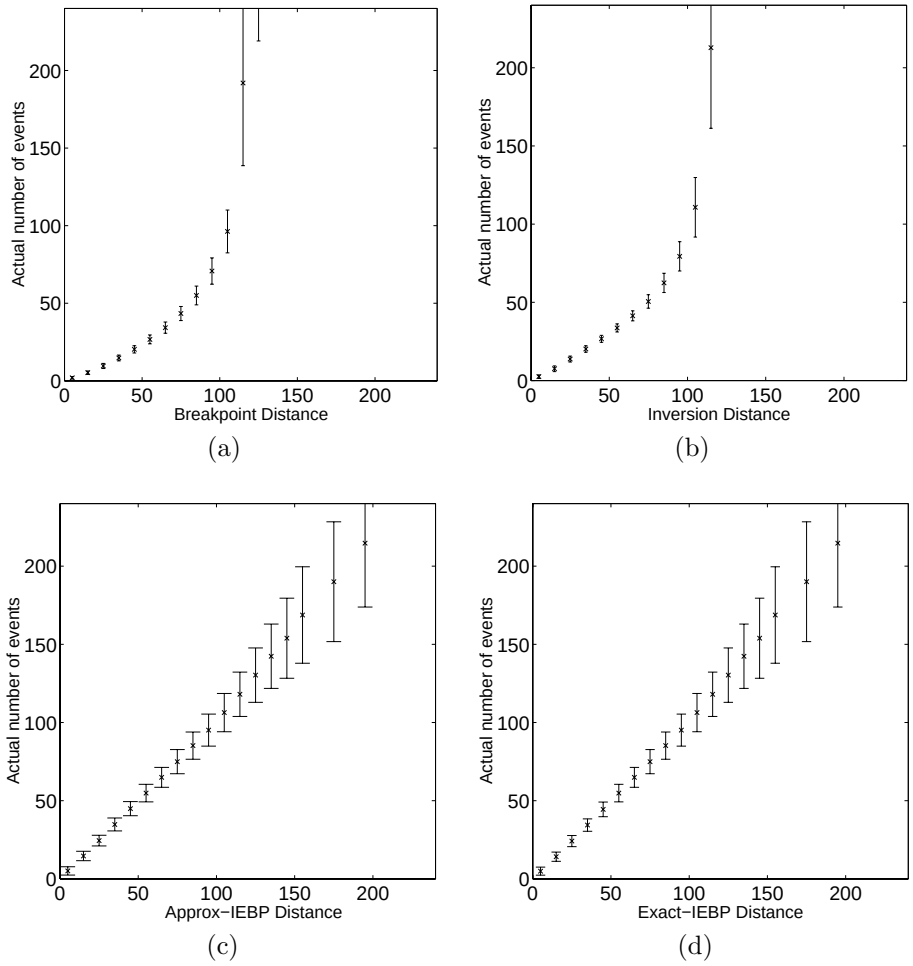


FIG. 5. *Accuracy of the estimators (see section* 7.2*). The number of genes is* 120 *(the typical value for chloroplast chromosomes). Each plot is a comparison between some distance measures and the actual number of rearrangements. The evolutionary model is such that the three types of rearrangement events are equiprobable ($w_I = w_T = w_{IT} = 1/3$). The x-axis is divided into* 25 *bins; the length of the vertical bars indicate the standard deviation. The distance estimators are* (a) `BP`, (b) `INV`, (c) `Approx-IEBP`, *and* (d) `Exact-IEBP`.

TABLE 3
Settings for the NJ performance simulation study.

Parameter	Value
1. Number of genes	37, 120
2. Number of leaves	160
3. Expected number of rearrangements in each edge	Discrete uniform within the following intervals: [1,3], [1,5], [1,10], [3,5], [3,10], and [5,10]
4. Probability settings: $(w_I, w_T, w_{IT})^\dagger$	(1,0,0) (Inversion only) $(0, 1, 0)$ (Transposition only) $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ (The three rearrangement classes are equally likely)
5. Datasets for each setting	100

† The probabilities that a rearrangement is an inversion, a transposition, or an inverted transposition are $1 - w_T - w_{IT}$, w_T, and w_{IT}, respectively.

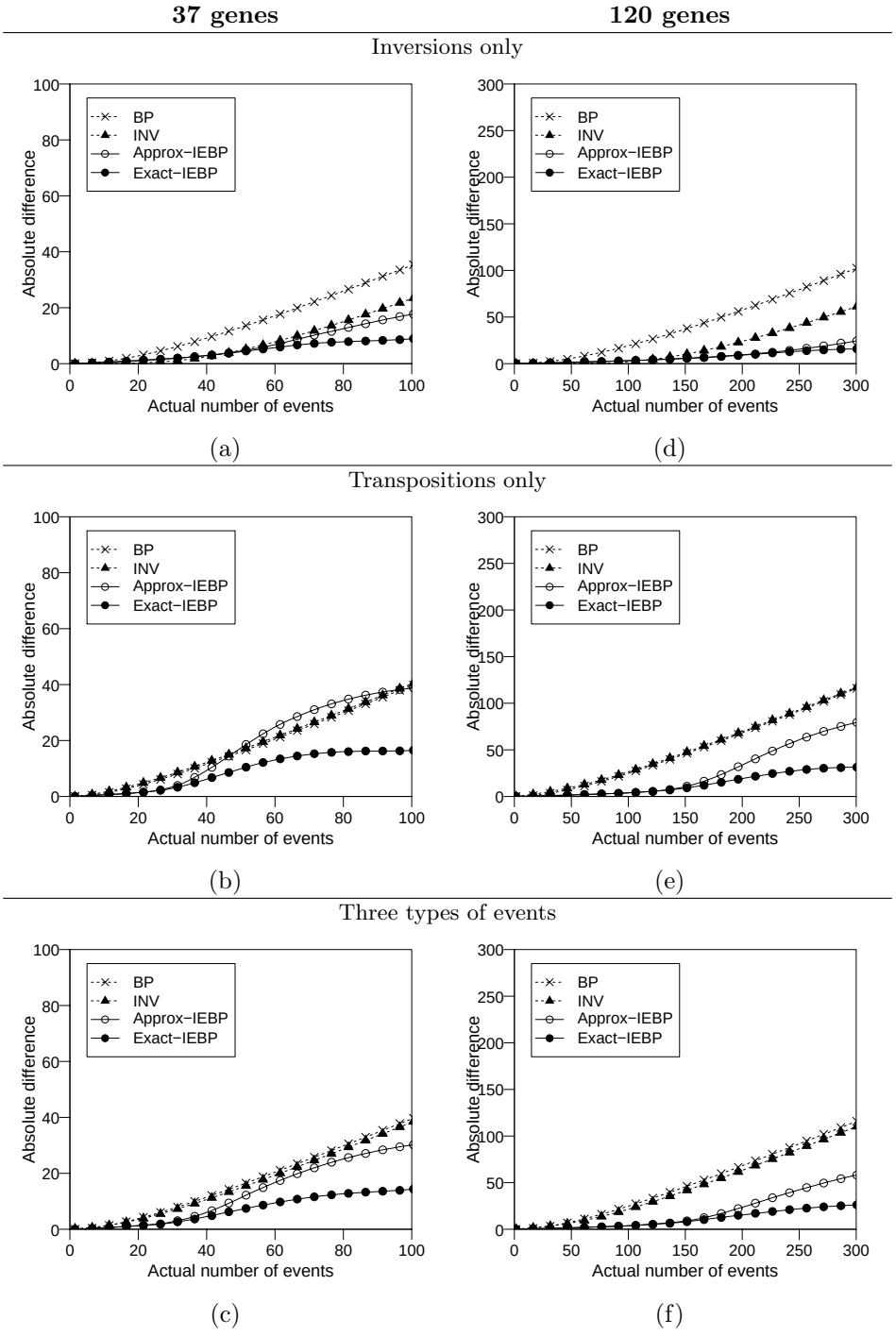


FIG. 6. *Accuracy of the estimators by absolute difference (see section* 7.2 *for the details). We simulate the evolution on* 37 *and* 120 *genes.*

We begin by defining the criteria by which we will evaluate the accuracy of the reconstructed trees.

Topological accuracy. During the evolutionary process, some edges of the model tree may have no changes (i.e., evolutionary events) on them. Since reconstructing such edges is at best guesswork, we are not interested in these edges. Hence, we define the "true tree" to be the tree that is obtained by *contracting* the edges in the model tree on which there are no changes [9, 16].

We now define how we score an inferred tree, by comparison to the true tree. Note that both trees are on the same set of leaves.

Let T be any tree on the set S of taxa, and let e be an edge in T. The removal of e (but not its endpoints) from T produces a bipartition π_e on the leaf set S. The set $C(T) = \{\pi_e : e \in E(T)\}$ is called the "character encoding" of T, and it uniquely identifies T (up to nodes of degree two). Now suppose that T_0 is the true tree and T_1 is the inferred tree. The *missing edges* are the edges in T_0 which do not correspond to bipartitions in $C(T_1)$; these are also referred to as *false negatives*. Note that the external edges (i.e., edges incident to a leaf) are trivial in the sense that they are present in every tree with the same set of leaves. Thus, the *false negative rate* is the number of false negatives, divided by the number of internal edges in T_0 (that set of internal edges is denoted by $E_I(T_0)$). In other words, the false negative rate is

$$FN(T_0, T_1) = \frac{|C(T_0) - C(T_1)|}{|E_I(T_0)|}.$$

Experimental setting. For each setting of the parameters (number of leaves, probabilities of rearrangements, and edge lengths), we generate 100 datasets of chromosomes as follows. First, we generate a random leaf-labeled tree (from the uniform distribution on topologies). The leaf-labeled tree and the parameter settings thus define a model tree in the GNT model. We run the simulator on the model tree and produce a set of chromosomes at the leaves.

For each set of chromosomes, we compute the four distances. We then compute NJ trees on each of the four distance matrices and compare the resultant trees to the true tree. The results of this experiment are in Figure 7. Distance matrices with some normalized edit distances equal to 1 are said to be "saturated"; it is well known that it is very difficult to obtain highly accurate phylogenies from datasets whose distance matrices are close to being saturated [12]. For this reason, we study the impact of the largest distance in the matrix on the accuracy of the phylogeny reconstructed for the data. Therefore, we bin the datasets according to their maximum distance, and we let the x-axis be the maximum normalized inversion distance (as computed by the linear time algorithm for minimum inversion distances given in [2]) between any two chromosomes in the input. The y-axis is the false negative rate (i.e., the proportion of missing edges) of the computed tree. False negative rates of less than 5% are excellent, but false negative rates of up to 10% can be tolerated.

We use NJ(D) to denote the tree returned by NJ using distance D; thus we have NJ(`BP`), NJ(`INV`), NJ(`Approx-IEBP`), and NJ(`Exact-IEBP`).

Observations. We begin with some general observations. First, under all conditions (number of genes and evolutionary model), for each of the distance estimator techniques, the topological error of the NJ trees increases with the diameter of the dataset; this is consistent with the observation made earlier that errors in estimating distances increase with the number of events. Second, independent largely of the evolutionary model and the distance estimator, trees reconstructed on 120 genes are

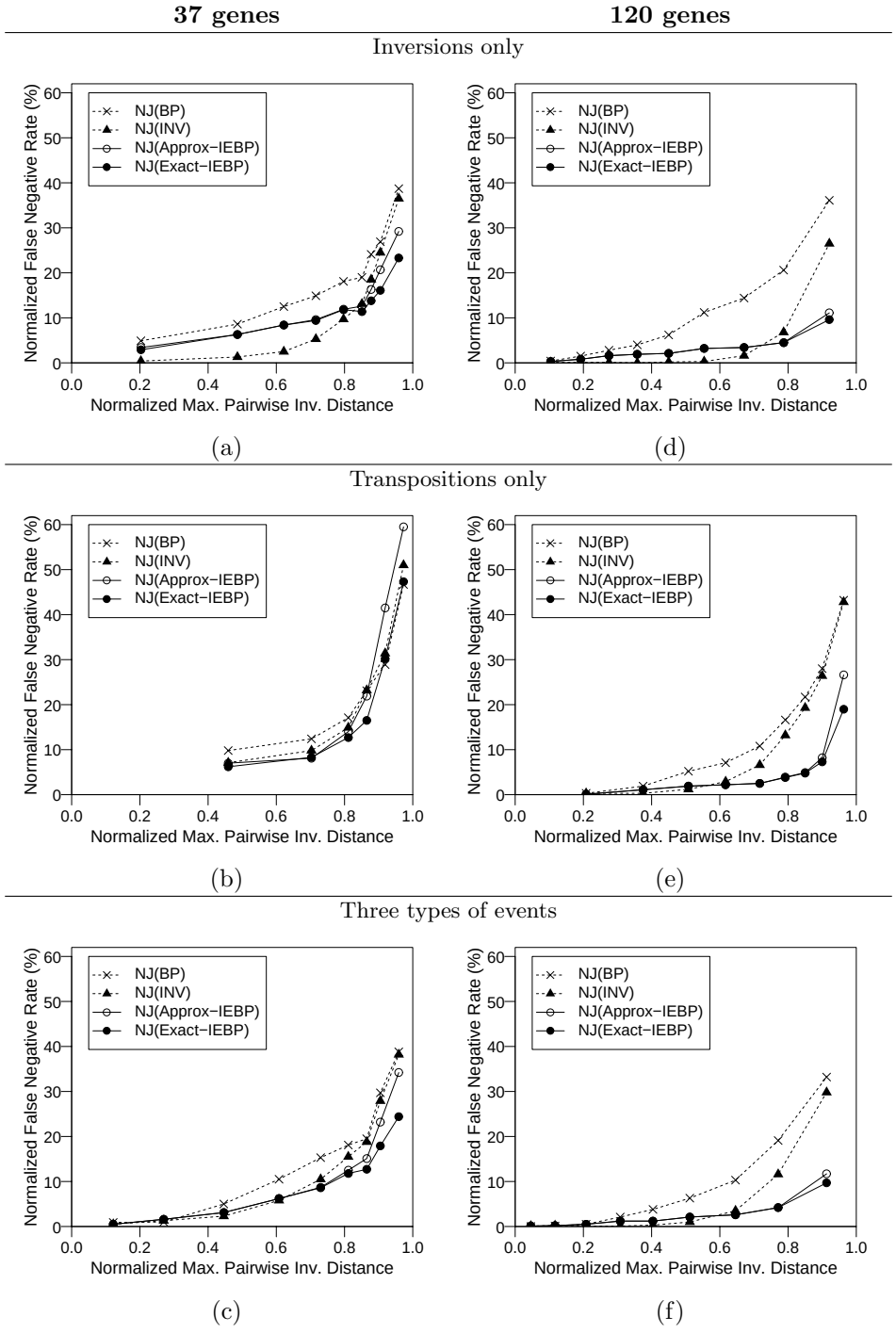


FIG. 7. *NJ performance under several distances (see section* 7.3*). See Table* 3 *for the settings in the experiment.*

more accurate than trees reconstructed on 37 genes, once the diameter is fixed (this observation fails only for high diameter datasets analyzed using BP). Also, NJ(BP) provides the worst trees, and (except for inversion-only evolution, as we will discuss below) NJ(Exact-IEBP) provides the best trees.

Trends specific to the number of genes, or the model of evolution, are as follows. First, inversion-only evolution is generally most accurately reconstructed using INV distances, rather than our new estimators; however, once the diameter of the dataset is large enough, our new estimators provide better phylogenies than INV. On datasets with 120 genes, there is very little difference between Exact-IEBP and Approx-IEBP, and both (except for very low diameter datasets) produce significantly better trees than BP or INV.

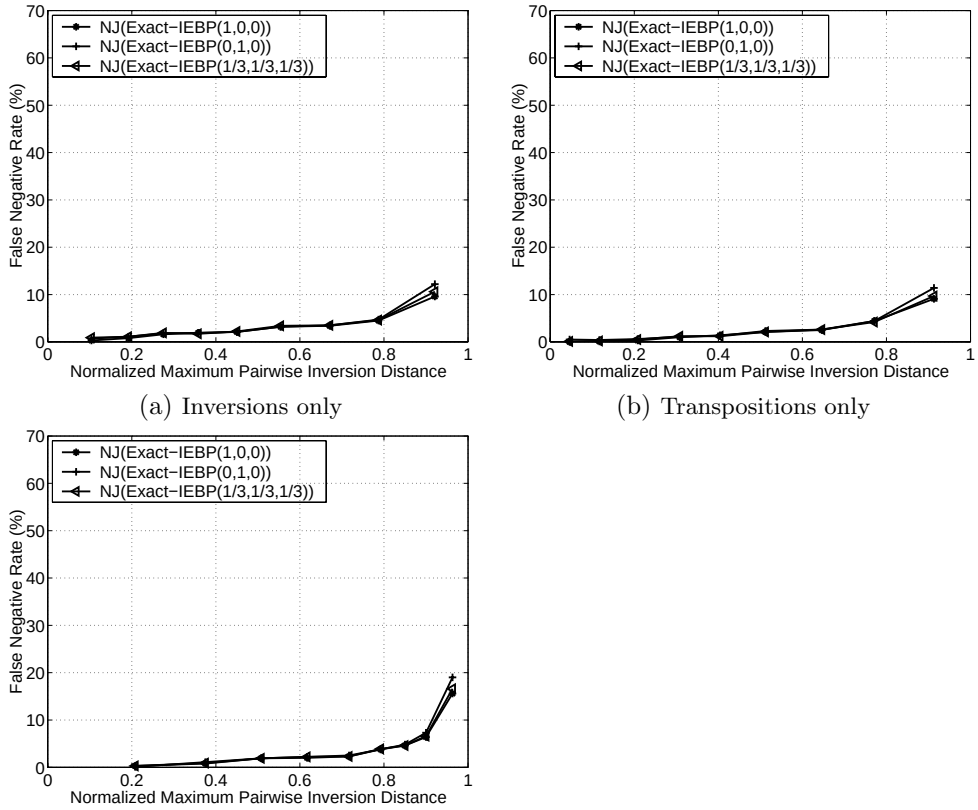
7.4. Robustness to unknown model parameters. In this section we consider the problem where the model parameters (w_I, w_T, w_{IT}) are unknown when using Exact-IEBP and Approx-IEBP, as will in general be the case in any real data analysis. Though it may be possible to estimate these parameters from the data, we will approach the problem here by assuming that the parameters are simply incorrectly specified: thus, the data evolve under one model (perhaps inversion-only), but distances are estimated under the assumption of another model (perhaps transposition-only). We will examine the consequences of these incorrect assumptions on the phylogenies reconstructed using these distances.

We explore this question for NJ(Exact-IEBP) in Figure 8; for the study of NJ(Approx-IEBP), see [28], which shows comparable robustness. The settings are given in Table 3. The experiment is similar to the previous experiment, except here we use both the correct and the incorrect values of (w_I, w_T, w_{IT}) for the Exact-IEBP distance. Note that all the constructed trees, whether using correct or highly incorrect model parameters, are indistinguishable with respect to topological error, showing that NJ(Exact-IEBP) is highly robust against errors in (w_I, w_T, w_{IT}).

8. Extension to other models. Approx-IEBP is applicable to rearrangement models that are more general than the GNT model, as long as the probability a rearrangement occurs depends on the rearrangement itself but not on the chromosome it acts upon.

For example, Pinter and Skiena proposed an inversion-only stochastic model of chromosomal evolution in [22] which assumes that the probability of an inversion depends only on its length (the number of genes being inverted) and that any two inversions with the same length are equiprobable. Our method can be used to estimate true evolutionary distances under this model, since this model is a mixture where we put all inversions of the same length into the same class.

9. Conclusion. We presented two polynomial time algorithms, Exact-IEBP and Approx-IEBP, for estimating the actual number of rearrangements that have taken place in the evolutionary history between two chromosomes that have evolved under the GNT model. Exact-IEBP uses exact estimates of the expected breakpoint distance under the GNT model, while Approx-IEBP uses approximate estimates but can be easily applied to more general models and is much faster. Thus, these techniques provide the only general purpose polynomial time methods for computing evolutionary distances between chromosomes that have evolved under inversions, transpositions, and inverted transpositions. We demonstrated the impact of these tools on phylogenetic reconstruction through the simulation study we presented, showing that many more



(a) Inversions only

(b) Transpositions only

(c) Three types of events equally likely

FIG. 8. *Robustness of the* `Exact-IEBP` *method to incorrectly specified parameters (see section* 7.4*). See Table* 3 *for the settings in the experiment. The three values in the legend are the* w_I*,* w_T*, and* w_{IT} *values used in the* `Exact-IEBP` *method, where* w_I*,* w_T*, and* w_{IT} *are the probabilities that a rearrangement event is an inversion, a transposition, or an inverted transposition.*

accurate trees can be reconstructed using these two estimators than using previously defined estimators, including the polynomial time inversion or breakpoint distances.

REFERENCES

[1] K. ATTESON, *The performance of the neighbor-joining methods of phylogenetic reconstruction*, Algorithmica, 25 (1999), pp. 251–278.

[2] D. A. BADER, B. M. E. MORET, AND M. YAN, *A linear-time algorithm for computing inversion distance between two signed permutations with an experimental study*, J. Comp. Biol., 8 (2001), pp. 251–278.

[3] M. BLANCHETTE, G. BOURQUE, AND D. SANKOFF, *Breakpoint phylogenies*, in Genome Informatics, S. Miyano and T. Takagi, eds., Universal Academy Press, Tokyo, Japan, 1997, pp. 25–34.

[4] M. BLANCHETTE, M. KUNISAWA, AND D. SANKOFF, *Gene order breakpoint evidence in animal mitochondrial phylogeny*, J. Mol. Evol., 49 (1999), pp. 193–203.

[5] A. CAPRARA AND G. LANCIA, *Experimental and statistical analysis of sorting by reversals*, in Comparative Genomics, D. Sankoff and J. H. Nadeau, eds., Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000, pp. 171–184.

[6] M. E. COSNER, R. K. JANSEN, B. M. E. MORET, L. A. RAUBESON, L.-S. WANG, T. WARNOW, AND S. WYMAN, *A new fast heuristic for computing the breakpoint phylogeny and a phylo-*

genetic analysis of a group of highly rearranged chloroplast genomes, in Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology (ISMB 2000), AAAI Press, Menlo Park, CA, 2000, pp. 104–115.

[7] R. Durrett, *Genome Rearrangement: Recent Progress and Open Problems*, 2003, http://www.math.cornell.edu/∼durrett/FGR/.

[8] P. L. Erdos, M. Steel, L. Szekely, and T. Warnow, *A few logs suffice to build almost all trees*—I. Random Structures Algorithms, 14 (1999), pp. 153–184.

[9] O. Gascuel, *private communication*, 2001.

[10] S. Hannenhalli and P. Pevzner, *Transforming cabbage into turnip (polynomial algorithm for genomic distance problems)*, in Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC95), 1995, pp. 178–189.

[11] D. Huson, *private communication*, 1999.

[12] D. Huson, S. Nettles, K. Rice, T. Warnow, and S. Yooseph, *Hybrid tree reconstruction methods*, ACM J. Experimental Algorithmics, 4 (1999), article 5.

[13] R. K. Jansen, *private communication*, 2000.

[14] H. Kaplan, R. Shamir, and R. E. Tarjan, *Faster and simpler algorithm for sorting signed permutations by reversals*, in Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA97), 1997, pp. 344–351.

[15] J. Kim and T. Warnow, *Tutorial on Phylogenetic Tree Estimation*, 1999, http://kim.bio.upenn.edu/∼jkim/media/ISMBtutorial.pdf.

[16] S. Kumar, *Minimum evolution trees*, Mol. Biol. Evol., 15 (1996), pp. 584–593.

[17] W.-H. Li, *Molecular Evolution*, Sinauer Associates, Sunderland, MA, 1997.

[18] J. H. Nadeau and B. A. Taylor, *Lengths of chromosome segments conserved since divergence of man and mouse*, Proc. Natl. Acad. Sci. U.S.A., 81 (1984), pp. 814–818.

[19] L. Nakhleh, B. M. E. Moret, U. Roshan, K. St. John, J. Sun, and T. Warnow, *The accuracy of fast phylogenetic methods for large datasets*, in Proceedings of the 7th Pacific Symposium on BioComputing (PSB02), 2002, pp. 211–222.

[20] R. G. Olmstead and J. D. Palmer, *Chloroplast DNA systematics: A review of methods and data analysis*, Amer. J. Bot., 81 (1994), pp. 1205–1224.

[21] J. D. Palmer, *Chloroplast and mitochondrial genome evolution in land plants*, in Cell Organelles, R. Herrmann, ed., Springer-Verlag, Wein, 1992, pp. 99–133.

[22] R. Y. Pinter and S. Skiena, *Genomic sorting with length-weighted reversals*, Genome Informatics, 13 (2002), pp. 103–111.

[23] L. A. Raubeson and R. K. Jansen, *Chloroplast DNA evidence on the ancient evolutionary split in vascular land plants*, Science, 255 (1992), pp. 1697–1699.

[24] A. Rokas and P. W. H. Holland, *Rare genomic changes as a tool for phylogenetics*, Trends in Ecology and Evolution, 15 (2000), pp. 454–459.

[25] N. Saitou and M. Nei, *The neighbor-joining method: A new method for reconstructing phylogenetic trees*, Mol. Biol. Evol., 4 (1987), pp. 406–425.

[26] D. Sankoff and M. Blanchette, *Probability models for genome rearrangements and linear invariants for phylogenetic inference*, in Proceedings of the 3rd International Conference on Computational Molecular Biology (RECOMB99), 1999, pp. 302–309.

[27] D. Swofford, *PAUP* 4.0*, Sinauer Associates, Sunderland, MA, 2001.

[28] L.-S. Wang and T. Warnow, *Estimating true evolutionary distances between genomes*, in Proceedings of the 33th Annual ACM Symposium on Theory of Computing (STOC 2001), 2001, pp. 637–646.

[29] M. Waterman, T. F. Smith, M. Singh, and W. A. Beyer, *Additive evolutionary trees*, J. Theoret. Biol., 64 (1977), pp. 199–213.

SIAM J. COMPUT.
Vol. 36, No. 1, pp. 132–157

FAST MONTE CARLO ALGORITHMS FOR MATRICES I: APPROXIMATING MATRIX MULTIPLICATION*

PETROS DRINEAS†, RAVI KANNAN‡, AND MICHAEL W. MAHONEY§

Abstract. Motivated by applications in which the data may be formulated as a matrix, we consider algorithms for several common linear algebra problems. These algorithms make more efficient use of computational resources, such as the computation time, random access memory (RAM), and the number of passes over the data, than do previously known algorithms for these problems. In this paper, we devise two algorithms for the matrix multiplication problem. Suppose A and B (which are $m \times n$ and $n \times p$, respectively) are the two input matrices. In our main algorithm, we perform c independent trials, where in each trial we randomly sample an element of $\{1, 2, \ldots, n\}$ with an appropriate probability distribution $\mathcal{P}$ on $\{1, 2, \ldots, n\}$. We form an $m \times c$ matrix C consisting of the sampled columns of A, each scaled appropriately, and we form a $c \times n$ matrix R using the corresponding rows of B, again scaled appropriately. The choice of $\mathcal{P}$ and the column and row scaling are crucial features of the algorithm. When these are chosen judiciously, we show that CR is a good approximation to AB. More precisely, we show that

$$\|AB - CR\|_F = O(\|A\|_F \|B\|_F / \sqrt{c}),$$

where $\|\cdot\|_F$ denotes the Frobenius norm, i.e., $\|A\|_F^2 = \sum_{i,j} A_{ij}^2$. This algorithm can be implemented without storing the matrices A and B in RAM, provided it can make two passes over the matrices stored in external memory and use $O(c(m+n+p))$ additional RAM to construct C and R. We then present a second matrix multiplication algorithm which is similar in spirit to our main algorithm. In addition, we present a model (the pass-efficient model) in which the efficiency of these and other approximate matrix algorithms may be studied and which we argue is well suited to many applications involving massive data sets. In this model, the scarce computational resources are the number of passes over the data and the additional space and time required by the algorithm. The input matrices may be presented in any order of the entries (and not just row or column order), as is the case in many applications where, e.g., the data has been written in by multiple agents. In addition, the input matrices may be presented in a sparse representation, where only the nonzero entries are written.

Key words. randomized algorithms, Monte Carlo methods, massive data sets, streaming models, matrix multiplication

AMS subject classification. 68W20

DOI. 10.1137/S0097539704442684

1. Introduction.

We are interested in developing and analyzing fast Monte Carlo algorithms for performing useful computations on large matrices. Examples of such computations include matrix multiplication, the computation of the singular value decomposition of a matrix, and the computation of compressed approximate decompositions of a matrix. In this paper, we present a computational model for computing on massive data sets (the pass-efficient model) in which our algorithms

*Received by the editors April 5, 2004; accepted for publication (in revised form) November 17, 2005; published electronically May 26, 2006. The technical report version of this paper appeared as *Fast Monte Carlo Algorithms for Matrices* I: *Approximating Matrix Multiplication*, by P. Drineas, R. Kannan, and M. W. Mahoney [13]. A preliminary version of this paper, including the main algorithm and main theorem of section 4, appeared as *Fast Monte-Carlo algorithms for approximate matrix multiplication*, by P. Drineas and R. Kannan, in Proceedings of the 42nd Annual Symposium on Foundations of Computer Science, 2001, pp. 452–459.

http://www.siam.org/journals/sicomp/36-1/44268.html

†Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180 (drinep@cs.rpi.edu).

‡Department of Computer Science, Yale University, New Haven, CT 06520 (kannan@cs.yale.edu).
This author was supported in part by a grant from the NSF.
§Department of Mathematics, Yale University, New Haven, CT 06520 (mahoney@cs.yale.edu).

may naturally be formulated; we also present two algorithms for the approximation of the product of two matrices. In a second paper we present two algorithms for the computation of low-rank approximations to a matrix [11]. Finally, in a third paper we present two algorithms to compute a compressed approximate decomposition to a matrix that has several appealing properties [12]. We expect our algorithms to be useful in many applications where data sets are modeled by matrices and are extremely large. For example, in information retrieval and data mining (two rapidly growing areas of research in computer science and scientific computation that build on techniques and theories from fields such as statistics, linear algebra, database theory, pattern recognition, and learning theory) a large collection of m objects, e.g., documents, genomes, images, or web pages, is implicitly presented as a set of points in an n-dimensional Euclidean space, where n is the number of features that describe the object. This collection may be represented by an $m \times n$ matrix A, the rows of which are the object vectors and the columns of which are the feature vectors.

Recent interest in computing with massive data sets has led to the development of computational models in which the usual notions of time efficiency and space efficiency have been modified [23, 19, 3, 14, 10, 5]. In the applications that motivate these data-streaming models [19, 5], e.g., the observational sciences and the monitoring and operation of large networked systems, the data sets are much too large to fit into main memory. Thus, they are either not stored or are stored in a secondary storage device which may be read sequentially as a data stream but for which random access is very expensive. Typically, algorithms that compute on a data stream examine the data stream, keep a small "sketch" of the data, and perform computations on the sketch. Thus, these algorithms are usually randomized and approximate, and their performance is evaluated by considering resources such as the time to process an item in the data stream, the number of passes over the data, the additional workspace and additional time required, and the quality of the approximations returned. (Note that in some cases the term "data-streaming model" refers to a model in which only a single pass over the data is allowed [19, 5].)

The motivation for our particular "pass-efficient" approach is that in modern computers the amount of external memory (e.g., disk storage or tape storage) has increased enormously, while RAM and computing speeds have increased, but at a substantially slower pace. Thus, we have the ability to store large amounts of data, but not in RAM, and we do not have the computational ability to process these data with algorithms that require superlinear time. A related motivation is that input-output rates have not increased proportionally. Thus, the size of the data inputs (as limited, e.g., by the size of disks) has increased substantially faster than the rate at which we can access the data randomly.

In order to provide a framework in which to view the algorithms presented herein, we first introduce and describe the pass-efficient model of data-streaming computation [10]. In the pass-efficient model the computational resources are the number of sequential-access passes over the data and the additional RAM space and the additional time required. Thus, our algorithms are quite different from traditional numerical analysis approaches and generally fit within the following framework. Our algorithms will be allowed to read the matrices from external storage a few—e.g., one or two or three—times and keep a small randomly chosen and rapidly computable "sketch" of the matrices in RAM. Our algorithms will also be permitted additional RAM space and additional time in order to perform computations on the "sketch." The results of these computations will be returned as approximations to the solution of the original problem.

In all of our algorithms, an important implementation issue will be how to form the random sample. An obvious choice is to use uniform sampling, where each data object is equally likely to be picked. Uniform sampling can be performed blindly, in which case the sample to be chosen can be decided before seeing the data. Even when the number of data elements is not known in advance an element can be selected uniformly at random in one pass over the data; see Lemma 1. Uniform sampling fits within our framework and is useful for certain (restricted) classes of problems. To obtain much more generality, we will sample according to a judiciously chosen (and data-dependent) set of nonuniform sampling probabilities. This nonuniform sampling, in which in the first pass through the data we compute sampling probabilities (e.g., we may keep rows or columns of a data matrix with probability proportional to the square of their lengths) and in the second pass we draw the sample, offers substantial gains. For example, it allows us to approximately solve problems in sparse matrices as well as dense matrices.

The idea of sampling rows or columns of matrices in order to approximate various operations is not new; indeed, a motivation for our main matrix multiplication algorithm came from [15]. In this paper and accompanying work [11, 12], we extend those ideas and develop algorithms with provable error bounds for a variety of matrix operations. One of the main contributions of our work is to demonstrate that a "sketch" consisting of a small judiciously chosen random sample of rows and/or columns of the input matrix or matrices is adequate for provably rapid and efficient approximation of several common matrix operations. We believe that the underlying principle of using nonuniform sampling to create "sketches" of the data in a small number of passes (and "pass-efficient" approaches more generally) constitutes an appealing and fruitful direction for algorithmic research in order to address the size and nature of modern data sets.

In the present paper, we present two simple and intuitive algorithms which, when given an $m \times n$ matrix A and an $n \times p$ matrix B, compute an approximation to the product AB. In the first algorithm, the BasicMatrixMultiplication algorithm of section 4, we perform c independent trials, where in each trial we randomly sample an element of $\{1, 2, \ldots, n\}$ with an appropriate probability distribution $\mathcal{P}$ on $\{1, 2, \ldots, n\}$. We form an $m \times c$ matrix C consisting of the sampled columns of A, each scaled appropriately, and we form a $c \times n$ matrix R using the corresponding rows of B, again scaled appropriately. The choice of $\mathcal{P}$ and the column and row scaling are crucial features of the algorithm. When these are chosen judiciously, we show that CR is a good approximation to AB. More precisely, we show that

$$\|AB - CR\|_F = O(\|A\|_F \|B\|_F / \sqrt{c}),$$

where $\|\cdot\|_F$ denotes the Frobenius norm, i.e., $\|A\|_F^2 = \sum_{i,j} A_{ij}^2$, holds in expectation and with high probability. Thus, in particular, when $B = A^T$ we have that if $c = \Omega(1/\epsilon^2)$, then $\|AA^T - CC^T\|_F \leq \epsilon \|A\|_F^2$ holds with high probability. This algorithm can be implemented without storing the matrices A and B in RAM, provided it can make two passes over the matrices stored in external memory and use $O(c(m+n+p))$ additional RAM; thus it will be efficient in the pass-efficient model.

In the second algorithm, the ElementwiseMatrixMultiplication algorithm of section 5, which is an extension of ideas from [2, 1], elements of A and B, rather than columns and rows, are randomly either zeroed out or kept and rescaled, thereby constructing matrices $\tilde{A}$ and $\tilde{B}$. Although this algorithm lacks a useful bound on

$\|AB - \tilde{A}\tilde{B}\|_F$, under appropriate assumptions a bound on the spectral norm of the form

$$\|AB - \tilde{A}\tilde{B}\|_2 = O(\|A\|_F \|B\|_F / \sqrt{c})$$

holds with high probability.

After this introduction, we provide in section 2 a review of the relevant linear algebra, and in section 3 we introduce the pass-efficient model of data-streaming computation and discuss several technical sampling lemmas. In section 4 we introduce and analyze in detail the BasicMatrixMultiplication algorithm to approximate the product of two matrices. Then, in section 5 we describe and analyze the ElementwiseMatrixMultiplication algorithm which is based on the ideas of [2, 1]. Finally, in section 6 we provide a discussion and conclusion. In the appendix, we provide further analysis of the BasicMatrixMultiplication algorithm.

2. Review of linear algebra. This section contains a review of some linear algebra that will be useful throughout the paper. For more detail, see [18, 20, 25, 6] and the references therein.

For a vector $x \in \mathbb{R}^n$ we let $|x| = \left(\sum_{i=1}^n |x_i|^2\right)^{1/2}$ denote its Euclidean length. For a matrix $A \in \mathbb{R}^{m \times n}$ we let $A^{(j)}$, $j = 1, \ldots, n$, denote the jth column of A as a column vector and $A_{(i)}$, $i = 1, \ldots, m$, denote the ith row of A as a row vector. We denote matrix norms by $\|A\|_\xi$, using subscripts to distinguish between various norms. Of particular interest will be the Frobenius norm which is defined by

$$\|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n A_{ij}^2}, \tag{1}$$

and the spectral norm which is defined by

$$\|A\|_2 = \sup_{x \in \mathbb{R}^n,\ x \neq 0} \frac{|Ax|}{|x|}. \tag{2}$$

These norms are related to each other as $\|A\|_2 \leq \|A\|_F \leq \sqrt{n}\,\|A\|_2$.

3. The pass-efficient model and sampling lemmas. In this section, we informally define a computational model in which the computational resources are the number of passes over the data and the additional space and additional time required. In addition, we present several technical sampling lemmas.

3.1. The pass-efficient model. The pass-efficient model of data-streaming computation is a model that is motivated by the observation that in modern computers the amount of disk storage, i.e., sequential access memory, has increased very rapidly while random access memory (RAM) and computing speeds have increased at a substantially slower pace [10]. Thus, one has the ability to store very large amounts of data but does not have random access to the data. Additionally, processing the data with algorithms that take low polynomial time or linear time with large constants is prohibitive.

To model this phenomenon, we consider the pass-efficient model, in which the three computational resources of interest are the number of passes over the data and the additional space and time required [10]. The data are assumed to be stored in an external disk space, to consist of elements whose size is bounded by a constant, and to be presented to an algorithm on a read-only tape. The only access an algorithm has

to the data is via a pass, where a *pass* over the data is a sequential read of the entire input from disk where only a constant amount of processing time is permitted per bit read. Note that this is a more restrictive notion of a pass over the data than in other data-streaming models [23, 19, 14]; in particular, in the pass-efficient model only a constant rather than a logarithmic (in the data input length) amount of computation is permitted per bit read. In addition to the external disk space to store the data and to a small number of passes over the data, an algorithm in the pass-efficient model is permitted to use *additional RAM space* and *additional computation time*. An algorithm operating in this model is considered *pass-efficient* if it requires a fixed number of passes, independent of the input size, and additional space and time which are sublinear in the length of the data stream in order to compute a "description" of the solution, which is then returned by the algorithm. A *description* of the solution is either an explicit solution (if that is possible within the specified additional space and time) or an implicit representation of the solution that can be computed in the allotted additional space and time, and that can be expanded into an explicit solution with the additional expense of one pass over the data and linear (in the data input length) additional space and time. Note that, depending on the application, this last step may or may not be necessary. Note also that if the data are represented by an $m \times n$ matrix, then the data stream has length $O(mn)$ and an algorithm which uses additional space and time that is linear in the number of data points or in the dimensionality of the data points, i.e., that is $O(m)$ or $O(n)$, is sublinear in the length of the data stream and thus is pass-efficient. We will be primarily interested in models that require additional space and time that is either $O(m+n)$ or constant with respect to m and n.

The *sparse-unordered representation* of data is a form of data representation in which each element of the data stream consists of a pair $((i, j), A_{ij})$ where the elements in the data stream may be unordered with respect to the indices (i, j), and only the nonzero elements of the matrix A need to be presented. This very general form is suited to applications where, e.g., multiple agents may write parts of a matrix to a central database and where one cannot make assumptions about the rules for write-conflict resolution. The data stream read by algorithms in the pass-efficient model is assumed to be presented in the *sparse-unordered representation*. Other related methods of data representation have been studied within the data-streaming context; see, e.g., [17] for applications to the problem of dynamic histogram maintenance.

3.2. Sampling lemmas. In this section we present two sampling primitives that will be used by our algorithms. Consider the SELECT algorithm presented in Figure 1. The following lemma establishes that in one pass over the data one can sample an element according to certain probability distributions.

LEMMA 1. *Suppose that* $\{a_1, \ldots, a_n\}$, $a_i \geq 0$, *are read in one pass, i.e., one sequential read over the data, by the* SELECT *algorithm. Then the* SELECT *algorithm requires* $O(1)$, *i.e., constant with respect to* n, *additional storage space and returns a random* i^* *sampled from the probability distribution* $\mathbf{Pr}[i^* = i] = a_i / \sum_{i'=1}^{n} a_{i'}$.

Proof. First, note that retaining the selected value and the running sum requires $O(1)$ additional space. The remainder of the proof is by induction. After reading the first element a_1, $i^* = 1$ with probability $a_1/a_1 = 1$. Let $D_\ell = \sum_{i'=1}^{\ell} a_{i'}$ and suppose that the algorithm has read $a_1, \ldots, a_\ell$ thus far and has retained the running sum D_ℓ and a sample i^* such that $\mathbf{Pr}[i^* = i] = a_i/D_\ell$. Upon reading $a_{\ell+1}$ the algorithm lets $i^* = \ell + 1$ with probability $a_{\ell+1}/D_{\ell+1}$ and retains i^* at its previous value otherwise. At that point, clearly $\mathbf{Pr}[i^* = \ell + 1] = a_{\ell+1}/D_{\ell+1}$; furthermore for

SELECT Algorithm.

Input: $\{a_1, \ldots, a_n\}$, $a_i \geq 0$, read in one pass, i.e., one sequential read, over the data.

Output: i^*, a_{i^*}.

1. $D = 0$.
2. For $i = 1$ to n,
   (a) $D = D + a_i$.
   (b) With probability a_i/D, let $i^* = i$ and $a_{i^*} = a_i$.
3. Return i^*, a_{i^*}.

FIG. 1. *The* SELECT *algorithm.*

$i = 1, \ldots, \ell$, $\mathbf{Pr}[i^* = i] = \frac{a_i}{D_\ell}(1 - \frac{a_{\ell+1}}{D_{\ell+1}}) = \frac{a_i}{D_{\ell+1}}$. By induction this result holds when $\ell + 1 = n$ and the lemma follows. □

In a single pass over the data this algorithm can be run in parallel with $O(s)$ total memory units to return s independent samples $i_1^*, \ldots, i_s^*$ such that for each i_t^*, $t = 1, \ldots, s$, we have $\mathbf{Pr}[i_t^* = i] = a_i / \sum_{i'=1}^n a_{i'}$.

The next lemma is a modification of the previous lemma to deal with the case where a matrix is read in the sparse-unordered representation and one wants to choose a row label with a certain probability. This can also be implemented in $O(1)$ additional space and time. Note that a trivial modification would permit choosing a column label.

LEMMA 2. *Suppose that $A \in \mathbb{R}^{m \times n}$ is presented in the sparse-unordered representation and is read in one pass, i.e., one sequential read over the data, by the* SELECT *algorithm. Then the algorithm requires $O(1)$, i.e., constant with respect to m and n, additional storage space and returns i^*, j^* such that $\mathbf{Pr}[i^* = i \wedge j^* = j] = A_{ij}^2 / \|A\|_F^2$ and thus $\mathbf{Pr}[i^* = i] = |A_{(i)}|^2 / \|A\|_F^2$.*

Proof. Since $A_{i^*j^*}^2 > 0$ the first claim follows from Lemma 1; the second follows since

$$\mathbf{Pr}[i^* = i] = \sum_{j=1}^n \mathbf{Pr}[i^* = i \wedge j^* = j] = \sum_{j=1}^n \frac{A_{ij}^2}{\|A\|_F^2} = \frac{|A_{(i)}|^2}{\|A\|_F^2}. \qquad \square$$

Algorithms such as the SELECT algorithm, which select elements from a large pool of elements whose size is initially unknown, have been called reservoir algorithms [28].

4. The basic matrix multiplication approximation algorithm. In this section, which describes the main result of the paper, the BASICMATRIXMULTIPLICATION algorithm to approximate the product of two matrices is presented; it is analyzed in this section and in the appendix. After describing the algorithm in section 4.1 we describe its implementation and running time issues in section 4.2. In section 4.3 we analyze the algorithm and provide error bounds for arbitrary probability distributions; in section 4.4 error bounds are derived for probability distributions which are nearly optimal in a well-defined sense. We provide further discussion of the algorithm in section 6, and in the appendix we provide further analysis of the BASICMATRIXMULTIPLICATION algorithm.

BasicMatrixMultiplication Algorithm.

Input: $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$, $c \in \mathbb{Z}^+$ such that $1 \le c \le n$, and $\{p_i\}_{i=1}^n$ such that $p_i \ge 0$ and $\sum_{i=1}^n p_i = 1$.

Output: $C \in \mathbb{R}^{m\times c}$ and $R \in \mathbb{R}^{c\times p}$.

1. For $t = 1$ to c,
   (a) Pick $i_t \in \{1, \ldots, n\}$ with $\mathbf{Pr}[i_t = k] = p_k$, $k = 1, \ldots, n$, independently and with replacement.
   (b) Set $C^{(t)} = A^{(i_t)}/\sqrt{cp_{i_t}}$ and $R_{(t)} = B_{(i_t)}/\sqrt{cp_{i_t}}$.
2. Return C, R.

Fig. 2. *The* BasicMatrixMultiplication *algorithm.*

4.1. The algorithm. Recall that for $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$, the product AB may be written as the sum of n rank-one matrices

$$AB = \sum_{t=1}^{n} A^{(t)} B_{(t)}. \tag{3}$$

When matrix multiplication is formulated in this manner, a simple randomized algorithm to approximate the product matrix AB suggests itself: randomly sample with replacement from the terms in the summation c times according to a probability distribution $\{p_i\}_{i=1}^n$, scale each term in an appropriate manner, and output the sum of the scaled terms. If $m = p = 1$, then $A^{(t)}, B_{(t)} \in \mathbb{R}$ and it is straightforward to show that this sampling procedure produces an unbiased estimator for the sum. When the terms in the sum are rank-one matrices, as in (3), we show that similar results hold.

Consider the BasicMatrixMultiplication algorithm described in Figure 2. When this algorithm is given as input two matrices A and B, a probability distribution $\{p_i\}_{i=1}^n$, and a number c of column-row pairs to choose, it returns as output matrices C and R such that the product CR is an approximation to AB. Observe that since

$$CR = \sum_{t=1}^{c} C^{(t)} R_{(t)} = \sum_{t=1}^{c} \frac{1}{cp_{i_t}} A^{(i_t)} B_{(i_t)}$$

the procedure for sampling and scaling column and row pairs that is used in the BasicMatrixMultiplication algorithm corresponds to sampling terms in (3) and rescaling by dividing by cp_{i_t} if the tth term is sampled. Alternatively, one could define the sampling matrix $S \in \mathbb{R}^{n\times c}$ to be the zero-one matrix where $S_{ij} = 1$ if the ith column of A (and thus also the ith row of B) is chosen in the jth independent random trial, and $S_{ij} = 0$ otherwise. If the rescaling matrix $D \in \mathbb{R}^{c\times c}$ is the diagonal matrix with $D_{tt} = 1/\sqrt{cp_{i_t}}$, then

$$C = ASD \quad \text{and} \quad R = (SD)^T B$$

so that $CR = ASD(SD)^T B \approx AB$. Figure 3 presents a diagram illustrating the action of the BasicMatrixMultiplication algorithm. The product AB is shown as B and then A operating between the high-dimensional $\mathbb{R}^p$ and $\mathbb{R}^m$ via the high-dimensional $\mathbb{R}^n$; this is approximated by CR, which is shown as R and then C operating between $\mathbb{R}^p$ and $\mathbb{R}^m$ via the low-dimensional subspace $\mathbb{R}^c$. Also shown are the sampling matrix S and the diagonal rescaling matrix D.

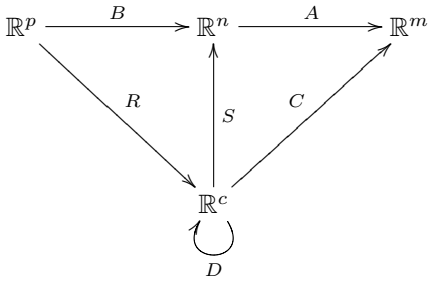


FIG. 3. *Diagram for the* BASICMATRIXMULTIPLICATION *algorithm.*

An important issue is the choice of the probabilities $\{p_i\}_{i=1}^n$ and the scaling. It is easily seen that the scaling of $1/\sqrt{cp_{i_t}}$ used in the BASICMATRIXMULTIPLICATION algorithm makes CR an unbiased estimator of AB; see Lemma 3. Lemma 3 also computes $\mathbf{Var}\left[(CR)_{ij}\right]$ under general probabilities $\{p_i\}_{i=1}^n$. We then compute $\mathbf{E}[\,\|AB - CR\|_F^2]$ and see that probabilities of the form $p_k = |A^{(k)}||B_{(k)}|/N, k = 1, \ldots, n$, where N is a normalization, are optimal in that they minimize this quantity; see Lemma 4.

This approach for approximating matrix multiplication has several advantages. First, it is conceptually simple, and in some cases it can be generalized to approximate the product of more than two matrices; see section A.1 for more on the latter point. Second, since the heart of the algorithm involves matrix multiplication of smaller matrices, it can use any algorithm in the literature for performing the desired matrix multiplication [18, 26, 8]. Third, this approach does not tamper with the sparsity of the matrices, unlike an algorithm that would project both A and B to the same random c-dimensional subspace and take the product of the projections. Finally, the algorithm can be easily implemented; see sections 4.2 and 6 for more discussion.

4.2. Implementation of the sampling and running time. To implement the BASICMATRIXMULTIPLICATION algorithm, it must be decided which elements of the input to sample and those elements must then be sampled. In the case of uniform sampling one can decide before the input is seen which column-row pairs to sample. Then, a single pass over the matrices is sufficient to sample the columns and rows of interest and to construct C and R; this requires $O(c(m+p))$ additional time and space. We will see below that it is useful to sample according to a nonuniform probability distribution that depends on column and row lengths, e.g., see (5) and (7). In order to decide which column-row pairs to sample in such a case, one pass through the matrices and $O(n)$ additional time and space is sufficient; in the additional space running totals of $|A^{(k)}|^2$ and $|B_{(k)}|^2$ are kept, so that after the first pass $|A^{(k)}|$, $|B_{(k)}|$, $k = 1, \ldots, n$, and thus the probabilities, can be calculated in $O(n)$ additional time. Then in a second pass the columns and rows of interest can be sampled and C and R can be constructed and stored; this requires $O(c(m+p))$ additional space and time. Thus, in addition to either one or two passes over the data, for both uniform and nonuniform sampling, $O(c(m+n+p))$ additional space and time is sufficient to sample from the matrices A and B of the input and to construct the matrices C and R.

If $B = A^T$ and nonuniform sampling is performed (assuming probabilities of the form (5) or (7)), the resource requirements are slightly different. Due to Lemma 2 we can select which columns of A to choose using constant (with respect to n) additional

space and time during the first pass. Then, during the second pass, these columns may be extracted and the matrices C and $R = C^T$ may be constructed using $O(cm)$ additional space and time; this will be used in the LINEARTIMESVD algorithm of [11]. Note that if only a constant-sized part of the columns of C is needed, as, for example, in the CONSTANTTIMESVD algorithm of [11], then extracting and storing this constant-sized subset of the samples desired may be performed using constant additional space and time.

4.3. Analysis of the algorithm for arbitrary probabilities. In this section we prove upper bounds for $\|AB - CR\|_F^2$, where C and R are returned from the BASICMATRIXMULTIPLICATION algorithm. Recall that by Jensen's inequality bounding $\|AB - CR\|_F^2$ (in expectation) implies a bound for $\|AB - CR\|_F$. Recall also that a bound on $\|AB - CR\|_F$ immediately provides a bound on $\|AB - CR\|_2$ since $\|AB - CR\|_2 \leq \|AB - CR\|_F$.

Our first lemma proves that the expectation of the (i,j)th element of the approximation is equal to the (i,j)th element of the exact product; it also describes the variance of the approximation of the (i,j)th element.

LEMMA 3. *Suppose $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$, $c \in \mathbb{Z}^+$ such that $1 \leq c \leq n$, and $\{p_i\}_{i=1}^n$ are such that $p_i \geq 0$ and $\sum_{i=1}^n p_i = 1$. Construct C and R with the* BASICMATRIXMULTIPLICATION *algorithm, and let CR be an approximation to AB. Then*

$$\mathbf{E}\left[(CR)_{ij}\right] = (AB)_{ij}$$

and

$$\mathbf{Var}\left[(CR)_{ij}\right] = \frac{1}{c}\sum_{k=1}^{n}\frac{A_{ik}^2 B_{kj}^2}{p_k} - \frac{1}{c}(AB)_{ij}^2.$$

Proof. Fix i, j. For $t = 1, \ldots, c$, define $X_t = \left(\frac{A^{(i_t)}B_{(i_t)}}{cp_{i_t}}\right)_{ij} = \frac{A_{ii_t}B_{i_t j}}{cp_{i_t}}$. Thus,

$$\mathbf{E}\left[X_t\right] = \sum_{k=1}^{n} p_k \frac{A_{ik}B_{kj}}{cp_k} = \frac{1}{c}(AB)_{ij} \quad \text{and} \quad \mathbf{E}\left[X_t^2\right] = \sum_{k=1}^{n}\frac{A_{ik}^2 B_{kj}^2}{c^2 p_k}.$$

Since by construction $(CR)_{ij} = \sum_{t=1}^c X_t$, we have $\mathbf{E}\left[(CR)_{ij}\right] = \sum_{t=1}^c \mathbf{E}\left[X_t\right] = (AB)_{ij}$. Since $(CR)_{ij}$ is the sum of c independent random variables, $\mathbf{Var}\left[(CR)_{ij}\right] = \sum_{t=1}^c \mathbf{Var}\left[X_t\right]$. Since $\mathbf{Var}\left[X_t\right] = \mathbf{E}\left[X_t^2\right] - \mathbf{E}\left[X_t\right]^2$, we see that

$$\mathbf{Var}\left[X_t\right] = \sum_{k=1}^{n}\frac{A_{ik}^2 B_{kj}^2}{c^2 p_k} - \frac{1}{c^2}(AB)_{ij}^2$$

and the lemma follows. ∎

Using this lemma, we bound $\mathbf{E}\left[\,\|AB - CR\|_F^2\,\right]$ in the next lemma. In addition, we note how this measure of the error depends on the p_i's.

LEMMA 4. *Suppose $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$, $c \in \mathbb{Z}^+$ such that $1 \leq c \leq n$, and $\{p_i\}_{i=1}^n$ are such that $p_i \geq 0$ and $\sum_{i=1}^n p_i = 1$. Construct C and R with the* BASICMATRIXMULTIPLICATION *algorithm, and let CR be an approximation to AB.*

Then

$$\mathbf{E}\left[\,\|AB-CR\|_F^2\right]=\sum_{k=1}^{n}\frac{\left|A^{(k)}\right|^2\left|B_{(k)}\right|^2}{cp_k}-\frac{1}{c}\,\|AB\|_F^2\,. \tag{4}$$

Furthermore, if

$$p_k=\frac{\left|A^{(k)}\right|\left|B_{(k)}\right|}{\sum_{k'=1}^{n}\left|A^{(k')}\right|\left|B_{(k')}\right|}, \tag{5}$$

then

$$\mathbf{E}\left[\,\|AB-CR\|_F^2\right]=\frac{1}{c}\left(\sum_{k=1}^{n}\left|A^{(k)}\right|\left|B_{(k)}\right|\right)^2-\frac{1}{c}\,\|AB\|_F^2\,. \tag{6}$$

This choice of p_k minimizes $\mathbf{E}\left[\,\|AB-CR\|_F^2\,\right]$ among possible choices for the sampling probabilities.

Proof. First, note that

$$\mathbf{E}\left[\,\|AB-CR\|_F^2\right]=\sum_{i=1}^{m}\sum_{j=1}^{p}\mathbf{E}\left[(AB-CR)_{ij}^2\right]=\sum_{i=1}^{m}\sum_{j=1}^{p}\mathbf{Var}\left[(CR)_{ij}\right].$$

Thus, from Lemma 3 it follows that

$$\begin{aligned}\mathbf{E}\left[\,\|AB-CR\|_F^2\right]&=\frac{1}{c}\sum_{k=1}^{n}\frac{1}{p_k}\left(\sum_i A_{ik}^2\right)\left(\sum_j B_{kj}^2\right)-\frac{1}{c}\,\|AB\|_F^2\\&=\frac{1}{c}\sum_{k=1}^{n}\frac{1}{p_k}\left|A^{(k)}\right|^2\left|B_{(k)}\right|^2-\frac{1}{c}\,\|AB\|_F^2\,.\end{aligned}$$

If the value $p_k=\frac{\left|A^{(k)}\right|\left|B_{(k)}\right|}{\sum_{k'=1}^{n}\left|A^{(k')}\right|\left|B_{(k')}\right|}$ is used in this expression, then

$$\mathbf{E}\left[\,\|AB-CR\|_F^2\right]=\frac{1}{c}\left(\sum_{k=1}^{n}\left|A^{(k)}\right|\left|B_{(k)}\right|\right)^2-\frac{1}{c}\,\|AB\|_F^2\,.$$

Finally, to prove that this choice for the p_k's minimizes $\mathbf{E}\left[\,\|AB-CR\|_F^2\,\right]$ define the function

$$f(p_1,\ldots,p_n)=\sum_{k=1}^{n}\frac{1}{p_k}\left|A^{(k)}\right|^2\left|B_{(k)}\right|^2,$$

which characterizes the dependence of $\mathbf{E}\left[\,\|AB-CR\|_F^2\,\right]$ on the p_k's. To minimize f subject to $\sum_{k=1}^{n}p_k=1$, introduce the Lagrange multiplier λ and define the function

$$g(p_1,\ldots,p_n)=f(p_1,\ldots,p_n)+\lambda\left(\sum_{k=1}^{n}p_k-1\right).$$

We then have at the minimum that

$$0 = \frac{\partial g}{\partial p_i} = \frac{-1}{p_i^2}\left|A^{(i)}\right|^2 \left|B_{(i)}\right|^2 + \lambda.$$

Thus,

$$p_i = \frac{\left|A^{(i)}\right| \left|B_{(i)}\right|}{\sqrt{\lambda}} = \frac{\left|A^{(i)}\right| \left|B_{(i)}\right|}{\sum_{i'=1}^{n} \left|A^{(i')}\right| \left|B_{(i')}\right|},$$

where the second equality comes from solving for $\sqrt{\lambda}$ in $\sum_{k=1}^{n-1} p_k = 1$. That these probabilities are a minimum follows since $\frac{\partial^2 g}{\partial p_i^2} > 0 \ \forall i$ such that $|A^{(i)}|^2 |B_{(i)}|^2 > 0$. □

4.4. Analysis of the algorithm for nearly optimal probabilities. With Lemma 4 and using Jensen's inequality, upper bounds on quantities such as $\mathbf{E}\left[\|AB - CR\|_F^2\right]$ and $\mathbf{E}\left[\|AB - CR\|_F\right]$ may be obtained for various sampling probabilities $\{p_i\}_{i=1}^n$. In many cases, by using a martingale argument to show that the error is tightly concentrated around its mean, the expectations in these bounds may be removed and the corresponding results can be shown to hold with high probability.

Rather than presenting these results in their full generality, we restrict our attention to two particular sets of probabilities. We will say that the sampling probabilities $p_k = \frac{|A^{(k)}||B_{(k)}|}{\sum_{k'=1}^{n} |A^{(k')}||B_{(k')}|}$ are the *optimal probabilities* since they minimize $\mathbf{E}\left[\, \|AB - CR\|_F^2 \,\right]$, which as Lemma 4 shows is one natural measure of the error. We will say that a set of sampling probabilities $\{p_i\}_{i=1}^n$ are *nearly optimal probabilities* if $p_k \geq \frac{\beta |A^{(k)}||B_{(k)}|}{\sum_{k'=1}^{n} |A^{(k')}||B_{(k')}|}$ for some positive constant $\beta \leq 1$.

We now prove, for nearly optimal sampling probabilities, results analogous to those of Lemma 4, and also that the corresponding results with the expectations removed hold with high probability. Notice that if $\beta \neq 1$, then we suffer a small β-dependent loss in accuracy.

THEOREM 1. *Suppose $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{n \times p}$, $c \in \mathbb{Z}^+$ such that $1 \leq c \leq n$, and $\{p_i\}_{i=1}^n$ are such that $\sum_{i=1}^n p_i = 1$ and such that for some positive constant $\beta \leq 1$*

$$p_k \geq \frac{\beta \left|A^{(k)}\right| \left|B_{(k)}\right|}{\sum_{k'=1}^{n} \left|A^{(k')}\right| \left|B_{(k')}\right|}. \tag{7}$$

Construct C and R with the BASICMATRIXMULTIPLICATION *algorithm, and let CR be an approximation to AB. Then*

$$\mathbf{E}\left[\, \|AB - CR\|_F^2 \right] \leq \frac{1}{\beta c} \|A\|_F^2 \, \|B\|_F^2 . \tag{8}$$

Furthermore, let $\delta \in (0,1)$ and $\eta = 1 + \sqrt{(8/\beta)\log(1/\delta)}$. Then, with probability at least $1 - \delta$,

$$\|AB - CR\|_F^2 \leq \frac{\eta^2}{\beta c} \|A\|_F^2 \, \|B\|_F^2 . \tag{9}$$

Proof. Following reasoning similar to that of Lemma 4 and using the probabilities

of (7), we see that

$$\begin{aligned}\mathbf{E}\left[\|AB-CR\|_F^2\right] &\le \frac{1}{c}\sum_{k=1}^{n}\frac{1}{p_k}\left|A^{(k)}\right|^2\left|B_{(k)}\right|^2\\ &\le \frac{1}{\beta c}\left(\sum_{k=1}^{n}\left|A^{(k)}\right|\left|B_{(k)}\right|\right)^2\\ &\le \frac{1}{\beta c}\|A\|_F^2\,\|B\|_F^2,\end{aligned}$$

where the last inequality follows due to the Cauchy–Schwarz inequality. Next, define the event $\mathcal{E}_2$ to be

$$\|AB-CR\|_F \le \frac{\eta}{\sqrt{\beta c}}\|A\|_F\,\|B\|_F \tag{10}$$

and note that to prove the remainder of the theorem it suffices to prove that $\mathbf{Pr}[\mathcal{E}_2] \ge 1-\delta$. To that end, note that C and R and thus $CR = \sum_{t=1}^{c}\frac{1}{cp_{i_t}}A^{i_t}B_{i_t}$ are formed by randomly selecting c elements from $\{1,\ldots,n\}$, independently and with replacement. Let the sequence of elements chosen be $\{i_t\}_{t=1}^{c}$. Consider the function

$$F(i_1,\ldots,i_c) = \|AB-CR\|_F. \tag{11}$$

We will show that changing one i_t at a time does not change F too much; this will enable us to apply a martingale inequality. To this end, consider changing one of the i_t to i_t' while keeping the other i_t's the same. Then construct the corresponding C' and R'. Note that C' differs from C in only a single column and that R' differs from R in only a single row. Thus,

$$\|CR-C'R'\|_F = \left\|\frac{A^{(i_t)}B_{(i_t)}}{cp_{i_t}} - \frac{A^{(i_t')}B_{(i_t')}}{cp_{i_t'}}\right\|_F \tag{12}$$

$$\le \frac{1}{cp_{i_t}}\left\|A^{(i_t)}B_{(i_t)}\right\|_F + \frac{1}{cp_{i_t'}}\left\|A^{(i_t')}B_{(i_t')}\right\|_F \tag{13}$$

$$= \frac{1}{cp_{i_t}}\left|A^{(i_t)}\right|\left|B_{(i_t)}\right| + \frac{1}{cp_{i_t'}}\left|A^{(i_t')}\right|\left|B_{(i_t')}\right| \tag{14}$$

$$\le \frac{2}{c}\max_{\alpha}\frac{\left|A^{(\alpha)}\right|\left|B_{(\alpha)}\right|}{p_\alpha}. \tag{15}$$

Equation (12) follows by construction and (14) follows since $\|xy^T\|_F = |x|\,|y|$ for $x\in\mathbb{R}^n$ and $y\in\mathbb{R}^n$. Thus, using the probabilities (7) and employing the Cauchy–Schwarz inequality we see that

$$\|CR-C'R'\|_F \le \frac{2}{\beta c}\sum_{k=1}^{n}\left|A^{(k)}\right|\left|B_{(k)}\right| \tag{16}$$

$$\le \frac{2}{\beta c}\|A\|_F\,\|B\|_F. \tag{17}$$

Therefore, using the triangle inequality we see that

$$\begin{aligned}\|AB-CR\|_F &\le \|AB-C'R'\|_F + \|C'R'-CR\|_F\\ &\le \|AB-C'R'\|_F + \frac{2}{\beta c}\|A\|_F\,\|B\|_F.\end{aligned} \tag{18}$$

By similar reasoning, we can derive

$$\|AB - C'R'\|_F \leq \|AB - CR\|_F + \frac{2}{\beta c} \|A\|_F \|B\|_F . \tag{19}$$

Define $\Delta = \frac{2}{\beta c} \|A\|_F \|B\|_F$; thus,

$$|F(i_1, \ldots, i_k, \ldots, i_c) - F(i_1, \ldots, i'_k, \ldots, i_c)| \leq \Delta. \tag{20}$$

Let $\gamma = \sqrt{2c \log(1/\delta)}\Delta$ and consider the associated Doob martingale. By the Hoeffding–Azuma inequality [22],

$$\mathbf{Pr}\left[\|AB - CR\|_F \geq \frac{1}{\sqrt{\beta c}} \|A\|_F \|B\|_F + \gamma \right] \leq \exp\left(-\gamma^2/2c\Delta^2\right) = \delta \tag{21}$$

and the theorem follows. □

An immediate consequence of Theorem 1 is that by choosing enough column-row pairs, the error in the approximation of the matrix product can be made arbitrarily small. In particular, if $c \geq 1/\beta\epsilon^2$, then by using Jensen's inequality it follows that

$$\mathbf{E}\left[\|AB - CR\|_F \right] \leq \epsilon \|A\|_F \|B\|_F \tag{22}$$

and if, in addition, $c \geq \eta^2/\beta\epsilon^2$, then with probability at least $1 - \delta$

$$\|AB - CR\|_F \leq \epsilon \|A\|_F \|B\|_F . \tag{23}$$

In certain applications, e.g., [11, 12], one is interested in an application of Theorem 1 to the case that $B = A^T$, i.e., one is interested in approximating $\left\|AA^T - CC^T\right\|_F^2$. In this case, sampling column-row pairs corresponds to sampling columns of A, and nearly optimal probabilities will be those such that $p_k \geq \frac{\beta \left|A^{(k)}\right|^2}{\|A\|_F^2}$ for some positive $\beta \leq 1$. By taking $B = A^T$ and applying Jensen's inequality, we have the following theorem as a corollary of Theorem 1.

THEOREM 2. *Suppose $A \in \mathbb{R}^{m \times n}$, $c \in \mathbb{Z}^+$, $1 \leq c \leq n$, and $\{p_i\}_{i=1}^n$ are such that $\sum_{i=1}^n p_i = 1$ and such that $p_k \geq \frac{\beta \left|A^{(k)}\right|^2}{\|A\|_F^2}$ for some positive constant $\beta \leq 1$. Furthermore, let $\delta \in (0,1)$ and $\eta = 1 + \sqrt{(8/\beta)\log(1/\delta)}$. Construct C (and $R = C^T$) with the* BASICMATRIXMULTIPLICATION *algorithm, and let CC^T be an approximation to AA^T. Then*

$$\mathbf{E}\left[\left\|AA^T - CC^T\right\|_F \right] \leq \frac{1}{\sqrt{\beta c}} \|A\|_F^2 \tag{24}$$

and with probability at least $1 - \delta$,

$$\left\|AA^T - CC^T\right\|_F \leq \frac{\eta}{\sqrt{\beta c}} \|A\|_F^2 . \tag{25}$$

5. A second matrix multiplication algorithm. In this section we describe the ELEMENTWISEMATRIXMULTIPLICATION algorithm to approximate the product of two matrices. First, in section 5.1, we describe the algorithm, its implementation, and running time issues; then in section 5.2 we analyze the algorithm and bound its error with respect to both the Frobenius and spectral norms. We will see that the algorithm returns good approximations with respect to the spectral norm but not with respect to the Frobenius norm.

ElementwiseMatrixMultiplication Algorithm.

Input: $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$, $\{p_{ij}\}_{i,j=1}^{m,n}$ such that $0 \le p_{ij} \le 1$, and $\{q_{ij}\}_{i,j=1}^{n,p}$ such that $0 \le q_{ij} \le 1$.

Output: $S \in \mathbb{R}^{m\times n}$ and $R \in \mathbb{R}^{n\times p}$.

Algorithm:

1. For $i = 1$ to m and $j = 1$ to n, independently,
   (a) Set
$$S_{ij} = \begin{cases} A_{ij}/p_{ij} & \text{with probability } p_{ij}, \\ 0 & \text{otherwise.} \end{cases}$$
2. For $i = 1$ to n and $j = 1$ to p, independently,
   (a) Set
$$R_{ij} = \begin{cases} B_{ij}/q_{ij} & \text{with probability } q_{ij}, \\ 0 & \text{otherwise.} \end{cases}$$
3. Return S, R.

Fig. 4. *The* ElementwiseMatrixMultiplication *algorithm.*

5.1. The algorithm and its implementation. The method to approximate the product of two matrices that is presented in this section differs from the previous algorithm and is inspired by [2] and [1]. In [2] the singular value decomposition of a matrix is approximated using elementwise uniform sampling; in [1] this approach is extended to include nonuniform sampling probabilities of a certain natural form. Since neither of these papers applies these methods to approximate matrix multiplication, we do so here for comparison with the BasicMatrixMultiplication algorithm.

Consider the ElementwiseMatrixMultiplication algorithm presented in Figure 4. When this algorithm is given as input two matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$ it creates two matrices $S \in \mathbb{R}^{m\times n}$ and $R \in \mathbb{R}^{n\times p}$ by keeping a few elements of A and a few elements of B, respectively, scaling in an appropriate manner those elements that are kept, and zeroing out the remaining elements. The algorithm then returns matrices S and R such that the product SR is an approximation to AB. Note that since S and R are formed independently of each other the algorithm does not keep "corresponding" elements; doing so would introduce dependence that would complicate the analysis.

The ElementwiseMatrixMultiplication algorithm can be implemented with the nonuniform probabilities used in this section with two passes over the data; we leave it as an open problem whether a single pass suffices when working within the pass-efficient framework. This algorithm differs from the BasicMatrixMultiplication algorithm in that we get an expected number of elements so we have an expected additional space required for storage and an expected additional time required for the associated sparse matrix multiplication. We do not provide a detailed analysis of these random variables.

5.2. Analysis of the algorithm. In this section we present error bounds for both $\|AB - SR\|_F$ and $\|AB - SR\|_2$. While the Frobenius norm error bound for this algorithm is rather easy to derive using very intuitive probability distributions, the spectral norm bound is more complicated and requires some additional technicalities.

Since whether or not (for a given i,j) $S_{ij}=0$ or $S_{ij}=A_{ij}/p_{ij}$ we have that $A_{ij}-S_{ij}$ is large (and similarly for the matrix R and thus the matrix SR) it is plausible that the ElementwiseMatrixMultiplication algorithm does not have a good bound for $\mathbf{E}\left[\,\|AB-SR\|_F^2\,\right]$. This intuition is formalized in the following lemma. Note that ℓ and ℓ' are chosen such that not more than ℓ and ℓ' of the elements of the matrices A and B are retained in expectation, respectively.

Lemma 5. *Suppose $A\in\mathbb{R}^{m\times n}$ and $B\in\mathbb{R}^{n\times p}$, let $\ell,\ell'\in\mathbb{Z}^+$, and let $p_{ij}=\min\{1,\ell A_{ij}^2/\|A\|_F^2\}$ and $q_{ij}=\min\{1,\ell' B_{ij}^2/\|B\|_F^2\}$. Construct S and R with the* ElementwiseMatrixMultiplication *algorithm, and let SR be an approximation to AB. Then, $\forall i,j$,*

$$\mathbf{E}\left[(SR)_{ij}\right]=(AB)_{ij},$$

$$\mathbf{Var}\left[(SR)_{ij}\right]=\sum_{k=1}^{n}\frac{A_{ik}^2}{p_{ik}}\frac{B_{kj}^2}{q_{kj}}-\sum_{k=1}^{n}A_{ik}^2B_{kj}^2$$

$$\mathbf{E}\left[\,\|AB-SR\|_F^2\right]\geq\frac{mpn}{\ell\ell'}\,\|A\|_F^2\,\|B\|_F^2-\sum_{k=1}^{n}\left|A^{(k)}\right|^2\left|B_{(k)}\right|^2. \tag{26}$$

Proof. Let us first fix i,j. Then, since for every k we have that $S_{ik}=A_{ik}/p_{ik}$ with probability p_{ik} and $S_{ik}=0$ with probability $1-p_{ik}$, we have that $\mathbf{E}\left[S_{ik}\right]=A_{ik}$; similarly for R_{kj}, we have that $\mathbf{E}\left[R_{kj}\right]=B_{kj}$. Thus, since S and R have been constructed independently, we have that

$$\mathbf{E}\left[(SR)_{ij}\right]=\mathbf{E}\left[\sum_{k=1}^{n}S_{ik}R_{kj}\right]=\sum_{k=1}^{n}\mathbf{E}\left[S_{ik}\right]\mathbf{E}\left[R_{kj}\right]=(AB)_{ij}.$$

Since $\mathbf{Var}\left[(SR)_{ij}\right]=\mathbf{E}\left[(SR)_{ij}^2\right]-\mathbf{E}\left[(SR)_{ij}\right]^2$ and since $(SR)_{ij}=\sum_{k=1}^{n}S_{ik}R_{kj}$ we get that

$$\begin{aligned}
\mathbf{Var}\left[(SR)_{ij}\right]&=\sum_{k_1=1}^{n}\sum_{k_2=1}^{n}\mathbf{E}\left[S_{ik_1}R_{k_1j}S_{ik_2}R_{k_2j}\right]-\mathbf{E}\left[(SR)_{ij}\right]^2\\
&=\sum_{k=1}^{n}\mathbf{E}\left[S_{ik}^2\right]\mathbf{E}\left[R_{kj}^2\right]+\sum_{k_1=1}^{n}\sum_{k_2\neq k_1}\mathbf{E}\left[S_{ik_1}\right]\mathbf{E}\left[R_{k_1j}\right]\mathbf{E}\left[S_{ik_2}\right]\mathbf{E}\left[R_{k_2j}\right]-(AB)_{ij}^2\\
&=\sum_{k=1}^{n}\frac{A_{ik}^2}{p_{ik}}\frac{B_{kj}^2}{q_{kj}}+\sum_{k_1=1}^{n}\sum_{k_2\neq k_1}A_{ik_1}B_{k_1j}A_{ik_2}B_{k_2j}-(AB)_{ij}^2\\
&=\sum_{k=1}^{n}\frac{A_{ik}^2}{p_{ik}}\frac{B_{kj}^2}{q_{kj}}-\sum_{k=1}^{n}A_{ik}^2B_{kj}^2,
\end{aligned}$$

where the last line follows by adding and subtracting $\sum_{k_1=1}^{n}\sum_{k_2=k_1}A_{ik_1}B_{k_1j}A_{ik_2}B_{k_2j}$ from the second-to-last line.

Thus, since $\mathbf{E}\left[\,\|AB-SR\|_F^2\,\right]=\sum_{i=1}^{m}\sum_{j=1}^{p}\mathbf{Var}\left[(SR)_{ij}\right]$ and since the probabilities p_{ij} and q_{ij} are such that $1/p_{ik}\geq\|A\|_F^2/\ell A_{ik}^2$ and $1/q_{kj}\geq\|B\|_F^2/\ell' B_{kj}^2$ we get

that

$$\mathbf{E}\left[\,\|AB-SR\|_F^2\right] = \sum_{i=1}^{m}\sum_{j=1}^{p}\sum_{k=1}^{n}\frac{A_{ik}^2}{p_{ik}}\frac{B_{kj}^2}{q_{kj}} - \sum_{i=1}^{m}\sum_{j=1}^{p}\sum_{k=1}^{n}A_{ik}^2B_{kj}^2$$
$$\geq \sum_{i,j=1}^{m,p}\sum_{k=1}^{n}\frac{\|A\|_F^2\,\|B\|_F^2}{\ell\ell'} - \sum_{k=1}^{n}\left|A^{(k)}\right|^2\left|B_{(k)}\right|^2.$$

The lemma then follows. □

Next we show that although the ElementwiseMatrixMultiplication algorithm does not yield a nice error bound for the Frobenius norm, it does for the spectral norm. In order to prove Theorem 4, which provides our bound on $\|AB-SR\|_2$, we will use the following theorem, which follows immediately from a result that was proved in [1] and which shows that with high probability the spectrum of a random matrix is close to its expectation. The theorem is proved by using a generalization of a result of Füredi and Komlós [16], combined with a more recent concentration result of Krivelevich and Vu based on Talagrand's inequality [21].

Theorem 3. *Given an $n\times n$ matrix A, let $\hat{A}$ be any random matrix whose entries are independent random variables such that $\forall i,j$, $\mathbf{E}\big[\hat{A}_{ij}\big]=A_{ij}$, $\mathbf{Var}\big[\hat{A}_{ij}\big]\leq\sigma^2$, and*

$$\left|\hat{A}_{ij}-A_{ij}\right| \leq \frac{\sigma\sqrt{2n}}{\log^3(2n)}. \tag{27}$$

For any $n\geq 10$, with probability at least $1-1/(2n)$,

$$\|A-\hat{A}\|_2 < 7\sigma\sqrt{2n}. \tag{28}$$

Prior to stating the main result of this section, we must address a technical issue that arises in our effort to apply the above theorem in order to bound $\|AB-SR\|_2$. Note that the construction of the matrices S and R by the ElementwiseMatrixMultiplication algorithm may be viewed as adding carefully constructed random matrices E and D such that $S=A+E$ and $R=B+D$; see [2] and [1] for a discussion. As we will see below, if we can bound $\|E\|_2$ and $\|D\|_2$, then a bound for $\|AB-SR\|_2$ follows easily. Since we will apply Theorem 3 in order to obtain such bounds, we need to satisfy the range constraint (27). Sampling with respect to the nonuniform probability distribution of Lemma 5 might violate this constraint since, in the unlikely event that a small element is kept, the resulting entry $S_{ij}=A_{ij}/p_{ij}$ will be very large (and similarly for R). Thus, following [1], we modify our sampling probabilities so that small elements are kept with a slightly larger probability which is proportional to $|A_{ij}|$ instead of A_{ij}^2:

$$p_{ij} = \begin{cases} \min\{1,\ell A_{ij}^2/\|A\|_F^2\} & \text{if } |A_{ij}| > \frac{\|A\|_F\log^3(2n)}{\sqrt{2n\ell}},\\ \min\left\{1,\frac{\sqrt{\ell}|A_{ij}|\log^3(2n)}{\sqrt{2n}\|A\|_F}\right\} & \text{otherwise,}\end{cases} \tag{29}$$

$$q_{ij} = \begin{cases} \min\{1,\ell' B_{ij}^2/\|B\|_F^2\} & \text{if } |B_{ij}| > \frac{\|B\|_F\log^3(2n)}{\sqrt{2n\ell'}},\\ \min\left\{1,\frac{\sqrt{\ell'}|B_{ij}|\log^3(2n)}{\sqrt{2n}\|B\|_F}\right\} & \text{otherwise.}\end{cases} \tag{30}$$

We now state and prove our main theorem of this section. In the interests of clarity we make several simplifying assumptions in the statement of the theorem.

THEOREM 4. *Suppose $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$, and let p_{ij} and q_{ij} be as specified in (29) and (30) with $\ell = \ell' \geq 1$. Assume that $\ell \leq \|A\|_F^2 / \max_{i,j} A_{ij}^2$ and that $\ell \leq \|B\|_F^2 / \max_{i,j} B_{ij}^2$; assume also that $m = n = p$ and that n is large enough so that $2n \geq \log^6(2n)$. Construct S and R with the* ELEMENTWISEMATRIXMULTIPLICATION *algorithm, and let SR be an approximation to AB. Then, with probability at least $1 - 1/n$,*

$$\|AB - SR\|_2 \leq \left(20\sqrt{\frac{n}{\ell}} + \frac{100n}{\ell}\right) \|A\|_F \|B\|_F . \tag{31}$$

Proof. By the assumptions on n and ℓ, neither p_{ij} nor q_{ij} exceed 1 for any i, j. Letting $E = S - A$ and $D = R - B$, we have

$$SR = (A + E)(B + D) = AB + AD + EB + ED. \tag{32}$$

Thus, by the triangle inequality and submultiplicitivity, we have that

$$\|AB - SR\|_2 \leq \|A\|_2 \|D\|_2 + \|E\|_2 \|B\|_2 + \|E\|_2 \|D\|_2 . \tag{33}$$

In order to apply Theorem 3 to $\|E\|_2$ and $\|D\|_2$ we first verify that the assumptions of the theorem are satisfied. From the proof of Lemma 5, we have that $\mathbf{E}[S_{ij}] = A_{ij}$. In addition,

$$\mathbf{Var}[S_{ij}] \leq \mathbf{E}\left[S_{ij}^2\right] = \frac{A_{ij}^2}{p_{ij}} \leq \frac{\|A\|_F^2}{\ell}$$

holds regardless of whether $|A_{ij}|$ is larger or smaller than the threshold. Similarly, we get that $\mathbf{E}[R_{ij}] = B_{ij}$ and that $\mathbf{Var}[D_{ij}] \leq \frac{\|B\|_F^2}{\ell}$. It is straightforward to show that regardless of whether or not $|A_{ij}|$ is above or below the threshold and regardless of whether or not $S_{ij} = 0$ or $S_{ij} = A_{ij}/p_{ij}$ we have that

$$|A_{ij} - S_{ij}| \leq \frac{\|A\|_F \sqrt{2n}}{\sqrt{\ell}\log^3(2n)}. \tag{34}$$

Similarly, one can show that

$$|B_{ij} - R_{ij}| \leq \frac{\|B\|_F \sqrt{2n}}{\sqrt{\ell}\log^3(2n)}. \tag{35}$$

Thus, the conditions of Theorem 3 are satisfied and with probability at least $1 - 1/2n$ each of the following holds:

$$\|E\|_2 \leq 7\|A\|_F \sqrt{2n}/\sqrt{\ell}, \tag{36}$$

$$\|D\|_2 \leq 7\|B\|_F \sqrt{2n}/\sqrt{\ell}. \tag{37}$$

Thus, with probability at least $1 - 1/n$ both of these inequalities hold. Combining the bounds (36) and (37) with (33), and since $\|\cdot\|_2 \leq \|\cdot\|_F$, we have

$$\begin{aligned}
\|AB - SR\|_2 &\leq \|A\|_2 \|D\|_2 + \|E\|_2 \|B\|_2 + \|E\|_2 \|D\|_2 \\
&\leq \frac{7\sqrt{2n}\|A\|_F \|B\|_F}{\sqrt{\ell}} + \frac{7\sqrt{2n}\|A\|_F \|B\|_F}{\sqrt{\ell}} + \frac{98n\|A\|_F \|B\|_F}{\ell} \\
&\leq \left(20\sqrt{n/\ell} + 100n/\ell\right) \|A\|_F \|B\|_F . \qquad \square
\end{aligned}$$

Notice that if we let $\ell = cn$ in Theorem 4, then the error bound (31) becomes

$$\|AB - SR\|_2 \leq \left(\frac{20}{\sqrt{c}} + \frac{100}{c}\right) \|A\|_F \|B\|_F = O(1/\sqrt{c}) \|A\|_F \|B\|_F .$$

Comparison with (9) of Theorem 1 reveals that (since $\|\cdot\|_2 \leq \|\cdot\|_F$) both of our matrix multiplication algorithms have, asymptotically, a similar bound with respect to the spectral norm.

6. Discussion and conclusion. To the best of our knowledge, the only previous randomized algorithm that approximates the product of two matrices is that of Cohen and Lewis [7]. This algorithm is based on random walks in a graph representation of the input matrices and attempts to identify all high-valued entries in nonnegative matrix products in order to improve estimates (relative to exact sparse multiplication) by spending less time on small-valued entries. Their algorithm is more complicated than ours, it requires different graph representations of the input matrices if the matrices are allowed to contain negative entries, it needs to store the complete input matrices, and it is especially useful when the matrices are not sparse.

It is worth emphasizing how the BasicMatrixMultiplication algorithm behaves when A and B are well approximated by low-rank matrices. Since a low-rank matrix or a matrix that is well approximated by a low-rank matrix is a matrix whose rows and columns contain much redundant information in terms of the subspaces they span, it is plausible that if the range of B overlaps appropriately with the domain of A, then we can get a good approximation to AB by carefully sampling a small number c of appropriately rescaled rank-one approximations to AB. Theorem 1 shows that if the $\{p_i\}_{i=1}^n$ are chosen judiciously, then this is the case and Figure 3 illustrates this.

We emphasize that in the case of sampling with nonuniform probabilities our sampling can be viewed as a two-pass algorithm; in the first pass the algorithm reads the matrix, it then decides which columns and rows to keep, and then in the second pass it extracts these columns and rows. In certain applications, two passes through the matrix are not possible and only one pass is allowed [14]. In these cases, we can still perform uniform sampling; in this case, if column-row pairs are all approximately the same size, i.e., $|A^{(k)}||B_{(k)}|$ is close to its mean value (more precisely, if there exists some positive constant $\beta \leq 1$ such that $\forall k \ |A^{(k)}||B_{(k)}| \leq \frac{1}{\beta n} \sum_{k'=1}^{n} |A^{(k')}||B_{(k')}|$), then the uniform probabilities are nearly optimal and we can sample uniformly with a small β-dependent loss in accuracy.

Note that although larger columns and rows get picked more often, the scaling is such that their weight is deemphasized in the estimator sum. One could imagine a situation when detailed information about the elements of, e.g., A may be obtained after a single pass but no information or no information except general bounds on the size of the elements may be possible for B. In this case, a set of sampling probabilities other than those discussed in section 4 may be appropriate. See Table 1 for a summary of the results for different probability distributions; these results are proven in section A.3

The ElementwiseMatrixMultiplication algorithm has been presented for completeness and because in some applications its use may be more appropriate than the use of the BasicMatrixMultiplication algorithm. It is worth emphasizing that the ElementwiseMatrixMultiplication algorithm achieves its spectral norm bound since its sampling procedure may be viewed as adding a carefully constructed random perturbation to every element of the original matrix; see [2, 1] for a nice discussion of these ideas.

TABLE 1
Summary of results for different probability distributions.

	$\mathbf{E}\left[\, \|AB-CR\|_F \,\right] \leq$	w.h.p. $\|AB-CR\|_F \leq$	Comments and restrictions
$p_k \geq \frac{\beta\left\|A^{(k)}\right\|\|B_{(k)}\|}{\sum_{k'}\left\|A^{(k')}\right\|\left\|B_{(k')}\right\|}$	$\frac{1}{\sqrt{\beta c}}\|A\|_F\,\|B\|_F$	$\frac{\eta}{\sqrt{\beta c}}\|A\|_F\,\|B\|_F$	$\eta = 1+\sqrt{\frac{8}{\beta}\log\left(\frac{1}{\delta}\right)}$
$p_k \geq \frac{\beta\left\|A^{(k)}\right\|^2}{\|A\|_F^2}$	$\frac{1}{\sqrt{\beta c}}\|A\|_F\,\|B\|_F$	$\frac{\eta}{\sqrt{\beta c}}\|A\|_F\,\|B\|_F$	$\eta = 1+\frac{\|A\|_F}{\|B\|_F}\mathcal{M}\sqrt{\frac{8}{\beta}\log\left(\frac{1}{\delta}\right)}$ $\mathcal{M} = \max_\alpha \frac{\|B_{(\alpha)}\|}{\|A^{(\alpha)}\|}$
$p_k \geq \frac{\beta\|B_{(k)}\|^2}{\|B\|_F^2}$	$\frac{1}{\sqrt{\beta c}}\|A\|_F\,\|B\|_F$	$\frac{\eta}{\sqrt{\beta c}}\|A\|_F\,\|B\|_F$	$\eta = 1+\frac{\|B\|_F}{\|A\|_F}\mathcal{M}\sqrt{\frac{8}{\beta}\log\left(\frac{1}{\delta}\right)}$ $\mathcal{M} = \max_\alpha \frac{\|A^{(\alpha)}\|}{\|B_{(\alpha)}\|}$
$p_k \geq \frac{\beta\left\|A^{(k)}\right\|}{\sum_{k'=1}^{n}\left\|A^{(k')}\right\|}$	$\frac{1}{\sqrt{\beta c}}\|A\|_F\sqrt{n}\mathcal{M}$	$\frac{\eta}{\sqrt{\beta c}}\|A\|_F\sqrt{n}\mathcal{M}$	$\eta = 1+\sqrt{\frac{8}{\beta}\log\left(\frac{1}{\delta}\right)}$ $\mathcal{M} = \max_\alpha \left\|B_{(\alpha)}\right\|$
$p_k \geq \frac{\beta\|B_{(k)}\|}{\sum_{k'=1}^{n}\left\|B_{(k')}\right\|}$	$\frac{1}{\sqrt{\beta c}}\sqrt{n}\mathcal{M}\,\|B\|_F$	$\frac{\eta}{\sqrt{\beta c}}\sqrt{n}\mathcal{M}\,\|B\|_F$	$\eta = 1+\sqrt{\frac{8}{\beta}\log\left(\frac{1}{\delta}\right)}$ $\mathcal{M} = \max_\alpha \left\|A^{(\alpha)}\right\|$
$p_k \geq \frac{\beta\left\|A^{(k)}\right\|\|B_{(k)}\|}{\|A\|_F\|B\|_F}$	$\frac{1}{\sqrt{\beta c}}\|A\|_F\,\|B\|_F$	$\frac{\eta}{\sqrt{\beta c}}\|A\|_F\,\|B\|_F$	$\eta = 1+\sqrt{\frac{8}{\beta}\log\left(\frac{1}{\delta}\right)}$
$p_k = \frac{1}{n}$	See Lemma 11.	See Lemma 11.	See Lemma 11.

Recent work has focused on establishing lower bounds on the number of queries a sampling algorithm is required to perform in order to approximate a given function accurately with low probability of error; see, e.g., [4]. See also [24, 27] for recent related work.

Appendix. Further analysis of the basic matrix multiplication algorithm. In this section we provide further analysis of the BASICMATRIXMULTIPLICATION algorithm. In section A.1 we consider approximating the product of more than two matrices by a similar sampling process. Then, in section A.2 we examine element-wise error bounds for the algorithm, and in section A.3 we consider error bounds for probability distributions which are not nearly optimal in the sense of section 4.4.

A.1. Approximating the product of more than two matrices. In this section we consider the task of approximating the product of three or more matrices using the ideas of the BASICMATRIXMULTIPLICATION algorithm of section 4. For simplicity our exposition will be restricted to the case of approximating the product ABC of three matrices. Recall that given matrices $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$, $C \in \mathbb{R}^{p\times q}$, the product ABC may be written as

$$ABC = \sum_{s=1}^{n}\sum_{t=1}^{p} A^{(s)}B_{st}C_{(t)}. \tag{38}$$

One possible way of extending the ideas of section 4.1 is the following. Randomly choose $i_s \in \{1,\ldots,n\}$ independently and with replacement c_1 times according to a probability distribution $\{p_i\}_{i=1}^n$ and randomly choose $j_t \in \{1,\ldots,p\}$ independently

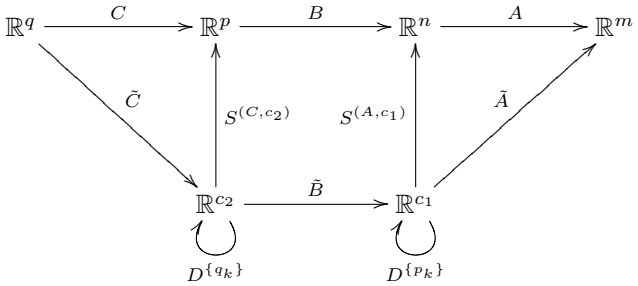


FIG. 5. *Diagram for the algorithm to approximately multiply three matrices.*

and with replacement c_2 times according to a probability distribution $\{q_j\}_{j=1}^{p}$. Then form the matrix $\tilde{A} \in \mathbb{R}^{m\times c_1}$ with columns $\tilde{A}^{(s)} = A^{(i_s)}/\sqrt{c_1 p_{i_s}}$, the matrix $\tilde{B} \in \mathbb{R}^{c_1\times c_2}$ with elements $\tilde{B}_{st} = B_{i_s j_t}/\sqrt{c_1 c_2 p_{i_s} q_{j_t}}$, and the matrix $\tilde{C} \in \mathbb{R}^{c_2\times q}$ with rows $\tilde{C}_{(t)} = C_{(j_t)}/\sqrt{c_2 q_{j_t}}$ so that

$$\tilde{A}\tilde{B}\tilde{C} = \sum_{s=1}^{c_1}\sum_{t=1}^{c_2} \frac{A^{(i_s)} B_{i_s j_t} C_{(j_t)}}{c_1 c_2 p_{i_s} q_{j_t}}.$$

Figure 5 presents a diagram illustrating the action of the algorithm just described to approximate the product of three matrices. One could then define sampling matrices $S^{(A,c_1)}$ and $S^{(C,c_2)}$ and diagonal rescaling matrices $D^{\{p_k\}}$ and $D^{\{q_k\}}$ in a manner analogous to that of section 4.1 and as indicated in Figure 5. Then $\tilde{A}\tilde{B}\tilde{C} = AS^{(A,c_1)}{D^{\{p_k\}}}^2 {S^{(A,c_1)}}^T BS^{(C,c_2)}{D^{\{q_k\}}}^2 {S^{(C,c_2)}}^T C \approx ABC$. An intuitively appealing aspect of this algorithm is that the product ABC is shown as C, B, and then A operating between the high-dimensional $\mathbb{R}^q$ and $\mathbb{R}^m$ via the high-dimensional $\mathbb{R}^p$ and $\mathbb{R}^n$; this is approximated by $\tilde{A}\tilde{B}\tilde{C}$, which acts between $\mathbb{R}^q$ and $\mathbb{R}^m$ via the low-dimensional subspaces $\mathbb{R}^{c_2}$ and $\mathbb{R}^{c_1}$. One difficulty with this algorithm is that its analysis is quite complicated due to the correlation in the nonindependent sampling of the elements of the matrix B.

A second way of extending the ideas of section 4.1 is the following. Randomly choose $(i_s, j_t) \in \{1,\ldots,n\}\times\{1,\ldots,p\}$ independently and with replacement c times according to a probability distribution $\{p_{kl}\}_{(k,l)=1}^{n,p}$. This corresponds to sampling c terms from the sum (38). Then define

$$P = \sum_{u\equiv(s,t)=1}^{c} \frac{1}{c p_{k_s l_t}} A^{(k_s)} B_{k_s l_t} C_{(l_t)},$$

where the summation is a single sum over the c pairs $(k_s, l_t) \in \{1,\ldots,n\}\times\{1,\ldots,p\}$ chosen by the algorithm. In this second algorithm the subspace interpretation of the first algorithm is lost but the analysis simplifies considerably. Using ideas similar to those in section 4 we can prove the following lemma about this algorithm.

LEMMA 6. *Given matrices $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$, $C \in \mathbb{R}^{p\times q}$, construct an approximation P to the product ABC by sampling as described in the second algorithm above with probabilities $\{p_{k,l}\}_{(k,l)=1}^{np}$. Then, for every i, j we have that $\mathbf{E}[(P)_{ij}] = (ABC)_{ij}$ and that*

$$\mathbf{Var}\left[\,(P)_{ij}\,\right] = \frac{1}{c}\sum_{k=1}^{n}\sum_{l=1}^{p}\frac{1}{p_{kl}} A_{ik}^2 B_{kl}^2 C_{lj}^2 - \frac{1}{c}\,(ABC)_{ij}^2\,.$$

In addition,

$$\mathbf{E}\left[\,\|ABC-P\|_F^2\right]=\frac{1}{c}\sum_{k=1}^{n}\sum_{l=1}^{p}\frac{1}{p_{kl}}\left|A^{(k)}\right|^2B_{kl}^2\left|C_{(l)}\right|^2-\frac{1}{c}\,\|ABC\|_F^2$$

and the probabilities

$$p_{kl}=\frac{\left|A^{(k)}\right|\,|B_{kl}|\,\left|C_{(l)}\right|}{\sum_{k'}\sum_{l'}\,\left|A^{(k')}\right|\,|B_{k'l'}|\,\left|C_{(l')}\right|}$$

minimize $\mathbf{E}\left[\,\|ABC-P\|_F^2\,\right]$

Proof. The proof is similar to those of Lemmas 3 and 4. □

As in section 4.4 we will define probabilities $\{p_{kl}\}$ to be nearly optimal if

$$p_{kl}\geq\beta\frac{\left|A^{(k)}\right|\,|B_{kl}|\,\left|C_{(l)}\right|}{\sum_{k'}\sum_{l'}\,\left|A^{(k')}\right|\,|B_{k'l'}|\,\left|C_{(l')}\right|}$$

for some $\beta\leq 1$. If sampling is performed with these probabilities, one can show that

$$\mathbf{E}\left[\,\|ABC-P\|_F^2\right]\leq\frac{1}{c\beta}\sum_k\sum_l\left|A^{(k)}\right|\,|B_{kl}|\,\left|C_{(l)}\right|,$$

and a similar result can be shown to hold with high probability.

Unfortunately, computing the optimal probabilities in the general case is not pass-efficient since it would require $O(np)$ additional space and time. This situation would be relatively worse if one wanted to compute the product of more than three matrices, rendering this method uncompetitive with the exact algorithm. On the other hand, if the matrices are known to have a special structure or if the data are presented in a more specialized format, then this algorithm may be useful. For example, if it is known that none of the elements of B are too big, i.e., that the elements of B are such that there exists a ξ_B such that $\forall i,j$ we have that $B_{ij}\leq\xi_B\,\|B\|_F^2/np$, then there will exist a set of probabilities that are nearly optimal that do not depend on B and that can be computed efficiently.

A.2. Elementwise error bounds. In this section we provide elementwise error bounds on $|(AB)_{ij}-(CR)_{ij}|$ for the BasicMatrixMultiplication algorithm for two different probability distributions. We have the following lemma.

Lemma 7. *Suppose* $A\in\mathbb{R}^{m\times n}$, $B\in\mathbb{R}^{n\times p}$, $c\in\mathbb{Z}^+$ *such that* $1\leq c\leq n$, *and* $\{p_i\}_{i=1}^n$ *are such that* $p_i\geq 0$ *and* $\sum_{i=1}^n p_i=1$. *Let* M *be such that* $|A_{ij}|\leq M$ *and* $|B_{ij}|\leq M$ *for every appropriate* i,j. *Construct* C *and* R *with the* BasicMatrixMultiplication *algorithm, and let* CR *be an approximation to* AB. *If* $p_k=1/n$ *for every* k, *then for every* $\delta>0$ *with probability at least* $1-\delta$

$$|(AB)_{ij}-(CR)_{ij}|<\frac{nM^2}{\sqrt{c}}\sqrt{8\ln(2mp/\delta)}\quad\forall i,j. \tag{39}$$

If $p_k\geq\frac{\beta\left|A^{(k)}\right|\left|B_{(k)}\right|}{\sum_{k'=1}^n\left|A^{(k')}\right|\left|B_{(k')}\right|}$ *for some positive constant* $\beta\leq 1$, *then for every* $\delta>0$ *with probability at least* $1-\delta$

$$|(AB)_{ij}-(CR)_{ij}|<\frac{n\sqrt{mp}M^2}{\sqrt{\beta c}}\sqrt{(8/\beta)\ln(2mp/\delta)}\quad\forall i,j. \tag{40}$$

Proof. Let us first consider the case of uniform sampling probabilities, i.e., when $p_k = 1/n$. First, fix attention on one particular $(i,j) \in (\{1,\ldots,m\},\{1,\ldots,p\})$. Define $X_t^{(ij)} = \left(\frac{A^{(i_t)}B_{(i_t)}}{cp_{i_t}}\right)_{ij} = \frac{A_{ii_t}B_{i_tj}}{cp_{i_t}}$. From Lemma 3 we see that $\mathbf{E}\left[X_t^{(ij)}\right] = \frac{1}{c}(AB)_{ij}$. Define $Y_t^{(ij)} = X_t^{(ij)} - \frac{1}{c}(AB)_{ij}$, $t = 1,\ldots,c$, and note that the Y_t's are independent random variables with $\mathbf{E}\left[Y_t^{(ij)}\right] = 0$ for every $t = 1,\ldots,c$. In addition,

$$\begin{aligned}\left|Y_t^{(ij)}\right| &\le \left|\frac{A_{ii_t}B_{i_tj}}{cp_{i_t}}\right| + \left|\frac{1}{c}(AB)_{ij}\right| \\ &\le \left|\frac{A_{ii_t}B_{i_tj}}{cp_{i_t}}\right| + \frac{nM^2}{c} \qquad (41)\\ &\le \frac{2nM^2}{c}. \qquad (42)\end{aligned}$$

Inequality (42) follows since for the uniform probabilities $|\frac{A_{ii_t}B_{i_tj}}{cp_{i_t}}| \le \frac{nM^2}{c}$. By combining the upper and lower bounds provided by (42) with Hoeffding's inequality, we have that for any $t > 0$

$$\mathbf{Pr}\left[\left|\sum_{t=1}^{c} Y_t^{(ij)}\right| \ge ct\right] \le 2\exp\left(-\frac{2c^2t^2}{\sum_{i=1}^{c}\left(4nM^2/c\right)^2}\right) = 2\exp\left(-\frac{c^3t^2}{8n^2M^4}\right). \qquad (43)$$

Define the event $\mathcal{E}_{ij}$ to be $|\sum_{t=1}^{c} Y_t^{(ij)}| \ge ct$ and the event $\mathcal{E} = \bigcup_{i=1}^{m}\bigcup_{j=1}^{p}\mathcal{E}_{ij}$. If we then let $t = nM^2\frac{2\sqrt{2}}{c^{3/2}}\sqrt{\ln(2mp/\delta)}$, then by (43) we have that $\mathbf{Pr}[\mathcal{E}_{ij}] \le \frac{\delta}{mp}$. Thus, (39) then follows since

$$\mathbf{Pr}[\mathcal{E}] \le \sum_{i=1}^{m}\sum_{j=1}^{p}\mathbf{Pr}[\mathcal{E}_{ij}] \le \sum_{ij}\delta/mp = \delta.$$

When applied to the nonuniform probabilities $p_k \ge \frac{\beta|A^{(k)}||B_{(k)}|}{\sum_{k'=1}^{n}|A^{(k')}||B_{(k')}|}$ a similar line of reasoning establishes (40). The key step is to note that when using these probabilities we have that

$$\left|\frac{A_{ii_t}B_{i_tj}}{cp_{i_t}}\right| \le \left|\frac{A_{ii_t}B_{i_tj}}{c\beta\left|A^{(k)}\right|\left|B_{(k)}\right|}\sum_{k'=1}^{n}\left|A^{(k')}\right|\left|B_{(k')}\right|\right| \le \left|\frac{n\sqrt{mp}}{c\beta}M^2\right|. \qquad (44)$$

Since $nM^2/c \le n\sqrt{mp}M^2/(c\beta)$ this, when combined with (41), implies that

$$\left|Y_t^{(ij)}\right| \le \frac{2n\sqrt{mp}M^2}{c\beta}, \qquad (45)$$

which provides the upper and lower bounds on the random variable required to apply Hoeffding's inequality. □

When the uniform probabilities are used

$$\|AB - CR\|_F^2 = \sum_{ij}|(AB)_{ij} - (CR)_{ij}|^2 \le \frac{mn^2pM^4}{c}8\log(2mp/\delta)$$

holds with probability greater than $1-\delta$. The difference between this result and the result of Theorem 1 or its variants such as Lemma 11 is that Lemma 7 guarantees that

every element of the approximation will have small additive error, while Theorem 1 provides a tighter Frobenius norm bound but not elementwise guarantees.

It may seem counterintuitive that by sampling with respect to the optimal probabilities of section 4 the bound of (40) is worse than that of (39) by a factor of $\sqrt{mp/\beta}$. (Relatedly, when the nonuniform probabilities of Lemma 7 are used, we have that

$$\|AB - CR\|_F^2 \leq \frac{m^2 n^2 p^2 M^4}{\beta^2 c} 8 \log(2mp/\delta)$$

with probability greater than $1 - \delta$.) The reason for this is that the optimal probabilities are optimal with respect to minimizing $\mathbf{E}\left[\, \|AB - CR\|_F^2 \,\right]$, in which case elements corresponding to smaller columns and rows contribute relatively little. On the other hand, the two statements of Lemma 7 are required to hold for every i and j. Thus (whether or not the uniform probabilities are nearly optimal) because the optimal sampling probabilities bias toward elements corresponding to larger columns and rows an extra factor of $\sqrt{mp}$ is needed.

A.3. Analysis of the algorithm for nonnearly optimal probabilities. Note that the nearly optimal probabilities (7) use information from both matrices A and B in a particular form. In some cases, such detailed information about both matrices may not be available. Thus, we present results for the BasicMatrixMultiplication algorithm for several other sets of probabilities. See Table 1 in section 6 for a summary of these results.

In the first case, to estimate the product AB one could use the probabilities (46) which use information from the matrix A only. In this case $\|AB - CR\|_F$ can still be shown to be small in expectation, and under an additional assumption the expectation can be removed and the corresponding result can be shown to hold with high probability.

Lemma 8. *Suppose $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{n \times p}$, $c \in \mathbb{Z}^+$ such that $1 \leq c \leq n$, and $\{p_i\}_{i=1}^n$ are such that $\sum_{i=1}^n p_i = 1$ and such that*

$$p_k \geq \frac{\beta \left| A^{(k)} \right|^2}{\|A\|_F^2} \tag{46}$$

for some positive constant $\beta \leq 1$. Construct C and R with the BasicMatrixMultiplication *algorithm, and let CR be an approximation to AB. Then*

$$\mathbf{E}\left[\, \|AB - CR\|_F^2 \right] \leq \frac{1}{\beta c} \|A\|_F^2 \, \|B\|_F^2 . \tag{47}$$

Furthermore, let $\mathcal{M} = \max_\alpha \frac{|B_{(\alpha)}|}{\left|A^{(\alpha)}\right|}$, let $\delta \in (0,1)$, and let $\eta = 1 + \frac{\|A\|_F}{\|B\|_F} \mathcal{M} \sqrt{(8/\beta) \log(1/\delta)}$. Then with probability at least $1 - \delta$,

$$\|AB - CR\|_F^2 \leq \frac{\eta^2}{\beta c} \|A\|_F^2 \, \|B\|_F^2 . \tag{48}$$

Proof. The proof is similar to that of Theorem 1 except that the indicated probabilities are used. ∎

Alternatively, to estimate the product AB one could use the probabilities (49) which also use information from the matrix A only, but in a different form than the probabilities (46). In this case, under an additional assumption $\|AB - CR\|_F$ can still be shown to be small both in expectation and with high probability.

LEMMA 9. *Suppose* $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$, $c \in \mathbb{Z}^+$ *such that* $1 \le c \le n$, *and* $\{p_i\}_{i=1}^n$ *are such that* $\sum_{i=1}^n p_i = 1$ *and such that*

$$p_k \ge \frac{\beta \left|A^{(k)}\right|}{\sum_{k'=1}^n \left|A^{(k')}\right|} \tag{49}$$

for some positive constant $\beta \le 1$. *Let* $\mathcal{M} = \max_\alpha |B_{(\alpha)}|$. *Construct* C *and* R *with the* BASICMATRIXMULTIPLICATION *algorithm, and let* CR *be an approximation to* AB. *Then*

$$\mathbf{E}\left[\, \|AB - CR\|_F^2 \right] \le \frac{1}{\beta c} \|A\|_F^2 \, n\mathcal{M}^2. \tag{50}$$

Furthermore, let $\delta \in (0,1)$ *and* $\eta = 1 + \sqrt{(8/\beta)\log(1/\delta)}$. *Then with probability at least* $1-\delta$,

$$\|AB - CR\|_F^2 \le \frac{\eta^2}{\beta c} \|A\|_F^2 \, n\mathcal{M}^2. \tag{51}$$

Proof. The proof is similar to that of Theorem 1 except that the indicated probabilities are used. □

The probabilities (46) and (49) depend on only the lengths of the columns of A. Results similar to those of the previous two lemmas hold if the probabilities depend on the rows of B rather than the columns of A; see Table 1.

Alternatively, to estimate the product of AB one could use the probabilities (52); interestingly, although the probabilities differ from those of (7) we are able to derive the same bounds as those of Theorem 1 without additional assumptions.

LEMMA 10. *Suppose* $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$, $c \in \mathbb{Z}^+$ *such that* $1 \le c \le n$, *and* $\{p_i\}_{i=1}^n$ *are such that* $\sum_{i=1}^n p_i = 1$ *and such that*

$$p_k \ge \frac{\beta \left|A^{(k)}\right| \left|B_{(k)}\right|}{\|A\|_F \|B\|_F} \tag{52}$$

for some positive constant $\beta \le 1$. *Construct* C *and* R *with the* BASICMATRIXMULTIPLICATION *algorithm, and let* CR *be an approximation to* AB. *Then*

$$\mathbf{E}\left[\, \|AB - CR\|_F^2 \right] \le \frac{1}{\beta c} \|A\|_F^2 \, \|B\|_F^2 . \tag{53}$$

Furthermore, let $\delta \in (0,1)$ *and* $\eta = 1 + \sqrt{(8/\beta)\log(1/\delta)}$. *Then with probability at least* $1-\delta$,

$$\|AB - CR\|_F^2 \le \frac{\eta^2}{\beta c} \|A\|_F^2 \, \|B\|_F^2 . \tag{54}$$

Proof. The proof is similar to that of Theorem 1 except that the indicated probabilities are used. □

Of course one could estimate the product AB using the uniform probabilities (55). In this case for simplicity we consider bounding $\|AB - CR\|_F$ directly.

LEMMA 11. *Suppose* $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$, $c \in \mathbb{Z}^+$ *such that* $1 \le c \le n$, *and* $\{p_i\}_{i=1}^n$ *are such that*

$$p_k = \frac{1}{n}. \tag{55}$$

Construct C and R with the BasicMatrixMultiplication *algorithm, and let CR be an approximation to AB. Then*

$$\mathbf{E}\left[\,\|AB-CR\|_F\right] \leq \sqrt{\frac{n}{c}}\left(\sum_{k=1}^{n}\left|A^{(k)}\right|^2\left|B_{(k)}\right|^2\right)^{1/2}. \tag{56}$$

Furthermore, let $\delta \in (0,1)$ and $\gamma = \frac{n}{\sqrt{c}}\sqrt{8\log(1/\delta)}\max_\alpha \left|A^{(\alpha)}\right|\left|B_{(\alpha)}\right|$. Then with probability at least $1-\delta$,

$$\|AB-CR\|_F \leq \sqrt{\frac{n}{c}}\left(\sum_{k=1}^{n}\left|A^{(k)}\right|^2\left|B_{(k)}\right|^2\right)^{1/2} + \gamma. \tag{57}$$

Proof. The proof is similar to that of Theorem 1 except that the indicated probabilities are used. ∎

Acknowledgments. We would like to thank Dimitris Achlioptas for bringing to our attention the results of [21] and the National Science Foundation for partial support of this work. In addition, we would like to thank an anonymous reviewer for providing a careful reading of this paper, for making numerous constructive suggestions, and for bringing [28] to our attention.

REFERENCES

[1] D. Achlioptas and F. McSherry, *Fast computation of low rank matrix approximations*, J. ACM, to appear.

[2] D. Achlioptas and F. McSherry, *Fast computation of low rank matrix approximations*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, 2001, pp. 611–618.

[3] N. Alon, Y. Matias, and M. Szegedy, *The space complexity of approximating the frequency moments*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing, 1996, pp. 20–29.

[4] Z. Bar-Yossef, *Sampling lower bounds via information theory*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, 2003, pp. 335–344.

[5] D. Barbara, C. Faloutsos, J. Hellerstein, Y. Ioannidis, H. V. Jagadish, T. Johnson, R. Ng, V. Poosala, K. Ross, and K. C. Sevcik, *The New Jersey data reduction report*, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 1997.

[6] R. Bhatia, *Matrix Analysis*, Springer-Verlag, New York, 1997.

[7] E. Cohen and D. D. Lewis, *Approximating matrix multiplication for pattern recognition tasks*, J. Algorithms, 30 (1999), pp. 211–252.

[8] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, J. Symbolic Comput., 9 (1990), pp. 251–280.

[9] P. Drineas and R. Kannan, *Fast Monte-Carlo algorithms for approximate matrix multiplication*, in Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science, 2001, pp. 452–459.

[10] P. Drineas and R. Kannan, *Pass efficient algorithms for approximating large matrices*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, 2003, pp. 223–232.

[11] P. Drineas, R. Kannan, and M. W. Mahoney, *Fast Monte Carlo algorithms for matrices* II: *Computing a low-rank approximation to a matrix*, SIAM J. Comput., 36 (2006), pp. 158–183.

[12] P. Drineas, R. Kannan, and M. W. Mahoney, *Fast Monte Carlo algorithms for matrices* III: *Computing a compressed approximate matrix decomposition*, SIAM J. Comput., 36 (2006), pp. 184–206.

[13] P. Drineas, R. Kannan, and M. W. Mahoney, *Fast Monte Carlo Algorithms for Matrices* I: *Approximating Matrix Multiplication*, Tech. Report YALEU/DCS/TR-1269, Department of Computer Science, Yale University, New Haven, CT, 2004.

[14] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan, *An approximate L^1-difference algorithm for massive data sets*, in Proceedings of the 40th Annual IEEE Symposium on the Foundations of Computer Science, 1999, pp. 501–511.

[15] A. Frieze, R. Kannan, and S. Vempala, *Fast Monte-Carlo algorithms for finding low-rank approximations*, in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, 1998, pp. 370–378.

[16] Z. Füredi and J. Komlós, *The eigenvalues of random symmetric matrices*, Combinatorica, 1 (1981), pp. 233–241.

[17] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss, *Fast, small-space algorithms for approximate histogram maintenance*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, 2002, pp. 389–398.

[18] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, 1989.

[19] M. R. Henzinger, P. Raghavan, and S. Rajagopalan, *Computing on Data Streams*, Tech. Report 1998-011, Digital Systems Research Center, Palo Alto, CA, 1998.

[20] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, New York, 1985.

[21] M. Krivelevich and V. H. Vu, *On the Concentration of Eigenvalues of Random Symmetric Matrices*, Tech. Report MSR-TR-2000-60, Microsoft Research, Redmond, WA, 2000.

[22] C. McDiarmid, *On the method of bounded differences*, in Surveys in Combinatorics, 1989, J. Siemons, ed., London Math. Soc. Lecture Notes Ser., Cambridge University Press, Cambridge, UK, 1989, pp. 148–188.

[23] J. I. Munro and M. S. Paterson, *Selection and sorting with limited storage*, in Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science, 1978, pp. 253–258.

[24] M. Rudelson and R. Vershynin, *Approximation of Matrices*, manuscript.

[25] G. W. Stewart and J. G. Sun, *Matrix Perturbation Theory*, Academic Press, New York, 1990.

[26] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., 14 (1969), pp. 354–356.

[27] R. Vershynin, *Coordinate Restrictions of Linear Operators in l_2^n*, manuscript; available online from http://arxiv.org/abs/math/0011232.

[28] J. S. Vitter, *Random sampling with a reservoir*, ACM Trans. Math. Softw., 11 (1985), pp. 37–57.

SIAM J. COMPUT.
Vol. 36, No. 1, pp. 158–183

FAST MONTE CARLO ALGORITHMS FOR MATRICES II: COMPUTING A LOW-RANK APPROXIMATION TO A MATRIX*

PETROS DRINEAS†, RAVI KANNAN‡, AND MICHAEL W. MAHONEY§

Abstract. In many applications, the data consist of (or may be naturally formulated as) an $m \times n$ matrix A. It is often of interest to find a low-rank approximation to A, i.e., an approximation D to the matrix A of rank not greater than a specified rank k, where k is much smaller than m and n. Methods such as the singular value decomposition (SVD) may be used to find an approximation to A which is the best in a well-defined sense. These methods require memory and time which are superlinear in m and n; for many applications in which the data sets are very large this is prohibitive. Two simple and intuitive algorithms are presented which, when given an $m \times n$ matrix A, compute a description of a low-rank approximation D^* to A, and which are qualitatively faster than the SVD. Both algorithms have provable bounds for the error matrix $A - D^*$. For any matrix X, let $\|X\|_F$ and $\|X\|_2$ denote its Frobenius norm and its spectral norm, respectively. In the first algorithm, c columns of A are randomly chosen. If the $m \times c$ matrix C consists of those c columns of A (after appropriate rescaling), then it is shown that from $C^T C$ approximations to the top singular values and corresponding singular vectors may be computed. From the computed singular vectors a description D^* of the matrix A may be computed such that $\text{rank}(D^*) \leq k$ and such that

$$\|A - D^*\|_\xi^2 \leq \min_{D:\text{rank}(D)\leq k} \|A - D\|_\xi^2 + poly(k, 1/c)\,\|A\|_F^2$$

holds with high probability for both $\xi = 2, F$. This algorithm may be implemented without storing the matrix A in random access memory (RAM), provided it can make two passes over the matrix stored in external memory and use $O(cm + c^2)$ additional RAM. The second algorithm is similar except that it further approximates the matrix C by randomly sampling r rows of C to form a $r \times c$ matrix W. Thus, it has additional error, but it can be implemented in three passes over the matrix using only constant additional RAM. To achieve an additional error (beyond the best rank k approximation) that is at most $\epsilon\|A\|_F^2$, both algorithms take time which is polynomial in k, $1/\epsilon$, and $\log(1/\delta)$, where $\delta > 0$ is a failure probability; the first takes time linear in $\max(m, n)$ and the second takes time independent of m and n. Our bounds improve previously published results with respect to the rank parameter k for both the Frobenius and spectral norms. In addition, the proofs for the error bounds use a novel method that makes important use of matrix perturbation theory. The probability distribution over columns of A and the rescaling are crucial features of the algorithms which must be chosen judiciously.

Key words. randomized algorithms, Monte Carlo methods, massive data sets, singular value decomposition

AMS subject classification. 68W20

DOI. 10.1137/S0097539704442696

1. Introduction.

We are interested in developing and analyzing fast Monte Carlo algorithms for performing useful computations on large matrices. In this paper we consider the singular value decomposition (SVD); in two related papers we consider matrix multiplication and a new method for computing a compressed approximate decomposition of a large matrix [13, 14]. Since such computations generally require

*Received by the editors April 5, 2004; accepted for publication (in revised form) November 17, 2005; published electronically May 26, 2006. The technical report version of this paper appeared as *Fast Monte Carlo Algorithms for Matrices* II: *Computing a Low-Rank Approximation to a Matrix*, by P. Drineas, R. Kannan, and M. W. Mahoney [15].

http://www.siam.org/journals/sicomp/36-1/44269.html

†Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180 (drinep@cs.rpi.edu).

‡Department of Computer Science, Yale University, New Haven, CT 06520 (kannan@cs.yale.edu).
This author was supported in part by a grant from the NSF.
§Department of Mathematics, Yale University, New Haven, CT 06520 (mahoney@cs.yale.edu).

time which is superlinear in the number of nonzero elements of the matrix, we expect our algorithms to be useful in many applications where data sets are modeled by matrices and are extremely large. In all these cases, we assume that the input matrices are prohibitively large to store in random access memory (RAM) and thus that only external memory storage is possible. Our algorithms will be allowed to read the matrices a few, e.g., one, two or three, times and keep a small randomly chosen and rapidly computable "sketch" of the matrices in RAM; computations will then be performed on this "sketch." We will work within the framework of the pass-efficient computational model, in which the scarce computational resources are the number of passes over the data, the additional RAM space required, and the additional time required [13, 12].

In many applications, the data consist of (or may be naturally formulated as) an $m \times n$ matrix A which is either low-rank or is well approximated by a low-rank matrix [7, 9, 25, 3, 26, 28, 29, 24, 22]. In these application areas, e.g., latent semantic indexing, DNA microarray analysis, facial and object recognition, and web search models, the data may consist of m points in $\mathbb{R}^n$. Let $A \in \mathbb{R}^{m \times n}$ be the matrix with these points as rows. Two methods for dealing with such high-dimensional data are the SVD (and the related principal components analysis) and multidimensional scaling [18, 23]. Thus, it is often of interest to find a low-rank approximation to A, i.e., an approximation D, of rank no greater than a specified rank k, to the matrix A, where k is much smaller than m and n. For example, this rank reduction is used in many applications of linear algebra and statistics as well as in image processing, lossy data compression, text analysis, and cryptography [6]. The SVD may be used to find an approximation to A which is the best in a well-defined sense [18, 19], but it requires a superlinear (in m and n) polynomial time dependence that is prohibitive for many applications in which the data sets are very large. Another method that has attracted interest recently is the traditional "random projection" method where one projects the problem into a randomly chosen low-dimensional subspace [21, 30, 20]. This dimensional reduction requires performing an operation that amounts to premultiplying the given $m \times n$ matrix A by an $s \times m$ matrix which takes time dependent in a superlinear manner on $m + n$.

In this paper we present two simple and intuitive algorithms which, when given an $m \times n$ matrix A, compute a description of a low-rank approximation D^* to A, and which are qualitatively faster than the SVD. Both algorithms have provable bounds for the error matrix $A - D^*$. For any matrix X, let $\|X\|_F$ and $\|X\|_2$ denote its Frobenius norm and its spectral norm (as defined in section 3.1), respectively. In the first algorithm, the LinearTimeSVD algorithm of section 4, c columns of A are randomly chosen. If the $m \times c$ matrix C consists of those c columns of A (after appropriate rescaling), then it is shown that from $C^T C$ approximations to the top singular values and corresponding singular vectors of A may be computed. From the computed singular vectors a description D^* of the matrix A may be computed such that $\text{rank}(D^*) \leq k$ and such that

$$\|A - D^*\|_\xi^2 \leq \min_{D:\text{rank}(D)\leq k} \|A - D\|_\xi^2 + poly(k, 1/c)\, \|A\|_F^2 \tag{1}$$

holds with high probability for each of $\xi = 2, F$. This algorithm may be implemented without storing the matrix A in RAM, provided it can make two passes over the matrix stored in external memory and use $O(cm + c^2)$ additional RAM. The second algorithm, the ConstantTimeSVD algorithm of section 5, is similar except that it further approximates the matrix C by randomly sampling r rows of C to form an $r \times c$

TABLE 1
Summary of sampling complexity.

Additional error for:	LINEARTIMESVD	CONSTANTTIMESVD	REF. [16, 17]
$\|A - D^*\|_2^2$	$1/\epsilon^2$	$1/\epsilon^4$	k^4/ϵ^3
$\|A - D^*\|_F^2$	k/ϵ^2	k^2/ϵ^4	k^4/ϵ^3

matrix W. Thus, it has additional error but can be implemented in three passes over the matrix using additional RAM that is $O(cr)$, i.e., that is constant for constant c and r. To achieve an additional error that is at most $\epsilon \|A\|_F^2$, both algorithms take time which is polynomial in k, $1/\epsilon$, and $\log(1/\delta)$, where $\delta > 0$ is a failure probability; see Table 1 for a summary of the dependence of the sampling complexity on k and ϵ. The first algorithm takes time linear in $\max(m, n)$ and the other takes time independent of m and n. Our bounds improve previously published results with respect to the rank parameter k for both the Frobenius and the spectral norms. In addition, the proofs for the error bounds use a novel method that makes important use of matrix perturbation theory. The probability distribution over columns of A and the rescaling are crucial features of the algorithms which must be chosen judiciously.

It is worth emphasizing how this work fits into recent work on computing low-rank matrix approximations. In the original work of Frieze, Kannan, and Vempala [16] (see also [17]) it was shown that by working with a randomly chosen and constant-sized submatrix of A, one could obtain bounds of the form (1) for the Frobenius norm (and thus indirectly for the spectral norm). To achieve an additional error that is at most $\epsilon \|A\|_F^2$, the size of the submatrix was a constant with respect to m and n but depended polynomially on k and $1/\epsilon$; although the submatrix was constant-sized, its construction (in particular, the construction of the sampling probabilities) required space and thus time that was linear in $m + n$. In this work, we modify the algorithm of [16] so that both the construction of *and* the computation on the constant-sized submatrix requires only constant additional space and time; thus, it fits within the framework of the pass-efficient model of data-streaming computation [12, 13]. In addition, we provide a different proof of the main result of [16] for the Frobenius norm and improve the polynomial dependence on k. Our proof method is quite different than that of [16]; it relies heavily on the approximate matrix multiplication result of [13] and [11] and it uses the Hoffman–Wielandt inequality. In addition, we provide a proof of a direct and significantly improved bound with respect to the spectral norm. Since these results are technically quite complex, we also present the corresponding proofs for both norms in the linear additional space and time framework [12, 13]. These latter results have been presented in the context of clustering applications [10], but are included here for completeness and to provide motivation and clarity for the more complex constant time results. Table 1 provides a summary of our results, for both the linear and the constant time models, and shows the number of rows and columns to be sampled sufficient to ensure, with high probability, an additional error of $\epsilon \|A\|_F^2$ in (1); see section 6 for more discussion.

In other related work, Achlioptas and McSherry have also computed low-rank approximations using somewhat different sampling techniques [2, 1]. The primary focus of their work was in introducing methods to accelerate orthogonal iteration and Lanczos iteration, which are two commonly used methods for computing low-rank approximations to a matrix. Also included in [2, 1] is a comparison of their methods with those of [10, 12, 16] and thus with the results we present here. Our algorithms

and those of [16] and [2, 1] come with mathematically rigorous guarantees of the running time and of the quality of the approximation produced. As far as we know, so-called incremental SVD algorithms, which bring as much of the data as possible into memory, compute the SVD, and then update this SVD in an incremental fashion with the remaining data, do not come with such guarantees.

In section 2 several applications areas that deal with large matrices are discussed, and in section 3 we provide a review of relevant linear algebra, the pass-efficient model, and an approximate matrix multiplication result that will be used extensively. Then, in section 4 our linear additional space and time approximation algorithm, the LINEARTIMESVD algorithm, is presented and analyzed; in section 5 our constant additional space and time approximation algorithm, the CONSTANTTIMESVD algorithm, is presented and analyzed. Finally, in section 6 a discussion and conclusion are presented.

2. Some applications. There are numerous applications in which the data are well approximated by a low-rank matrix. In this section we discuss several such applications to provide motivation for our algorithms.

2.1. Latent semantic indexing. Latent semantic indexing (LSI) is a general technique for analyzing a collection of documents which are assumed to be related [7, 9, 25]. Approaches to retrieving textual information from databases that depend on a lexical match between words in the query and words in the document can be inaccurate, both because often users want to retrieve information on the basis of conceptual content and because individual words do not in general provide reliable evidence about the conceptual topic of a document. LSI is an alternative matching method that attempts to overcome problems associated with lexical matching; it does so by assuming that there is some underlying or latent semantic structure that is partially obscured by variability in word choice and then using techniques such as SVD to remove the noise and estimate this latent structure.

Suppose that there are m documents and n terms which occur in the documents. Latent semantic structure analysis starts with a term-document matrix, e.g., a matrix $A \in \mathbb{R}^{m \times n}$, where A_{ij} is frequency of the jth term in the ith document. A topic is modeled as an n-vector of nonnegative reals summing to 1, where the jth component of a topic vector is interpreted as the frequency with which the jth term occurs in a discussion of the topic. By assumption, the number of topics that the documents are about is small relative to the number of unique terms n. It can be argued that, for a given k, finding a set of k topics which best describe the documents corresponds to keeping only the top k singular vectors of A; most of the important underlying structure in the association of terms and documents will then be kept and most of the noise or variability in word usage will be removed.

2.2. DNA microarray data. DNA microarray technology has been used to study a variety of biological processes since it permits the monitoring of the expression levels of thousands of genes under a range of experimental conditions [3, 26, 28]. Depending on the particular technology, either the absolute or the relative expression levels of m genes, which for model organisms may constitute nearly the entire genome, are probed simultaneously by a single microarray. A series of n arrays probe genome-wide expression levels in n different samples, i.e., under n different experimental conditions. The data from microarray experiments may thus be represented as a matrix $A \in \mathbb{R}^{m \times n}$, where A_{ij} represents the expression level of gene i under experimental condition j. From this matrix, both the relative expression level of the

ith gene under every condition and also the relative expression level of every gene under the jth condition may be easily extracted.

This matrix is low-rank and thus a small number of eigengenes and corresponding eigenarrays (left and right singular vectors) are sufficient to capture most of the gene expression information. Removing the rest, which correspond to noise or experimental artifacts, enables meaningful comparison of the expression levels of different genes. When processing and modeling genome-wide expression data, the SVD and its low-rank approximation provides a framework such that the mathematical variables and operations suggest assigned biological meaning, e.g., in terms of cellular regulatory processes and cellular states, that may be hidden in the original data due to experimental noise or hidden dependencies. Expression data has been used for inference tasks such as to identify genes based on coexpression, predict regulatory elements, and reverse-engineer transcription networks, but this inference is difficult with noise or dependencies.

2.3. Eigenfaces and facial recognition. Applications of SVD and low-rank approximations in computer vision include pattern estimation, image compression and restoration, and facial and object recognition, where the concept of eigenfaces has been useful [29, 24].

The goal of facial recognition is to recognize a certain face given a database of photographs of human faces under variations in lighting conditions and pose viewpoints. A common approach is to represent the database as a matrix in which the rows of the matrix are the images represented as vectors. Thus, if there are m images, each of which is of size $n \times n$, the matrix $A \in \mathbb{R}^{m \times n^2}$ represents the database of images, where A_{ij} is the jth pixel value in the ith image. Typically, $m \ll n^2$ and, although many of the singular vectors are needed for very accurate reconstruction of an image, often only a few of the singular vectors are needed to extract the major appearance characteristics of an image. The right singular vectors of the matrix A are known as eigenfaces since they are the principal components or eigenvectors of the associated correlation matrix of the set of face images. The eigenfaces are computed and they are used to project the database of photographs to a lower-dimensional space that spans the significant variations among known facial images. Then, given a new image, it is projected to the same low-dimensional space, and its position is then compared to the images in the database.

2.4. Web search model. The problem of how to extract information from the network structure of a hyperlinked environment such as the World Wide Web was considered by Kleinberg [22]. This is of interest, for example, if one wants to find web pages that are relevant to a given query and one is using a keyword-based web search program, since there is no obvious endogenous measure of an authoritative page that would favor it under a text-based ranking system.

Starting with a set of pages returned by a text-based search engine, a document is defined to be an authority if many other documents returned by the search point to it, i.e., have a hypertext link to it. A document is defined to be a hub if it points to many other documents. More generally, suppose n documents are returned by the search engine. Then, a matrix $A \in \mathbb{R}^{m \times n}$ is defined, where A_{ij} is 1 or 0 depending on whether the ith document points to the jth document. Kleinberg attempts to find two n-vectors, x and y, where x_i is the hub weight of document i and y_j is the authority weight of document j. He then argues that it is desirable to find $\max_{|x|=|y|=1} x^T A y$, where $|\cdot|$ denotes the Euclidean length, since in maximizing x, y one expects the hub weights and authority weights to be mutually consistent. This is simply the problem of

finding the singular vectors of A. Since A is large, he judiciously chooses a submatrix of A and computes only the singular vectors of it. In the case when the key word has multiple meanings, not only the top but also some of the other singular vectors with large singular values are interesting. Thus, it is of interest to find the k largest singular vectors form some small k. This is the problem we are considering, and we also find the singular vectors of a submatrix, but a randomly chosen one.

3. Review of relevant background. This section contains a review of linear algebra that will be useful throughout the paper; for more detail, see [18, 19, 27, 8] and the references therein. This section also contains a review of the pass-efficient model of data-streaming computation (which provides a framework within which our SVD results may be viewed) and a matrix multiplication result that will be used extensively in our proofs; see [11, 12, 13] for more details.

3.1. Review of linear algebra. For a vector $x \in \mathbb{R}^n$ we let x_i, $i = 1, \ldots, n$, denote the ith element of x and we let $|x| = (\sum_{i=1}^n |x_i|^2)^{1/2}$. For a matrix $A \in \mathbb{R}^{m \times n}$ we let $A^{(j)}$, $j = 1, \ldots, n$, denote the jth column of A as a column vector and $A_{(i)}$, $i = 1, \ldots, m$, denote the ith row of A as a row vector; thus, if A_{ij} denotes the (i, j)th element of A, $A_{ij} = (A^{(j)})_i = (A_{(i)})_j$. The range of an $A \in \mathbb{R}^{m \times n}$ is

$$\text{range}(A) = \{y \in \mathbb{R}^m : \ y = Ax \ \text{ for some } \ x \in \mathbb{R}^n\} = \mathbf{span}(A^{(1)}, \ldots, A^{(n)}).$$

The rank of A, $\text{rank}(A)$, is the dimension of $\text{range}(A)$ and is equal to the number of linearly independent columns of A; since this is equal to $\text{rank}(A^T)$ it also equals the number of linearly independent rows of A. The null space of A is

$$\text{null}(A) = \{x \in \mathbb{R}^n : Ax = 0\}.$$

For a matrix $A \in \mathbb{R}^{m \times n}$ we denote matrix norms by $\|A\|_\xi$, using subscripts to distinguish between various norms. Of particular interest will be the Frobenius norm, which is defined by

$$\|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n A_{ij}^2}. \tag{2}$$

If $\mathbf{Tr}\,(A)$ is the matrix trace which is the sum of the diagonal elements of A, then $\|A\|_F^2 = \mathbf{Tr}(A^T A) = \mathbf{Tr}(AA^T)$. Also of interest is the spectral norm, which is defined by

$$\|A\|_2 = \sup_{x \in \mathbb{R}^n, \ x \neq 0} \frac{|Ax|}{|x|}. \tag{3}$$

Both of these norms are submultiplicative and unitarily invariant and they are related to each other as

$$\|A\|_2 \leq \|A\|_F \leq \sqrt{n}\, \|A\|_2 .$$

Both of these norms provide a measure of the "size" of the matrix A. Note that if $A \in \mathbb{R}^{m \times n}$, then there exists an $x \in \mathbb{R}^n$ such that $|x| = 1$ and $A^T Ax = \|A\|_2^2\, x$ and that if $\{x^1, x^2, \ldots, x^n\}$ is any basis of $\mathbb{R}^n$ and if $A \in \mathbb{R}^{m \times n}$, then $\|A\|_F^2 = \sum_{i=1}^n \left|Ax^i\right|^2$.

If $A \in \mathbb{R}^{m\times n}$, then there exist orthogonal matrices $U = [u^1 u^2 \dots u^m] \in \mathbb{R}^{m\times m}$ and $V = [v^1 v^2 \dots v^n] \in \mathbb{R}^{n\times n}$ where $\{u^t\}_{t=1}^m \in \mathbb{R}^m$ and $\{v^t\}_{t=1}^n \in \mathbb{R}^n$ are such that

$$U^T A V = \Sigma = \mathbf{diag}(\sigma_1, \dots, \sigma_\rho), \tag{4}$$

where $\Sigma \in \mathbb{R}^{m\times n}$, $\rho = \min\{m, n\}$, and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_\rho \geq 0$. Equivalently,

$$A = U\Sigma V^T.$$

The three matrices U, V, and Σ constitute the SVD of A. The σ_i are the singular values of A and the vectors u^i, v^i are the ith left and the ith right singular vectors, respectively. The columns of U and V satisfy the relations $Av^i = \sigma_i u^i$ and $A^T u^i = \sigma_i v^i$. For symmetric matrices the left and right singular vectors are the same. The singular values of A are the nonnegative square roots of the eigenvalues of $A^T A$ and of AA^T; furthermore, the columns of U, i.e., the left singular vectors, are eigenvectors of AA^T and the columns of V, i.e., the right singular vectors, are eigenvectors of $A^T A$.

The SVD can reveal important information about the structure of a matrix. If we define r by $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_\rho = 0$, then $\mathrm{rank}(A) = r$, $\mathrm{null}(A) = \mathbf{span}(v^{r+1}, \dots, v^\rho)$, and $\mathrm{range}(A) = \mathbf{span}(u^1, \dots, u^r)$. If we let $U_r \in \mathbb{R}^{m\times r}$ denote the matrix consisting of the first r columns of U, $V_r \in \mathbb{R}^{r\times n}$ denote the matrix consisting of the first r columns of V, and $\Sigma_r \in \mathbb{R}^{r\times r}$ denote the principal $r \times r$ submatrix of Σ, then

$$A = U_r \Sigma_r V_r^T = \sum_{t=1}^{r} \sigma_t u^t v^{tT}. \tag{5}$$

Note that this decomposition property provides a canonical description of a matrix as a sum of r rank-one matrices of decreasing importance. If $k \leq r$ and we define

$$A_k = U_k \Sigma_k V_k^T = \sum_{t=1}^{k} \sigma_t u^t v^{tT}, \tag{6}$$

then $A_k = U_k U_k^T A = (\sum_{t=1}^k u^t u^{tT})A$ and $A_k = AV_k V_k^T = A(\sum_{t=1}^k v^t v^{tT})$, i.e., A_k is the projection of A onto the space spanned by the top k singular vectors of A. Furthermore, the distance (as measured by both $\|\cdot\|_2$ and $\|\cdot\|_F$) between A and any rank-k approximation to A is minimized by A_k, i.e.,

$$\min_{D\in\mathbb{R}^{m\times n}:\mathrm{rank}(D)\leq k} \|A - D\|_2 = \|A - A_k\|_2 = \sigma_{k+1}(A) \tag{7}$$

and

$$\min_{D\in\mathbb{R}^{m\times n}:\mathrm{rank}(D)\leq k} \|A - D\|_F^2 = \|A - A_k\|_F^2 = \sum_{t=k+1}^{r} \sigma_t^2(A). \tag{8}$$

Thus, A_k constructed from the k largest singular triplets of A is the optimal rank-k approximation to A with respect to both $\|\cdot\|_F$ and $\|\cdot\|_2$. More generally, one can also show that $\|A\|_2 = \sigma_1$ and that $\|A\|_F^2 = \sum_{i=1}^r \sigma_i^2$.

From the perturbation theory of matrices it is known that the size of the difference between two matrices can be used to bound the difference between the singular value spectrum of the two matrices [27, 8]. In particular, if $A, E \in \mathbb{R}^{m\times n}, m \geq n$, then

$$\max_{t:1\leq t\leq n} |\sigma_t(A + E) - \sigma_t(A)| \leq \|E\|_2 \tag{9}$$

and

$$\sum_{k=1}^{n}(\sigma_k(A+E)-\sigma_k(A))^2 \le \|E\|_F^2 . \tag{10}$$

The latter inequality is known as the Hoffman–Wielandt inequality.

3.2. Review of the pass-efficient model. The pass-efficient model of data-streaming computation is a computational model that is motivated by the observation that in modern computers the amount of disk storage, i.e., sequential access memory, has increased very rapidly, while RAM and computing speeds have increased at a substantially slower pace [13, 12]. In the pass-efficient model the three scarce computational resources are number of passes over the data and the additional RAM space and additional time required by the algorithm. The data are assumed to be stored on a disk, to consist of elements whose size is bounded by a constant, and to be presented to an algorithm on a read-only tape. See [13] for more details.

3.3. Review of matrix multiplication. The BasicMatrixMultiplication algorithm to approximate the product of two matrices is presented and analyzed in [13]. When this algorithm is given as input two matrices, $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$, a probability distribution $\{p_i\}_{i=1}^n$, and a number $c \le n$, it returns as output two matrices, C and R, such that $CR \approx AB$; $C \in \mathbb{R}^{m\times c}$ is a matrix whose columns are c randomly chosen columns of A (suitably rescaled) and $R \in \mathbb{R}^{c\times p}$ is a matrix whose rows are the c corresponding rows of B (also suitably rescaled). An important aspect of this algorithm is the probability distribution $\{p_i\}_{i=1}^n$ used to choose column-row pairs. Although one could always use a uniform distribution, superior results are obtained if the probabilities are chosen judiciously. In particular, a set of sampling probabilities $\{p_i\}_{i=1}^n$ are *nearly optimal probabilities* if they are of the form (11) and are the *optimal probabilities* (with respect to approximating the product AB) if they are of the form (11) with $\beta = 1$. In [13] we prove the following theorem.

Theorem 1. *Suppose $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$, $c \in \mathbb{Z}^+$ such that $1 \le c \le n$, and $\{p_i\}_{i=1}^n$ are such that $p_i \ge 0$, $\sum_{i=1}^n p_i = 1$ and such that for some positive constant $\beta \le 1$*

$$p_k \ge \frac{\beta \left|A^{(k)}\right| \left|B_{(k)}\right|}{\sum_{k'=1}^{n} \left|A^{(k')}\right| \left|B_{(k')}\right|}. \tag{11}$$

Construct C and R with the BasicMatrixMultiplication *algorithm of* [13] *and let CR be an approximation to AB. Then*

$$\mathbf{E}\left[\, \|AB-CR\|_F^2 \,\right] \le \frac{1}{\beta c}\, \|A\|_F^2\, \|B\|_F^2 . \tag{12}$$

Furthermore, let $\delta \in (0,1)$ and $\eta = 1+\sqrt{(8/\beta)\log(1/\delta)}$. Then, with probability at least $1-\delta$,

$$\|AB-CR\|_F^2 \le \frac{\eta^2}{\beta c}\, \|A\|_F^2\, \|B\|_F^2 . \tag{13}$$

In [13] it is shown that after one pass over the matrices nearly optimal probabilities can be constructed. In the present paper, we will be particularly interested in the case that $B = A^T$. In this case, using the Select algorithm of [13] random samples can be drawn according to nearly optimal probabilities using $O(1)$ additional space and time.

LINEARTIMESVD Algorithm.

Input: $A \in \mathbb{R}^{m \times n}$, $c, k \in \mathbb{Z}^+$ such that $1 \leq k \leq c \leq n$, $\{p_i\}_{i=1}^n$ such that $p_i \geq 0$ and $\sum_{i=1}^n p_i = 1$.

Output: $H_k \in \mathbb{R}^{m \times k}$ and $\sigma_t(C), t = 1, \ldots, k$.

1. For $t = 1$ to c,
   (a) Pick $i_t \in 1, \ldots, n$ with $\mathbf{Pr}[i_t = \alpha] = p_\alpha$, $\alpha = 1, \ldots, n$.
   (b) Set $C^{(t)} = A^{(i_t)}/\sqrt{cp_{i_t}}$.
2. Compute $C^T C$ and its SVD; say $C^T C = \sum_{t=1}^c \sigma_t^2(C) y^t y^{t^T}$.
3. Compute $h^t = Cy^t/\sigma_t(C)$ for $t = 1, \ldots, k$.
4. Return H_k, where $H_k^{(t)} = h^t$, and $\sigma_t(C), t = 1, \ldots, k$.

FIG. 1. *The* LINEARTIMESVD *algorithm.*

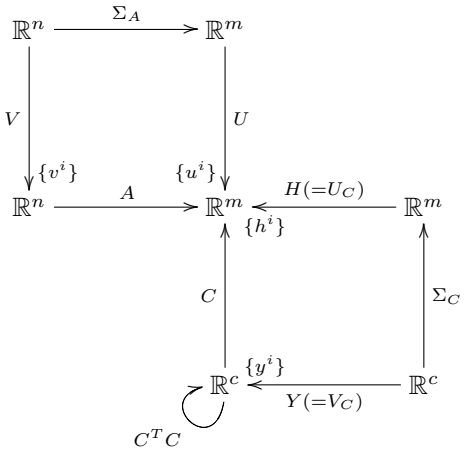


FIG. 2. *Diagram for the* LINEARTIMESVD *algorithm.*

4. Linear time SVD approximation algorithm.

4.1. The algorithm. Given a matrix $A \in \mathbb{R}^{m \times n}$ we wish to approximate its top k singular values and the corresponding singular vectors in a constant number of passes through the data and $O(cm + c^2)$ additional space and $O(c^2 m + c^3)$ additional time. The strategy behind the LINEARTIMESVD algorithm is to pick c columns of the matrix A, rescale each by an appropriate factor to form a matrix $C \in \mathbb{R}^{m \times c}$, and then compute the singular values and corresponding left singular vectors of the matrix C, which will be approximations to the singular values and left singular vectors of A, in a sense we make precise later. These are calculated by performing an SVD of the matrix $C^T C$ to compute the right singular vectors of C and from them calculating the left singular vectors of C.

The LINEARTIMESVD algorithm is described in Figure 1; it takes as input a matrix A and returns as output an approximation to the top k left singular values and the corresponding singular vectors. Note that by construction the SVD of C is

$C = H\Sigma_C Y^T$. A diagram illustrating the action of the LINEARTIMESVD algorithm is presented in Figure 2. The transformation represented by the matrix A is shown along with its SVD, and the transformation represented by the matrix C is also shown along with its SVD. It will be shown that if the probabilities $\{p_i\}_{i=1}^n$ are chosen judiciously, then the left singular vectors of C are with high probability approximations to the left singular vectors of A.

In section 4.2 we discuss running-time issues, and in section 4.3 we will prove the correctness of the algorithm.

4.2. Analysis of the implementation and running time. Assuming that nearly optimal sampling probabilities (as defined in section 3.3) are used, then in the LINEARTIMESVD algorithm the sampling probabilities p_k can be used to select columns to be sampled in one pass and $O(c)$ additional space and time using the SELECT algorithm of [13]. Given the elements to be sampled, the matrix C can then be constructed in one additional pass; this requires additional space and time that is $O(mc)$. Given $C \in \mathbb{R}^{m\times c}$, computing C^TC requires $O(mc)$ additional space and $O(mc^2)$ additional time, and computing the SVD of C^TC requires $O(c^3)$ additional time. Then computing H_k requires k matrix-vector multiplications for a total of $O(mck)$ additional space and time. Thus, overall $O(cm + c^2)$ additional space and $O(c^2m + c^3)$ additional time are required by the LINEARTIMESVD algorithm. Note that the "description" of the solution that is computable in the allotted additional space and time is the explicit approximation to the top k singular values and corresponding left singular vectors.

4.3. Analysis of the sampling step. Approximating A by $A_k = U_kU_k^TA$ incurs an error equal to $\|A - A_k\|_F^2 = \sum_{t=k+1}^r \sigma_t^2(A)$ and $\|A - A_k\|_2 = \sigma_{k+1}(A)$, since A_k is the "optimal" rank-k approximation to A with respect to both $\|\cdot\|_F$ and $\|\cdot\|_2$. We will show that in addition to this error the matrix $H_kH_k^TA$ has an error that depends on $\|AA^T - CC^T\|_F$. Then, using the results of Theorem 1, we will show that this additional error depends on $\|A\|_F^2$. We first consider obtaining a bound with respect to the Frobenius norm.

THEOREM 2. *Suppose $A \in \mathbb{R}^{m\times n}$ and let H_k be constructed from the* LINEARTIMESVD *algorithm. Then*

$$\left\|A - H_kH_k^TA\right\|_F^2 \le \|A - A_k\|_F^2 + 2\sqrt{k}\left\|AA^T - CC^T\right\|_F.$$

Proof. Recall that for matrices X and Y, $\|X\|_F^2 = \mathbf{Tr}(X^TX)$, $\mathbf{Tr}(X+Y) = \mathbf{Tr}(X) + \mathbf{Tr}(Y)$, and also that $H_k^TH_k = I_k$. Thus, we may express $\left\|A - H_kH_k^TA\right\|_F^2$ as

$$\begin{aligned}\left\|A - H_kH_k^TA\right\|_F^2 &= \mathbf{Tr}\left((A - H_kH_k^TA)^T(A - H_kH_k^TA)\right)\\ &= \mathbf{Tr}\left(A^TA - 2A^TH_kH_k^TA + A^TH_kH_k^TH_kH_k^TA\right)\\ &= \mathbf{Tr}\left(A^TA\right) - \mathbf{Tr}\left(A^TH_kH_k^TA\right)\\ &= \|A\|_F^2 - \left\|A^TH_k\right\|_F^2. \qquad (14)\end{aligned}$$

We may relate $\left\|A^T H^k\right\|_F^2$ and $\sum_{t=1}^k \sigma_t^2(C)$ by the following:

$$\begin{aligned}
\left| \left\|A^T H_k\right\|_F^2 - \sum_{t=1}^k \sigma_t^2(C) \right| &\leq \sqrt{k} \left(\sum_{t=1}^k \left(\left|A^T h^t\right|^2 - \sigma_t^2(C) \right)^2 \right)^{1/2} \\
&= \sqrt{k} \left(\sum_{t=1}^k \left(\left|A^T h^t\right|^2 - \left|C^T h^t\right|^2 \right)^2 \right)^{1/2} \\
&= \sqrt{k} \left(\sum_{t=1}^k \left({h^t}^T (AA^T - CC^T) h^t \right)^2 \right)^{1/2} \\
&\leq \sqrt{k} \left\|AA^T - CC^T\right\|_F . \qquad (15)
\end{aligned}$$

The first inequality follows by applying the Cauchy–Schwarz inequality; the last inequality follows by writing AA^T and CC^T with respect to a basis containing $\{h^t\}_{t=1}^k$. By again applying the Cauchy–Schwarz inequality, noting that $\sigma_t^2(X) = \sigma_t(XX^T)$ for a matrix X, and applying the Hoffman–Wielandt inequality (10), we may also relate $\sum_{k=1}^k \sigma_t^2(C)$ and $\sum_{k=1}^k \sigma_t^2(A)$ by the following:

$$\begin{aligned}
\left| \sum_{t=1}^k \sigma_t^2(C) - \sum_{t=1}^k \sigma_t^2(A) \right| &\leq \sqrt{k} \left(\sum_{t=1}^k \left(\sigma_t^2(C) - \sigma_t^2(A) \right)^2 \right)^{1/2} \\
&= \sqrt{k} \left(\sum_{t=1}^k \left(\sigma_t(CC^T) - \sigma_t(AA^T) \right)^2 \right)^{1/2} \\
&\leq \sqrt{k} \left(\sum_{t=1}^m \left(\sigma_t(CC^T) - \sigma_t(AA^T) \right)^2 \right)^{1/2} \\
&\leq \sqrt{k} \left\|CC^T - AA^T\right\|_F . \qquad (16)
\end{aligned}$$

Combining the results of (15) and (16) allows us to relate $\left\|A^T H_k\right\|_F^2$ and $\sum_{t=1}^k \sigma_t^2(A)$ by the following:

$$\left| \left\|A^T H_k\right\|_F^2 - \sum_{t=1}^k \sigma_t^2(A) \right| \leq 2\sqrt{k} \left\|AA^T - CC^T\right\|_F . \qquad (17)$$

Combining (17) with (14) yields the theorem. ∎

We next prove a similar result for the spectral norm; note that the factor $\sqrt{k}$ is not present.

THEOREM 3. *Suppose $A \in \mathbb{R}^{m \times n}$ and let H_k be constructed from the* LINEARTIMESVD *algorithm. Then*

$$\left\|A - H_k H_k^T A\right\|_2^2 \leq \|A - A_k\|_2^2 + 2 \left\|AA^T - CC^T\right\|_2 .$$

Proof. Let $\mathcal{H}_k = \text{range}(H_k) = \mathbf{span}(h^1, \ldots, h^k)$ and let $\mathcal{H}_{m-k}$ be the orthogonal complement of $\mathcal{H}_k$. Let $x \in \mathbb{R}^m$ and let $x = \alpha y + \beta z$, where $y \in \mathcal{H}_k$, $z \in \mathcal{H}_{m-k}$, and

$\alpha^2 + \beta^2 = 1$; then

$$\begin{aligned}
\left\|A - H_k H_k^T A\right\|_2 &= \max_{x \in \mathbb{R}^m, |x|=1} \left|x^T (A - H_k H_k^T A)\right| \\
&= \max_{y \in \mathcal{H}_k, |y|=1, z \in \mathcal{H}_{m-k}, |z|=1, \alpha^2+\beta^2=1} \left|(\alpha y^T + \beta z^T)(A - H_k H_k^T A)\right| \\
&\leq \max_{y \in \mathcal{H}_k, |y|=1} \left|y^T (A - H_k H_k^T A)\right| + \max_{z \in \mathcal{H}_{m-k}, |z|=1} \left|z^T (A - H_k H_k^T A)\right| \qquad (18) \\
&= \max_{z \in \mathcal{H}_{m-k}, |z|=1} \left|z^T A\right|. \qquad (19)
\end{aligned}$$

Inequality (18) follows since $\alpha, \beta \leq 1$ and (19) follows since $y \in \mathcal{H}_k$ and $z \in \mathcal{H}_{m-k}$. We next bound (19):

$$\begin{aligned}
\left|z^T A\right|^2 &= z^T C C^T z + z^T \left(A A^T - C C^T\right) z \\
&\leq \sigma_{k+1}^2(C) + \|A A^T - C C^T\|_2 \qquad (20) \\
&\leq \sigma_{k+1}^2(A) + 2\|A A^T - C C^T\|_2 \qquad (21) \\
&= \|A - A_k\|_2^2 + 2\|A A^T - C C^T\|_2. \qquad (22)
\end{aligned}$$

Inequality (20) follows since $\max_{z \in \mathcal{H}_{m-k}} |z^T C|$ occurs when z is the $(k+1)$st left singular vector, i.e., the maximum possible in the $\mathcal{H}_{m-k}$ subspace. Inequality (21) follows since $\sigma_{k+1}^2(C) = \sigma_{k+1}(C C^T)$ and since by (9) we have that $\sigma_{k+1}^2(C) \leq \sigma_{k+1}(A A^T) + \|A A^T - C C^T\|_2$; (22) follows since $\|A - A_k\|_2 = \sigma_{k+1}(A)$. The theorem then follows by combining (19) and (22). □

Theorems 2 and 3 hold regardless of the sampling probabilities $\{p_i\}_{i=1}^n$. Since $\|A - A_k\|_\xi$, $\xi = 2, F$, is a property of the matrix A, the choice of sampling probabilities enters into the error of $\|A - H_k H_k^T A\|_\xi^2$ only through the term involving the additional error beyond the optimal rank-k approximation, i.e., the term $\|A A^T - C C^T\|_\xi$. Although the additional error in Theorem 3 depends on $\|A A^T - C C^T\|_2$, we note that $\|A A^T - C C^T\|_2 \leq \|A A^T - C C^T\|_F$ and will use a bound for the latter quantity to bound the former in the following. Note that the prefactor of the additional error is $2\sqrt{k}$ for $\|\cdot\|_F^2$, while that for $\|\cdot\|_2^2$ is only 2.

In the following theorem we specialize the sampling probabilities to be those that are nearly optimal; by choosing enough columns, the error in the approximation of the SVD can be made arbitrarily small.

THEOREM 4. *Suppose $A \in \mathbb{R}^{m \times n}$; let H_k be constructed from the* LINEARTIMESVD *algorithm by sampling c columns of A with probabilities $\{p_i\}_{i=1}^n$ such that $p_i \geq \beta \left|A^{(i)}\right|^2 / \|A\|_F^2$ for some positive $\beta \leq 1$, and let $\eta = 1 + \sqrt{(8/\beta)\log(1/\delta)}$. Let $\epsilon > 0$. If $c \geq 4k/\beta\epsilon^2$, then*

$$\mathbf{E}\left[\left\|A - H_k H_k^T A\right\|_F^2\right] \leq \|A - A_k\|_F^2 + \epsilon \|A\|_F^2, \qquad (23)$$

and if $c \geq 4k\eta^2/\beta\epsilon^2$, then with probability at least $1 - \delta$,

$$\left\|A - H_k H_k^T A\right\|_F^2 \leq \|A - A_k\|_F^2 + \epsilon \|A\|_F^2. \qquad (24)$$

In addition, if $c \geq 4/\beta\epsilon^2$, then

$$\mathbf{E}\left[\left\|A - H_k H_k^T A\right\|_2^2\right] \leq \|A - A_k\|_2^2 + \epsilon \|A\|_F^2, \qquad (25)$$

and if $c \geq 4\eta^2/\beta\epsilon^2$*, then with probability at least* $1-\delta$*,*

$$\left\|A - H_k H_k^T A\right\|_2^2 \leq \|A - A_k\|_2^2 + \epsilon \|A\|_F^2 . \tag{26}$$

Proof. By combining Theorems 2 and 3 with Theorem 1 we have that

$$\mathbf{E}\left[\, \left\|A - H_k H_k^T A\right\|_F^2 \,\right] \leq \|A - A_k\|_F^2 + \left(\frac{4k}{\beta c}\right)^{1/2} \|A\|_F^2 , \tag{27}$$

$$\mathbf{E}\left[\, \left\|A - H_k H_k^T A\right\|_2^2 \,\right] \leq \|A - A_k\|_2^2 + \left(\frac{4}{\beta c}\right)^{1/2} \|A\|_F^2 , \tag{28}$$

and that with probability at least $1-\delta$,

$$\left\|A - H_k H_k^T A\right\|_F^2 \leq \|A - A_k\|_F^2 + \left(\frac{4\eta^2 k}{\beta c}\right)^{1/2} \|A\|_F^2 , \tag{29}$$

$$\left\|A - H_k H_k^T A\right\|_2^2 \leq \|A - A_k\|_2^2 + \left(\frac{4\eta^2}{\beta c}\right)^{1/2} \|A\|_F^2 . \tag{30}$$

The theorem follows by using the appropriate value of c. □

Note that alternatively one could sample rows instead of columns of a matrix; in this case, a modified version of the LinearTimeSVD algorithm leads to results analogous to Theorems 2 through 4.

5. Constant time SVD approximation algorithm.

5.1. The algorithm. Given a matrix $A \in \mathbb{R}^{m\times n}$ we now wish to approximate its top k singular values and the corresponding singular vectors in a constant number of passes through the data and additional space and time that are $O(1)$, independent of m and n. The strategy behind the ConstantTimeSVD algorithm is to pick c columns of the matrix A, rescale each by an appropriate factor to form a matrix $C \in \mathbb{R}^{m\times c}$, and then compute approximations to the singular values and left singular vectors of the matrix C, which will then be approximations to the singular values and left singular vectors of A. In the LinearTimeSVD algorithm of section 4, the left singular vectors of the matrix C are computed exactly; as the analysis of section 4.2 showed, this computation takes additional space and time that is linear in $m+n$ (assuming that c is constant). With the ConstantTimeSVD algorithm, in order to use only a constant $O(1)$ additional space and time, sampling is performed again, drawing rows of C to construct a matrix $W \in \mathbb{R}^{w\times c}$. The SVD of W^TW is then computed; let $W^TW = Z\Sigma_{W^TW}Z^T = Z\Sigma_W^2 Z^T$. The singular values and corresponding singular vectors so obtained are with high probability approximations to the singular values and singular vectors of C^TC and thus to the singular values and right singular vectors of C. Note that this is simply using the LinearTimeSVD algorithm to approximate the right singular vectors of C by randomly sampling rows of C.

The ConstantTimeSVD algorithm is described in Figure 3; it takes as input a matrix A and returns as output a "description" of an approximation to the top k left singular values and the corresponding singular vectors. This "description" of the approximations to the left singular vectors of A may, at the expense of one additional pass and linear additional space and time, be converted into an explicit

ConstantTimeSVD Algorithm.

Input: $A \in \mathbb{R}^{m \times n}$, $c, w, k \in \mathbb{Z}^+$ such that $1 \leq w \leq m$, $1 \leq c \leq n$, and $1 \leq k \leq \min(w, c)$, and $\{p_i\}_{i=1}^n$ such that $p_i \geq 0$ and $\sum_{i=1}^n p_i = 1$.

Output: $\sigma_t(W), t = 1, \ldots, \ell$ and a "description" of $\tilde{H}_\ell \in \mathbb{R}^{m \times \ell}$.

1. For $t = 1$ to c,
   (a) Pick $i_t \in 1, \ldots, n$ with $\mathbf{Pr}[i_t = \alpha] = p_\alpha$, $\alpha = 1, \ldots, n$, and save $\{(i_t, p_{j_t}) : t = 1, \ldots, c\}$.
   (b) Set $C^{(t)} = A^{(i_t)}/\sqrt{cp_{i_t}}$. (Note that C is not explicitly constructed in RAM.)
2. Choose $\{q_j\}_{j=1}^m$ such that $q_j = \left|C_{(j)}\right|^2 / \|C\|_F^2$.
3. For $t = 1$ to w,
   (a) Pick $j_t \in 1, \ldots, m$ with $\mathbf{Pr}[j_t = \alpha] = q_\alpha$, $\alpha = 1, \ldots, m$.
   (b) Set $W_{(t)} = C_{(j_t)}/\sqrt{wq_{j_t}}$.
4. Compute $W^T W$ and its SVD. Say $W^T W = \sum_{t=1}^c \sigma_t^2(W) z^t z^{tT}$.
5. If a $\|\cdot\|_F$ bound is desired, set $\gamma = \epsilon/100k$,
   Else if a $\|\cdot\|_2$ bound is desired, set $\gamma = \epsilon/100$.
6. Let $\ell = \min\{k, \max\{t : \sigma_t^2(W) \geq \gamma \|W\|_F^2\}\}$.
7. Return singular values $\{\sigma_t(W)\}_{t=1}^\ell$ and their corresponding singular vectors $\{z^t\}_{t=1}^\ell$.

Fig. 3. *The* ConstantTimeSVD *algorithm.*

approximation to the left singular vectors of A by using $C = \tilde{H}\Sigma_W Z^T$ to compute $\tilde{H}$, whose columns are approximations of the left singular vectors of C. Note that γ in the ConstantTimeSVD algorithm is introduced to bound small singular values of C that may be perturbed by the second level of sampling; as indicated, the particular value of γ that is chosen depends on the norm bound which is desired. Note also that the probabilities $\{q_j\}_{j=1}^m$ used in the algorithm are optimal (in the sense of section 3.3), as will be the probabilities $\{p_i\}_{i=1}^n$ which will enter into Theorem 5.

A diagram illustrating the action of the ConstantTimeSVD algorithm is presented in Figure 4. The transformation represented by the matrix A is represented along with its SVD, and the transformation represented by the matrix C is also shown (but note that its SVD is not shown). The transformation represented by the matrix W, which is constructed from C with the second level of sampling, is also shown along with its SVD. In addition, approximations to the right singular vectors of C and to the left singular vectors of C calculated from $C = \tilde{H}\Sigma_W Z^T$ are shown.

In section 5.2 we will show that this algorithm takes $O(1)$, i.e., a constant with respect to m and n, additional space and time, assuming that c and w are constant. In section 5.3 we will state Theorem 5, which will establish the correctness of the algorithm; this theorem is the main result of this section and is the analogue of Theorem 4. Finally, in section 5.4 we will prove Theorem 5.

5.2. Analysis of the implementation and running time. Assuming that optimal sampling probabilities (as defined in section 3.3) are used, then in the ConstantTimeSVD algorithm the sampling probabilities p_k can be used to select columns to be sampled in one pass and $O(c)$ additional space and time using the Select algorithm of [13]. Given the columns of A to be sampled, we do not explicitly construct the

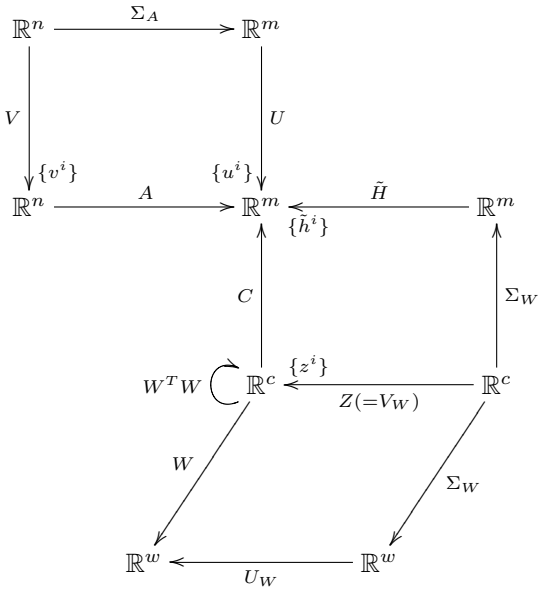


FIG. 4. *Diagram for the* CONSTANTTIMESVD *algorithm.*

matrix C but instead perform a second level of sampling and select w rows of C with probabilities $\{q_i\}_{i=1}^m$ (as described in the CONSTANTTIMESVD algorithm) in order to construct the matrix W. We do this by performing a second pass and using $O(w)$ additional space and time, again using the SELECT algorithm. Then in a third pass we explicitly construct W; this requires additional space and time that is $O(cw)$. Then, given W, computing W^TW requires $O(cw)$ additional space and $O(c^2w)$ additional time, and computing the SVD of W^TW requires $O(c^3)$ additional time. The singular values and corresponding singular vectors thus computed can then be returned as the "description" of the solution. The total additional time for the CONSTANTTIMESVD algorithm is then $O(c^3 + cw^2)$; this is a constant if c and w are assumed to be a constant. To explicitly compute $\tilde{H}_k$ would require k matrix-vector multiplications which would require another pass over the data and $O(mck)$ additional space and time.

5.3. Statement of Theorem 5. This subsection and the next provide an analysis of the CONSTANTTIMESVD algorithm similar to the analysis of the LINEARTIMESVD algorithm found in section 4.3. Recall that in section 4 we were interested in bounding $\left\|A - H_k H_k^T A\right\|_\xi^2$, where $\xi = F, 2$. In that case, $H_k^T H_k = I_k$, $H_k H_k^T$ was an orthonormal projection, and $H_k H_k^T A$ was our rank at most k approximation to A. In the constant time model, we do not have access to H_k but instead to $\tilde{H}_\ell$, where the columns of $\tilde{H}_\ell$, i.e., $\tilde{h}^t = Cz^t/\sigma_t(W), t = 1, \ldots, \ell$, do *not* form an orthonormal set. However, by Lemma 2 of section 5.4.1, if C and W are constructed by sampling with optimal probabilities, then with high probability the columns of $\tilde{H}_\ell$ are approximately orthonormal, $\tilde{H}_\ell^T \tilde{H}_\ell \approx I_\ell$, and $\tilde{H}_\ell \tilde{H}_\ell^T = \sum_{t=1}^\ell \tilde{h}^t \tilde{h}^{t^T}$ is approximately an orthonormal projection. Applying this to A, we will get our low-rank approximation. Note that in dealing with this nonorthonormality the original proof of [16] contained a small error which was corrected in the journal version [17].

In this section and the next we use the following notation. Recall that the SVD of $W^TW \in \mathbb{R}^{c\times c}$ is

$$W^TW = \sum_{t=1}^{c} \sigma_t^2(W) z^t z^{tT} = Z\Sigma_W^2 Z^T, \tag{31}$$

where $Z \in \mathbb{R}^{c\times c}$. Define $Z_{\alpha,\beta} \in \mathbb{R}^{c\times(\beta-\alpha+1)}$ to be the matrix whose columns are the αth through the βth singular vectors of W^TW. Then

$$\tilde{H}_\ell = CZ_{1,\ell}T, \tag{32}$$

where $T \in \mathbb{R}^{\ell\times\ell}$ is the diagonal matrix with elements $T_{tt} = 1/\sigma_t(W)$. In addition, let the SVD of $\tilde{H}_\ell$ be

$$\tilde{H}_\ell = B_\ell \Sigma_{\tilde{H}_\ell} D_\ell^T, \tag{33}$$

and let us define the matrix $\Delta \in \mathbb{R}^{\ell\times\ell}$ to be

$$\Delta = TZ_{1,\ell}^T(C^TC - W^TW)Z_{1,\ell}T. \tag{34}$$

We will see that Δ is a measure of the degree to which the columns of $\tilde{H}_\ell$ are not orthonormal.

Theorem 5 is the constant time analogue of Theorem 4 and is the main result of this section. Note that since the results from sampling at the second step, i.e., sampling from the matrix C to form the matrix W, depend on the samples chosen in the first sampling step, we do not state the following results in expectation, but instead state them with high probability.

THEOREM 5. *Suppose $A \in \mathbb{R}^{m\times n}$; let a description of $\tilde{H}_\ell$ be constructed from the* ConstantTimeSVD *algorithm by sampling c columns of A with probabilities $\{p_i\}_{i=1}^n$ and w rows of C with probabilities $\{q_j\}_{j=1}^m$ where $p_i = |A^{(i)}|^2/\|A\|_F^2$ and $q_j = |C_{(j)}|^2/\|C\|_F^2$. Let $\eta = 1+\sqrt{8\log(2/\delta)}$ and $\epsilon > 0$.*

If a Frobenius norm bound is desired, and hence the ConstantTimeSVD *algorithm is run with $\gamma = \epsilon/100k$, then by choosing $c = \Omega(k^2\eta^2/\epsilon^4)$ columns of A and $w = \Omega(k^2\eta^2/\epsilon^4)$ rows of C we have that with probability at least $1-\delta$,*

$$\left\|A - \tilde{H}_\ell\tilde{H}_\ell^T A\right\|_F^2 \le \|A - A_k\|_F^2 + \epsilon\|A\|_F^2. \tag{35}$$

If a spectral norm bound is desired, and hence the ConstantTimeSVD *algorithm is run with $\gamma = \epsilon/100$, then by choosing $c = \Omega(\eta^2/\epsilon^4)$ columns of A and $w = \Omega(\eta^2/\epsilon^4)$ rows of C we have that with probability at least $1-\delta$,*

$$\left\|A - \tilde{H}_\ell\tilde{H}_\ell^T A\right\|_2^2 \le \|A - A_k\|_2^2 + \epsilon\|A\|_F^2. \tag{36}$$

Proof. See section 5.4 for the proof. ∎

Recall that in section 4 we first proved Theorems 2 and 3, which provided a bound on $\left\|A - H_kH_k^TA\right\|_F^2$ and $\left\|A - H_kH_k^TA\right\|_2^2$, respectively, for arbitrary probabilities, and then we proved Theorem 4 for the nearly optimal probabilities. Although a similar presentation strategy could be adopted in this section, in the interests of simplicity (due to the technically more complicated proofs in the constant time model) we instead immediately restrict ourselves in Theorem 5 to the case of optimal sampling probabilities and defer the proofs of the supporting lemmas to section 5.4.

5.4. Proof of Theorem 5. In this section, we prove Theorem 5. We start in section 5.4.1 with several lemmas that are common to both the Frobenius and spectral norms. Then in section 5.4.2 we provide the proof of (35). Finally, in section 5.4.3 we provide the proof of (36).

5.4.1. General lemmas. In this section, we prove four lemmas that are used in the proofs of both the Frobenius and spectral norm results.

First, we relate $\left\|A - \tilde{H}_\ell \tilde{H}_\ell^T A\right\|_\xi^2$, for $\xi = 2, F$, to $\left\|A - B_\ell B_\ell^T A\right\|_\xi^2$ plus an error term; we do so since the columns of B_ℓ are orthonormal, which will allow us to bound $\left\|A - B_\ell B_\ell^T A\right\|_\xi^2$ using arguments similar to those used to bound $\left\|A - H_k H_k^T A\right\|_\xi^2$ in Theorems 2 and 3.

LEMMA 1. *For $\xi = 2, F$ and for any $\epsilon > 0$,*

$$\left\|A - \tilde{H}_\ell \tilde{H}_\ell^T A\right\|_\xi^2 \leq \left(1 + \frac{\epsilon}{100}\right) \left\|A - B_\ell B_\ell^T A\right\|_\xi^2 + \left(1 + \frac{100}{\epsilon}\right) \left\|B_\ell B_\ell^T - \tilde{H}_\ell \tilde{H}_\ell^T\right\|_\xi^2 \|A\|_\xi^2 .$$

Proof. By subadditivity and submultiplicitivity,

$$\left\|A - \tilde{H}_\ell \tilde{H}_\ell^T A\right\|_\xi^2 \leq \left(\left\|A - B_\ell B_\ell^T A\right\|_\xi + \left\|B_\ell B_\ell^T - \tilde{H}_\ell \tilde{H}_\ell^T\right\|_\xi \|A\|_\xi \right)^2.$$

The lemma follows since $(\alpha + \beta)^2 \leq (1 + \varepsilon)\alpha^2 + (1 + 1/\varepsilon)\beta^2$ for all $\varepsilon \geq 0$. ∎

Second, although the vectors $\tilde{h}^t = Cz^t/\sigma_t(W)$, $t = 1, \ldots, \ell$, do not in general form an orthonormal set, one would expect from their construction that if the matrix $W^T W$ is close to the matrix $C^T C$, then with high probability they will be approximately orthonormal. Lemma 2 establishes that Δ, defined in (34), characterizes how far $\tilde{H}_\ell$ is from having orthonormal columns and shows that the error introduced due to this nonorthonormality is bounded by a simple function of γ and the error introduced at the second level of sampling.

LEMMA 2. *When written in the basis with respect to Z,*

$$\tilde{H}_\ell^T \tilde{H}_\ell = I_\ell + \Delta.$$

Furthermore, for $\xi = 2, F$

$$\|\Delta\|_\xi \leq \frac{1}{\gamma \|W\|_F^2} \left\|C^T C - W^T W\right\|_\xi .$$

Proof. Recall that $\tilde{H}_\ell = C Z_{1,\ell} T$ and that $T^T Z_{1,\ell}^T W^T W Z_{1,\ell} T = I_\ell$, so that

$$\left\|\tilde{H}_\ell^T \tilde{H}_\ell - I_\ell\right\|_\xi = \left\|T^T Z_{1,\ell}^T C^T C Z_{1,\ell} T - T^T Z_{1,\ell}^T W^T W Z_{1,\ell} T\right\|_\xi \tag{37}$$

$$= \left\|T^T Z_{1,\ell}^T \left(C^T C - W^T W\right) Z_{1,\ell} T\right\|_\xi . \tag{38}$$

Using the submultiplicativity properties of the 2-norm, and in particular

$$\|AB\|_\xi \leq \|A\|_2 \|B\|_\xi , \tag{39}$$

$$\|AB\|_\xi \leq \|A\|_\xi \|B\|_2 , \tag{40}$$

for both $\xi = 2, F$, we get

$$\left\|\tilde{H}_\ell^T \tilde{H}_\ell - I_\ell\right\|_\xi \leq \left\|T^T Z_{1,\ell}^T\right\|_2 \left\|C^T C - W^T W\right\|_\xi \left\|Z_{1,\ell} T\right\|_2 \tag{41}$$

$$\leq \|T\|_2^2 \left\|C^T C - W^T W\right\|_\xi \tag{42}$$

$$\leq \max_{t=1,\ldots,\ell} \left(1/\sigma_t^2(W)\right) \left\|C^T C - W^T W\right\|_\xi, \tag{43}$$

since $\|Z_{1,\ell}\|_2 = 1$. The lemma follows since $\sigma_t^2(W) \geq \gamma \|W\|_F^2$ for all $t = 1, \ldots, \ell$ by the definition of ℓ. □

Third, we consider the second term in Lemma 1, $\left\|B_\ell B_\ell^T - \tilde{H}_\ell \tilde{H}_\ell^T\right\|_\xi^2$ and show that it can be related to $\|\Delta\|_\xi$.

LEMMA 3. *For* $\xi = 2, F$

$$\left\|B_\ell B_\ell^T - \tilde{H}_\ell \tilde{H}_\ell^T\right\|_\xi = \|\Delta\|_\xi.$$

Proof. Since $\tilde{H}_\ell = B_\ell \Sigma_{\tilde{H}_\ell} D_\ell^T$, we have

$$\begin{aligned}\left\|B_\ell B_\ell^T - \tilde{H}_\ell \tilde{H}_\ell^T\right\|_\xi &= \left\|B_\ell \left(I_\ell - \Sigma_{\tilde{H}_\ell}^2\right) B_\ell^T\right\|_\xi \\ &= \left\|I_\ell - \Sigma_{\tilde{H}_\ell}^2\right\|_\xi \\ &= \left\|D_\ell \left(I_\ell - \Sigma_{\tilde{H}_\ell}^2\right) D_\ell^T\right\|_\xi \\ &= \left\|I_\ell - \tilde{H}_\ell^T \tilde{H}_\ell\right\|_\xi. \qquad \square\end{aligned}$$

Fourth, Lemma 4 considers the special case in which the probabilities $\{p_i\}_{i=1}^n$ that are entered into the CONSTANTTIMESVD algorithm are optimal, as is the case for Theorem 5.

LEMMA 4. *Let* $A \in \mathbb{R}^{m \times n}$ *and let* $\tilde{H}_\ell$ *be constructed from the* CONSTANTTIMESVD *algorithm by sampling* c *columns of* A *with probabilities* $\{p_i\}_{i=1}^n$ *and* w *rows of* C *with probabilities* $\{q_j\}_{j=1}^m$, *where* $p_i = \mathbf{Pr}[i_t = i] = |A^{(i)}|^2 / \|A\|_F^2$ *and* $q_j = \mathbf{Pr}[j_t = j] = |C_{(j)}|^2 / \|C\|_F^2$. *Then*

$$\|W\|_F = \|C\|_F = \|A\|_F.$$

Proof. If $p_i = \left|A^{(i)}\right|^2 / \|A\|_F^2$, we have that $\|C\|_F^2 = \sum_{t=1}^c \left|C^{(t)}\right|^2 = \sum_{t=1}^c \frac{\left|A^{(i_t)}\right|^2}{c p_{i_t}}$ $= \|A\|_F^2$. Similarly, if $q_j = \left|C_{(j)}\right|^2 / \|C\|_F^2$, we have that $\|W\|_F^2 = \sum_{t=1}^w \left|W_{(t)}\right|^2$ $= \sum_{t=1}^w \frac{\left|C_{(i_t)}\right|^2}{w q_{i_t}} = \|C\|_F^2$. The lemma follows. □

5.4.2. Lemmas for the Frobenius norm proof. In this section we prove (35). We do this by first proving lemmas sufficient to bound $\|A - B_\ell B_\ell^T A\|_F^2$; when this is combined with the lemmas of section 5.4.1 we obtain a bound on $\|A - \tilde{H}_\ell \tilde{H}_\ell^T A\|_F^2$. The bound on $\|A - B_\ell B_\ell^T A\|_F^2$ depends on the error for the optimal rank-k approximation to A, i.e., $\|A - A_k\|_F^2$, and additional errors that depend on the quality of the sampling approximations, i.e., on $\|AA^T - CC^T\|_F$ and $\|C^T C - W^T W\|_F$. This will be the analogue of Theorem 2 applied to the constant additional space and time model. The result and associated proof will have a similar structure to that of Theorem 2, but will be more complicated due to the nonorthonormality of the vectors $\tilde{h}^t, t = 1, \ldots, \ell$, and will involve additional error terms since two levels of approximation are involved.

We now prove several lemmas which will provide a bound for the first term in Lemma 1 when applied to the Frobenius norm. We first rewrite the $\left\|A - B_\ell B_\ell^T A\right\|_F^2$ term from Lemma 1. Note that Lemma 5 is the constant time analogue of (14).

LEMMA 5.

$$\left\|A - B_\ell B_\ell^T A\right\|_F^2 = \|A\|_F^2 - \left\|B_\ell^T A\right\|_F^2.$$

Proof.

$$\begin{aligned}\left\|A - B_\ell B_\ell^T A\right\|_F^2 &= \mathbf{Tr}\left(\left(A - B_\ell B_\ell^T A\right)^T \left(A - B_\ell B_\ell^T A\right)\right) \\ &= \mathbf{Tr}\left(A^T A - A^T B_\ell B_\ell^T A\right). \qquad \square\end{aligned}$$

Next, we want to provide a lower bound for $\left\|B_\ell^T A\right\|_F^2$ in terms of the singular values of W. We do so in several steps. First, we relate $\|B_\ell^T A\|_F^2$ to $\|\tilde{H}_\ell^T A\|_F^2$. We note that the assumption $\|\Delta\|_F < 1$ is made since in Theorem 5 optimal probabilities are used and sufficiently many columns and rows are drawn; if this assumption is dropped, then bounds of the form in Theorem 5 may be obtained with slightly worse sampling complexity.

LEMMA 6. *If* $\|\Delta\|_F < 1$, *then*

$$\left\|B_\ell^T A\right\|_F^2 \geq (1 - \|\Delta\|_F) \left\|\tilde{H}_\ell^T A\right\|_F^2.$$

Proof. Since $\tilde{H}_\ell = B_\ell \Sigma_{\tilde{H}_\ell} D_\ell^T$

$$\left\|\tilde{H}_\ell^T A\right\|_F^2 = \left\|\Sigma_{\tilde{H}_\ell} B_\ell^T A\right\|_F^2 \leq \left\|\Sigma_{\tilde{H}_\ell}\right\|_2^2 \left\|B_\ell^T A\right\|_F^2 = \left\|\tilde{H}_\ell^T \tilde{H}_\ell\right\|_2^2 \left\|B_\ell^T A\right\|_F^2, \tag{44}$$

using (39). From the triangle inequality

$$\left\|\tilde{H}_\ell^T \tilde{H}_\ell\right\|_2 \geq \left| \|I_\ell\|_2 - \left\|\tilde{H}_\ell^T \tilde{H}_\ell - I_\ell\right\|_2 \right| = |1 - \|\Delta\|_2|. \tag{45}$$

The lemma follows since $\|\Delta\|_2 \leq \|\Delta\|_F < 1$ and by observing that $1 + x \leq 1/(1-x)$ for all $x \leq 1$. $\square$

Second, we relate $\|\tilde{H}_\ell^T A\|_F^2$ to $\|\tilde{H}_\ell^T C\|_F^2$.

LEMMA 7.

$$\left\|\tilde{H}_\ell^T A\right\|_F^2 \geq \left\|\tilde{H}_\ell^T C\right\|_F^2 - \left(k + \sqrt{k}\|\Delta\|_F\right) \left\|AA^T - CC^T\right\|_F.$$

Proof. Since $\|\tilde{H}_\ell^T A\|_F^2 = \mathbf{Tr}(\tilde{H}_\ell^T A A^T \tilde{H}_\ell^T)$, we have that

$$\begin{aligned}\left\|\tilde{H}_\ell^T A\right\|_F^2 &= \mathbf{Tr}\left(\tilde{H}_\ell^T C C^T \tilde{H}_\ell^T\right) + \mathbf{Tr}\left(\tilde{H}_\ell^T (AA^T - CC^T) \tilde{H}_\ell^T\right) \\ &\geq \left\|\tilde{H}_\ell^T C\right\|_F^2 - \left\|AA^T - CC^T\right\|_2 \left\|\tilde{H}_\ell\right\|_F^2,\end{aligned}$$

where the inequality follows since

$$\begin{aligned}\left|\mathbf{Tr}\left(\tilde{H}_\ell^T (AA^T - CC^T) \tilde{H}_\ell^T\right)\right| &\leq \sum_t \left|(\tilde{H}_\ell^T)_{(t)} (AA^T - CC^T) (\tilde{H}_\ell)^{(t)}\right| \\ &\leq \left\|AA^T - CC^T\right\|_2 \left\|\tilde{H}_\ell\right\|_F^2.\end{aligned}$$

The lemma follows since $\|\cdot\|_2 \leq \|\cdot\|_F$ and since

$$\left\|\tilde{H}_\ell\right\|_F^2 = \sum_{t=1}^{\ell} \left|\tilde{h}^{t^T} \tilde{h}^t\right| = \sum_{t=1}^{\ell} 1 + \Delta_{tt} \leq k + \sqrt{k}\|\Delta\|_F. \qquad \square$$

Third, we relate $\|\tilde{H}_\ell^T C\|_F^2$ to $\sum_{t=1}^{\ell} \sigma_t^2(W)$.

LEMMA 8.

$$\left\|\tilde{H}_\ell^T C\right\|_F^2 \geq \sum_{t=1}^{\ell} \sigma_t^2(W) - \frac{2}{\sqrt{\gamma}} \left\|C^T C - W^T W\right\|_F .$$

Proof. Since $\|\tilde{H}_\ell^T C\|_F^2 = \|C^T \tilde{H}_\ell\|_F^2 = \|C^T C Z_{1,\ell} T\|_F^2$, we have

$$\begin{aligned}\left\|\tilde{H}_\ell^T C\right\|_F^2 &\geq \left(\left\|W^T W Z_{1,\ell} T\right\|_F - \left\|(C^T C - W^T W) Z_{1,\ell} T\right\|_F \right)^2 \\ &\geq \left(\left(\sum_{t=1}^{\ell} \sigma_t^2(W) \right)^{1/2} - \frac{1}{\sqrt{\gamma}\,\|W\|_F} \left\|(C^T C - W^T W)\right\|_F \right)^2 ,\end{aligned}$$

where the second inequality uses that $\|XZ\|_F \leq \|X\|_F$ for any matrix X if the matrix Z has orthonormal columns. By multiplying out the right-hand side and ignoring terms that reinforce the inequality, the lemma follows since $(\sum_{t=1}^{\ell} \sigma_t^2(W))^{1/2} / \|W\|_F \leq 1$. □

By combining Lemmas 6, 7, and 8, we have our desired bound on $\|B_\ell^T A\|_F^2$ in terms of the singular values of W. Finally, we use matrix perturbation theory to relate $\sum_{t=1}^{\ell} \sigma_t^2(W)$ to $\sum_{t=1}^{k} \sigma_t^2(A)$.

LEMMA 9.

$$\sum_{t=1}^{\ell} \sigma_t^2(W) \geq \sum_{t=1}^{k} \sigma_t^2(A) - \sqrt{k} \left\|AA^T - CC^T\right\|_F - \sqrt{k} \left\|C^T C - W^T W\right\|_F - (k-\ell)\gamma \|W\|_F^2 .$$

Proof. Recalling the Hoffman–Wielandt inequality, we see that

$$\begin{aligned}\left| \sum_{t=1}^{k} \left(\sigma_t^2(C) - \sigma_t^2(A)\right) \right| &\leq \sqrt{k} \left(\sum_{t=1}^{k} \left(\sigma_t^2(C) - \sigma_t^2(A)\right)^2 \right)^{1/2} \\ &\leq \sqrt{k} \left(\sum_{t=1}^{k} \left(\sigma_t(CC^T) - \sigma_t(AA^T)\right)^2 \right)^{1/2} \\ &\leq \sqrt{k} \left\|AA^T - CC^T\right\|_F , \qquad (46)\end{aligned}$$

and, similarly, that

$$\begin{aligned}\left| \sum_{t=1}^{k} \left(\sigma_t^2(W) - \sigma_t^2(C)\right) \right| &\leq \sqrt{k} \left(\sum_{t=1}^{k} \left(\sigma_t^2(W) - \sigma_t^2(C)\right)^2 \right)^{1/2} \\ &\leq \sqrt{k} \left(\sum_{t=1}^{k} \left(\sigma_t(WW^T) - \sigma_t(CC^T)\right)^2 \right)^{1/2} \\ &\leq \sqrt{k} \left\|C^T C - W^T W\right\|_F . \qquad (47)\end{aligned}$$

By combining (46) and (47) we see that

$$\left| \sum_{t=1}^{k} \sigma_t^2(W) - \sum_{t=1}^{k} \sigma_t^2(A) \right| \leq \sqrt{k} \left\|AA^T - CC^T\right\|_F + \sqrt{k} \left\|C^T C - W^T W\right\|_F . \qquad (48)$$

Since $\sigma_t^2(W) < \gamma \|W\|_F^2$ for all $t = \ell+1, \ldots, k$ we have that $\sum_{t=\ell+1}^{k} \sigma_t^2(W) \leq (k-\ell)\gamma \|W\|_F^2$. Combining this with (48) allows us to relate $\sum_{t=1}^{\ell} \sigma_t^2(W)$ and $\sum_{t=1}^{k} \sigma_t^2(A)$, thus establishing the lemma. □

Now we combine these results in order to prove (35). Let $E_{AA^T} = AA^T - CC^T$ and $E_{C^TC} = C^TC - W^TW$. First, we establish a lower bound on $\left\|B_\ell^T A\right\|_F^2$. By combining Lemmas 6 and 7 and dropping terms that reinforce the inequality, we have that

$$\left\|B_\ell^T A\right\|_F^2 \geq \left\|\tilde{H}_\ell^T C\right\|_F^2 - \|\Delta\|_F \left\|\tilde{H}_\ell^T C\right\|_F^2 - \left(k + \sqrt{k}\,\|\Delta\|_F\right) \|E_{AA^T}\|_F .$$

By combining this with Lemmas 8 and 9 and dropping terms that reinforce the inequality, we have that

$$\begin{aligned}\left\|B_\ell^T A\right\|_F^2 \geq & \sum_{t=1}^{k} \sigma_t^2(A) - \left(k + \sqrt{k}\right) \|E_{AA^T}\|_F - \left(\sqrt{k} + \frac{2}{\sqrt{\gamma}}\right) \|E_{C^TC}\|_F \\ & - \|\Delta\|_F \sum_{t=1}^{k} \sigma_t^2(A) - \sqrt{k}\,\|\Delta\|_F \, \|E_{AA^T}\|_F - (k-\ell)\gamma \|W\|_F^2 .\end{aligned} \tag{49}$$

From Lemma 5 this immediately leads to the upper bound on $\left\|A - B_\ell B_\ell^T A\right\|_F^2$,

$$\begin{aligned}\left\|A - B_\ell B_\ell^T A\right\|_F^2 \leq & \|A - A_k\|_F^2 + \left(k + \sqrt{k}\right) \|E_{AA^T}\|_F + \left(\sqrt{k} + \frac{2}{\sqrt{\gamma}}\right) \|E_{C^TC}\|_F \\ & + \|\Delta\|_F \sum_{t=1}^{k} \sigma_t^2(A) + \sqrt{k}\,\|\Delta\|_F \, \|E_{AA^T}\|_F + (k-\ell)\gamma \|W\|_F^2 .\end{aligned} \tag{50}$$

From Lemmas 1 and 3,

$$\left\|A - \tilde{H}_\ell \tilde{H}_\ell^T A\right\|_F^2 \leq \left(1 + \frac{\epsilon}{100}\right) \left\|A - B_\ell B_\ell^T A\right\|_F^2 + \left(1 + \frac{100}{\epsilon}\right) \|\Delta\|_F^2 \, \|A\|_F^2 . \tag{51}$$

Recall that $\gamma = \epsilon/100k$, that $\sum_{t=1}^{k} \sigma_t^2(A) \leq \|A\|_F^2$, that $\|\Delta\|_F \leq \|E_{C^TC}\|_F / \gamma \|W\|_F^2$ by Lemma 2, and that $\|W\|_F = \|C\|_F = \|A\|_F$ by Lemma 4; (35) then follows by combining (50) and (51), using the sampling probabilities indicated in the statement of the theorem, and by choosing $c, w = \Omega(k^2\eta^2/\epsilon^4)$.

5.4.3. Lemmas for the spectral norm proof. In this section we prove (36). We do this by first proving lemmas sufficient to bound $\|A - B_\ell B_\ell^T A\|_2^2$; when this is combined with the lemmas of section 5.4.1, we obtain a bound on $\|A - \tilde{H}_\ell \tilde{H}_\ell^T A\|_2^2$. The bound on $\|A - B_\ell B_\ell^T A\|_2^2$ depends on the error for the optimal rank-k approximation to A, i.e., $\|A - A_k\|_2^2$, and additional errors that depend on the quality of the sampling approximations, i.e., on $\|AA^T - CC^T\|_2$ and $\|C^TC - W^TW\|_2$. This will be the analogue of Theorem 3 applied to the constant additional space and time model. The result and associated proof will have a similar structure to that of Theorem 3, but will be more complicated due to the nonorthonormality of the vectors $\tilde{h}^t, t = 1, \ldots, \ell$, and will involve additional error terms since two levels of approximation are involved.

We now prove three lemmas which will provide a bound for the first term in Lemma 1 when applied to the spectral norm. We first rewrite the $\|A - B_\ell B_\ell^T A\|_2^2$ term from Lemma 1.

LEMMA 10.

$$\left\|A - B_\ell B_\ell^T A\right\|_2^2 \leq \left\|Z_{k+1,c}^T C^T C Z_{k+1,c}\right\|_2 + \left\|Z_{\ell+1,k}^T C^T C Z_{\ell+1,k}\right\|_2 + \left\|AA^T - CC^T\right\|_2.$$

Proof. In order to bound $\|A - B_\ell B_\ell^T A\|_2$ we will project onto the subspace spanned by B_ℓ and its orthogonal complement in a manner analogous to that used in the proof of Theorem 3. Let $\mathcal{B}_\ell = \text{range}(B_\ell)$ and let $\mathcal{B}_{m-\ell}$ be the orthogonal complement of $\mathcal{B}_\ell$. Let $x = \alpha y + \beta z$, where $y \in \mathcal{B}_\ell$, $z \in \mathcal{B}_{m-\ell}$, and $\alpha^2 + \beta^2 = 1$. Then

$$\begin{aligned}
\left\|A - B_\ell B_\ell^T A\right\|_2 &= \max_{x \in \mathbb{R}^m, |x|=1} \left|x^T (A - B_\ell B_\ell^T A)\right| \\
&= \max_{y \in \mathcal{B}_\ell, |y|=1, z \in \mathcal{B}_{m-\ell}, |z|=1, \alpha^2+\beta^2=1} \left|(\alpha y^T + \beta z^T)(A - B_\ell B_\ell^T A)\right| \\
&\leq \max_{y \in \mathcal{B}_\ell, |y|=1} \left|y^T (A - B_\ell B_\ell^T A)\right| + \max_{z \in \mathcal{B}_{m-\ell}, |z|=1} \left|z^T (A - B_\ell B_\ell^T A)\right| \qquad (52) \\
&= \max_{z \in \mathcal{B}_{m-\ell}, |z|=1} \left|z^T A\right|. \qquad (53)
\end{aligned}$$

Inequality (52) follows since $\alpha, \beta \leq 1$ and (53) follows since $y \in \mathcal{B}_\ell$ and $z \in \mathcal{B}_{m-\ell}$. To bound (53), let $z \in \mathcal{B}_{m-\ell}$, $|z| = 1$; then

$$\begin{aligned}
\left|z^T A\right|^2 &= z^T (AA^T) z \\
&= z^T (CC^T) z + z^T (AA^T - CC^T) z \\
&= z^T (CC^T - CZ_{1,k} Z_{1,k}^T C^T) z + z^T (CZ_{1,k} Z_{1,k}^T C^T) z + z^T (AA^T - CC^T) z \qquad (54) \\
&= z^T (CZ_{k+1,c} Z_{k+1,c}^T C^T) z + z^T (CZ_{\ell+1,k} Z_{\ell+1,k}^T C^T) z + z^T (AA^T - CC^T) z. \qquad (55)
\end{aligned}$$

Equation (55) follows since $I_c = ZZ^T = Z_{1,k} Z_{1,k}^T + Z_{k+1,c} Z_{k+1,c}^T$ and, since

$$CZ_{1,\ell} Z_{1,\ell}^T C^T = \sum_{t=1}^{\ell} Cz^t z^{t^T} C^T = \sum_{t=1}^{\ell} \sigma_t^2(W) \tilde{h}^t \tilde{h}^{t^T},$$

implies that

$$z^T CZ_{1,\ell} Z_{1,\ell}^T C^T z = 0 \qquad (56)$$

for $z \in \mathcal{B}_{m-\ell}$. Thus, by combining (53) and (55)

$$\left\|A - B_\ell B_\ell^T A\right\|_2^2 \leq \left\|CZ_{k+1,c} Z_{k+1,c}^T C^T\right\|_2 + \left\|CZ_{\ell+1,k} Z_{\ell+1,k}^T C^T\right\|_2 + \left\|AA^T - CC^T\right\|_2.$$

The lemma follows since $\|X^T X\|_2 = \|XX^T\|_2$ for any matrix X. ∎

We next bound the $\|Z_{k+1,c}^T C^T C Z_{k+1,c}\|_2$ term from Lemma 10; note that matrix perturbation theory is used in (59).

LEMMA 11.

$$\left\|Z_{k+1,c}^T C^T C Z_{k+1,c}\right\|_2 \leq \|A - A_k\|_2^2 + \left\|AA^T - CC^T\right\|_2 + 2\left\|C^T C - W^T W\right\|_2.$$

Proof. First note that

$$\left\|Z_{k+1,c}^T C^T C Z_{k+1,c}\right\|_2 \leq \left\|Z_{k+1,c}^T W^T W Z_{k+1,c}\right\|_2 + \left\|Z_{k+1,c}^T (C^T C - W^T W) Z_{k+1,c}\right\|_2. \tag{57}$$

Since

$$\begin{aligned}\left\|Z_{k+1,c}^T (C^T C - W^T W) Z_{k+1,c}\right\|_2 &\leq \left\|C^T C - W^T W\right\|_2 \|Z_{k+1,c}\|_2^2 \\ &= \left\|C^T C - W^T W\right\|_2\end{aligned}$$

and

$$\left\|Z_{k+1,c}^T W^T W Z_{k+1,c}\right\|_2 = \sigma_{k+1}^2(W),$$

it follows from (57) that

$$\left\|Z_{k+1,c}^T C^T C Z_{k+1,c}\right\|_2 \leq \sigma_{k+1}^2(W) + \left\|C^T C - W^T W\right\|_2. \tag{58}$$

By a double application of (9), we see that

$$\sigma_{k+1}^2(W) \leq \sigma_{k+1}^2(A) + \left\|AA^T - CC^T\right\|_2 + \left\|C^T C - W^T W\right\|_2. \tag{59}$$

The lemma follows by combining (58) and (59) since $\|A - A_k\|_2 = \sigma_{k+1}(A)$. ∎

Finally, we bound the $\left\|Z_{\ell+1,k}^T C^T C Z_{\ell+1,k}\right\|_2$ term from Lemma 10; note that if $\ell = k$, it is unnecessary.

LEMMA 12.

$$\left\|Z_{\ell+1,k}^T C^T C Z_{\ell+1,k}\right\|_2 \leq \left\|C^T C - W^T W\right\|_2 + \gamma \|W\|_F^2.$$

Proof. First note that

$$\left\|Z_{\ell+1,k}^T C^T C Z_{\ell+1,k}\right\|_2 \leq \left\|Z_{\ell+1,k}^T \left(C^T C - W^T W\right) Z_{\ell+1,k}\right\|_2 + \left\|Z_{\ell+1,k}^T W^T W Z_{\ell+1,k}\right\|_2. \tag{60}$$

Since

$$\begin{aligned}\left\|Z_{\ell+1,k}^T \left(C^T C - W^T W\right) Z_{\ell+1,k}\right\|_2 &\leq \left\|C^T C - W^T W\right\|_2 \|Z_{\ell+1,k}\|_2^2 \\ &= \left\|C^T C - W^T W\right\|_2\end{aligned}$$

and

$$\left\|Z_{\ell+1,k}^T W^T W Z_{\ell+1,k}\right\|_2 = \sigma_{\ell+1}^2(W),$$

it follows from (60) that

$$\left\|Z_{\ell+1,k}^T C^T C Z_{\ell+1,k}\right\|_2 \leq \left\|C^T C - W^T W\right\|_2 + \sigma_{\ell+1}^2(W). \tag{61}$$

The lemma follows since $\sigma_t^2(W) < \gamma \|W\|_F^2$ for all $t = \ell + 1, \ldots, k$. ∎

Now we combine these results in order to prove (36). Recall that $E_{AA^T} = AA^T - CC^T$ and $E_{C^TC} = C^TC - W^TW$. By combining Lemmas 10, 11, and 12, we have that

$$\left\|A - B_\ell B_\ell^T A\right\|_2^2 \leq \|A - A_k\|_2^2 + 2\|E_{AA^T}\|_2 + 3\|E_{C^TC}\|_2 + \gamma\|W\|_F^2. \tag{62}$$

From Lemmas 1 and 3,

$$\left\|A - \tilde{H}_\ell \tilde{H}_\ell^T A\right\|_2^2 \leq \left(1 + \frac{\epsilon}{100}\right)\left\|A - B_\ell B_\ell^T A\right\|_2^2 + \left(1 + \frac{100}{\epsilon}\right)\|\Delta\|_2^2\,\|A\|_2^2. \tag{63}$$

Recall that $\gamma = \epsilon/100$, that $\|\cdot\|_2 \leq \|\cdot\|_F$, and that $\|\Delta\|_2 \leq \|E_{C^TC}\|_2/\gamma\|W\|_F^2$ by Lemma 2; (36) follows by combining (62) and (63), using the sampling probabilities indicated in the statement of the theorem, and by choosing $c, w = \Omega(\eta^2/\epsilon^4)$.

6. Discussion and conclusion. We have presented two algorithms to compute approximations to the SVD of a matrix $A \in \mathbb{R}^{m\times n}$ which do not require that A be stored in RAM, but for which the additional space and time required (in addition to a constant number of passes over the matrix) is either linear in $m+n$ or is a constant independent of m and n; we have also proven error bounds for both algorithms with respect to both the Frobenius and spectral norms. Table 1 in section 1 presents a summary of the dependence of the sampling complexity on k and ϵ. With the LINEARTIMESVD algorithm, the additional error (beyond the optimal rank-k approximation) in the spectral norm bound can be made less than $\epsilon\|A\|_F^2$ by sampling $\Theta(1/\epsilon^2)$ columns, and the additional error in the Frobenius norm can be made less than $\epsilon\|A\|_F^2$ by sampling $\Theta(k/\epsilon^2)$ columns. Likewise, with the CONSTANTTIMESVD algorithm, the additional error in the spectral norm can be made less than $\epsilon\|A\|_F^2$ by sampling $\Theta(1/\epsilon^4)$ columns and rows, and the additional error in the Frobenius norm can be made less than $\epsilon\|A\|_F^2$ by sampling $\Theta(k^2/\epsilon^4)$ columns and rows. The results of [16] require $\Theta(k^4/\epsilon^3)$ columns and rows for the Frobenius (and thus the spectral) norm bound.

Recent work has focused on developing new techniques for proving lower bounds on the number of queries a sampling algorithm is required to perform in order to approximate a given function accurately with a low probability of error [4, 5]. In [5] these methods have been applied to the low-rank matrix approximation problem (defined as approximating the SVD with respect to the Frobenius norm) and to the matrix reconstruction problem. It is shown that any sampling algorithm that with high probability finds a good low-rank approximation requires $\Omega(m+n)$ queries. In addition, it is shown that even if the algorithm is given the exact weight distribution over the columns of a matrix, it will still require $\Omega(k/\epsilon^2)$ column queries to approximate A. Thus, the LINEARTIMESVD algorithm (see also the original [10]) is optimal with respect to Frobenius norm bounds for the rank parameter k and the CONSTANTTIMESVD algorithm (see also the original [16]) is optimal with respect to Frobenius norm bounds up to polynomial factors.

Acknowledgments. We would like to thank the following individuals for comments and fruitful discussions: Dimitris Achlioptas, Alan Frieze, Mauro Maggioni, Frank McSherry, and Santosh Vempala. We would also like to thank Dimitris Achlioptas and Frank McSherry for providing us with a preprint of [1], i.e., the journal version of [2], which provides a useful comparison of their results with ours. We would like to

thank the National Science Foundation for partial support of this work. Finally, we would like to thank an anonymous reviewer for carefully reading the paper and making numerous useful suggestions; in particular, the reviewer provided elegant, short proofs for Lemmas 2 and 6.

REFERENCES

[1] D. Achlioptas and F. McSherry, *Fast computation of low rank matrix approximations*, J. ACM, to appear.
[2] D. Achlioptas and F. McSherry, *Fast computation of low rank matrix approximations*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, 2001, pp. 611–618.
[3] O. Alter, P. O. Brown, and D. Botstein, *Singular value decomposition for genome-wide expression data processing and modeling*, Proc. Natl. Acad. Sci. USA, 97 (2000), pp. 10101–10106.
[4] Z. Bar-Yossef, *The Complexity of Massive Data Set Computations*, Ph.D. thesis, University of California, Berkeley, 2002.
[5] Z. Bar-Yossef, *Sampling lower bounds via information theory*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, 2003, pp. 335–344.
[6] M. W. Berry, Z. Drmač, and E. R. Jessup, *Matrices, vector spaces, and information retrieval*, SIAM Rev., 41 (1999), pp. 335–362.
[7] M. W. Berry, S. T. Dumais, and G. W. O'Brian, *Using linear algebra for intelligent information retrieval*, SIAM Rev., 37 (1995), pp. 573–595.
[8] R. Bhatia, *Matrix Analysis*, Springer-Verlag, New York, 1997.
[9] S. T. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, *Indexing by latent semantic analysis*, J. Amer. Soc. Inform. Sci., 41 (1990), pp. 391–407.
[10] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay, *Clustering in large graphs and matrices*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1999, pp. 291–299.
[11] P. Drineas and R. Kannan, *Fast Monte-Carlo algorithms for approximate matrix multiplication*, in Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science, 2001, pp. 452–459.
[12] P. Drineas and R. Kannan, *Pass efficient algorithms for approximating large matrices*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2003, pp. 223–232.
[13] P. Drineas, R. Kannan, and M. W. Mahoney, *Fast Monte Carlo algorithms for matrices* I: *Approximating matrix multiplication*, SIAM J. Comput., 36 (2006), pp. 132–157.
[14] P. Drineas, R. Kannan, and M. W. Mahoney, *Fast Monte Carlo algorithms for matrices* III: *Computing a compressed approximate matrix decomposition*, SIAM J. Comput., 36 (2006), pp. 184–206.
[15] P. Drineas, R. Kannan, and M. W. Mahoney, *Fast Monte Carlo Algorithms for Matrices* II: *Computing a Low-Rank Approximation to a Matrix*, Tech. Report YALEU/DCS/TR-1270, Department of Computer Science, Yale University, New Haven, CT, 2004.
[16] A. Frieze, R. Kannan, and S. Vempala, *Fast Monte-Carlo algorithms for finding low-rank approximations*, in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, 1998, pp. 370–378.
[17] A. Frieze, R. Kannan, and S. Vempala, *Fast Monte-Carlo algorithms for finding low-rank approximations*, J. ACM, 51 (2004), pp. 1025–1041.
[18] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, 1989.
[19] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, New York, 1985.
[20] P. Indyk, *Stable distributions, pseudorandom generators, embeddings and data stream computation*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, 2000, pp. 189–197.
[21] J. Kleinberg, *Two algorithms for nearest-neighbor search in high dimensions*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, 1997, pp. 599–608.
[22] J. Kleinberg, *Authoritative sources in a hyperlinked environment*, in Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms, 1998, pp. 668–677.
[23] K. V. Mardia, J. T. Kent, and J. M. Bibby, *Multivariate Analysis*, Academic Press, London,

1979.

[24] H. Murase and S. K. Nayar, *Visual learning and recognition of* 3*-d objects from appearance*, Internat. J. Comput. Vision, 14 (1995), pp. 5–24.

[25] C. H. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala, *Latent semantic indexing: A probabilistic analysis*, in Proceedings of the 17th ACM Symposium on Principles of Database Systems, 1998, pp. 159–168.

[26] S. Raychaudhuri, J. M. Stuart, and R. B. Altman, *Principal components analysis to summarize microarray experiments: Application to sporulation time series*, in Proceedings of the Pacific Symposium on Biocomputing 2000, 2000, pp. 455–466.

[27] G. W. Stewart and J. G. Sun, *Matrix Perturbation Theory*, Academic Press, New York, 1990.

[28] O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, and R. B. Altman, *Missing value estimation methods for DNA microarrays*, Bioinformatics, 17 (2001), pp. 520–525.

[29] M. Turk and A. Pentland, *Eigenfaces for recognition*, J. Cognitive Neurosci., 3 (1991), pp. 71–96.

[30] S. Vempala, *Random projection: A new approach to VLSI layout*, in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, 1998, pp. 389–395.

SIAM J. COMPUT.
Vol. 36, No. 1, pp. 184–206

FAST MONTE CARLO ALGORITHMS FOR MATRICES III: COMPUTING A COMPRESSED APPROXIMATE MATRIX DECOMPOSITION*

PETROS DRINEAS†, RAVI KANNAN‡, AND MICHAEL W. MAHONEY§

Abstract. In many applications, the data consist of (or may be naturally formulated as) an $m \times n$ matrix A which may be stored on disk but which is too large to be read into random access memory (RAM) or to practically perform superlinear polynomial time computations on it. Two algorithms are presented which, when given an $m \times n$ matrix A, compute approximations to A which are the product of three smaller matrices, C, U, and R, each of which may be computed rapidly. Let $A' = CUR$ be the computed approximate decomposition; both algorithms have provable bounds for the error matrix $A - A'$. In the first algorithm, c columns of A and r rows of A are randomly chosen. If the $m \times c$ matrix C consists of those c columns of A (after appropriate rescaling) and the $r \times n$ matrix R consists of those r rows of A (also after appropriate rescaling), then the $c \times r$ matrix U may be calculated from C and R. For any matrix X, let $\|X\|_F$ and $\|X\|_2$ denote its Frobenius norm and its spectral norm, respectively. It is proven that

$$\left\|A - A'\right\|_\xi \leq \min_{D:\mathrm{rank}(D)\leq k} \|A - D\|_\xi + poly(k, 1/c)\, \|A\|_F$$

holds in expectation and with high probability for both $\xi = 2, F$ and for all $k = 1, \ldots, \mathrm{rank}(A)$; thus by appropriate choice of k

$$\left\|A - A'\right\|_2 \leq \epsilon \|A\|_F$$

also holds in expectation and with high probability. This algorithm may be implemented without storing the matrix A in RAM, provided it can make two passes over the matrix stored in external memory and use $O(m + n)$ additional RAM (assuming that c and r are constants, independent of the size of the input). The second algorithm is similar except that it approximates the matrix C by randomly sampling a constant number of rows of C. Thus, it has additional error but it can be implemented in three passes over the matrix using only constant additional RAM. To achieve an additional error (beyond the best rank-k approximation) that is at most $\epsilon\|A\|_F$, both algorithms take time which is a low-degree polynomial in k, $1/\epsilon$, and $1/\delta$, where $\delta > 0$ is a failure probability; the first takes time linear in $\max(m, n)$ and the second takes time independent of m and n. The proofs for the error bounds make important use of matrix perturbation theory and previous work on approximating matrix multiplication and computing low-rank approximations to a matrix. The probability distribution over columns and rows and the rescaling are crucial features of the algorithms and must be chosen judiciously.

Key words. randomized algorithms, Monte Carlo methods, massive data sets, CUR matrix decomposition

AMS subject classification. 68W20

DOI. 10.1137/S0097539704442702

*Received by the editors April 5, 2004; accepted for publication (in revised form) November 17, 2005; published electronically May 26, 2006. The technical report version of this journal paper appeared as *Fast Monte Carlo Algorithms for Matrices* III: *Computing a Compressed Approximate Matrix Decomposition*, by P. Drineas, R. Kannan, and M. W. Mahoney [12]. A preliminary version of parts of this paper, in particular the main algorithm and main theorem of section 3, appeared as *Pass efficient algorithms for approximating large matrices*, by P. Drineas and R. Kannan [9].

http://www.siam.org/journals/sicomp/36-1/44270.html

†Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180 (drinep@cs.rpi.edu).

‡Department of Computer Science, Yale University, New Haven, CT 06520 (kannan@cs.yale.edu).
This author was supported in part by a grant from the NSF.
§Department of Mathematics, Yale University, New Haven, CT 06520 (mahoney@cs.yale.edu).

1. Introduction. We are interested in developing and analyzing fast Monte Carlo algorithms for performing useful computations on large matrices. In this paper we consider a new method for computing a compressed approximate decomposition of a large matrix; in two related papers we consider matrix multiplication and the singular value decomposition (SVD) [10, 11]. Since such computations generally require time which is superlinear in the number of nonzero elements of the matrix, we expect our algorithms to be useful in many applications where data sets are modeled by matrices and are extremely large. In all of these cases, we assume that the input matrices are prohibitively large to store in random access memory (RAM) and thus that only external memory storage is possible. Thus, our algorithms will be allowed to read the matrices a few, e.g., one, two, or three, times and keep a small randomly chosen and rapidly computable "sketch" of the matrices in RAM; computations will then be performed on this "sketch." We will work within the framework of the pass-efficient computational model, in which the scarce computational resources are the number of passes over the data, the additional RAM space required, and the additional time required [10, 9].

In many applications, an $m \times n$ matrix A is stored on disk and is too large to be read into RAM or to practically perform superlinear polynomial time computations on it; thus, one may be interested in a succinctly described, easily computed $m \times n$ matrix A' that is an approximation to A. Let c and r be positive, usually constant, integers that we choose, and for any matrix X let $\|X\|_F$ and $\|X\|_2$ denote its Frobenius and spectral norms (as defined in section 2.1), respectively. We present two algorithms that compute an approximation A' to the matrix A that has the following properties:

(i) $A' = CUR$, where C is an $m \times c$ matrix consisting of c randomly picked columns of A, R is an $r \times n$ matrix consisting of r randomly picked rows of A, and U is a $c \times r$ matrix computed from C, R.

(ii) C, U, and R can be defined after making a small constant number of passes (2 or 3 for the two algorithms presented in this paper) through the whole matrix A from disk.

(iii) U can be constructed using additional RAM space and time that is $O(m + n)$ (for the LINEARTIMECUR algorithm) or is $O(1)$ (for the CONSTANTTIMECUR algorithm), assuming that c and r are constant.

(iv) For every $\epsilon > 0$ and every k such that $1 \leq k \leq \text{rank}(A)$ we can choose c and r (to be specified below) such that, with high probability, A' satisfies

$$\|A - A'\|_2 \leq \min_{D:\text{rank}(D)\leq k} \|A - D\|_2 + \epsilon \|A\|_F ,$$

and thus we can choose c and r such that $\|A - A'\|_2 \leq \epsilon \|A\|_F$.

(v) For every $\epsilon > 0$ and every k such that $1 \leq k \leq \text{rank}(A)$ we can choose c and r (to be specified below) such that, with high probability, A' satisfies

$$\|A - A'\|_F \leq \min_{D:\text{rank}(D)\leq k} \|A - D\|_F + \epsilon \|A\|_F .$$

In the first algorithm, the LINEARTIMECUR algorithm of section 3, c columns of A and r rows of A are randomly sampled. If the $m \times c$ matrix C consists of those c columns of A (after appropriate rescaling) and the $r \times n$ matrix R consists of those R rows of A (also after appropriate rescaling), then from C and R the $c \times r$ matrix U is computed. This algorithm may be implemented without storing the matrix A in RAM, provided it can make two passes over the matrix stored in external memory

TABLE 1
Summary of sampling complexity.

Additional error for:	LINEARTIMECUR	CONSTANTTIMECUR
$\|A - A'\|_2$	$c, r = \frac{\eta^2}{\epsilon^4}, \frac{k}{\delta^2\epsilon^2}$	$c, w, r = \frac{\eta^2}{\epsilon^8}, \frac{\eta^2}{\epsilon^8}, \frac{k}{\delta^2\epsilon^2}$
$\|A - A'\|_F$	$c, r = \frac{k\eta^2}{\epsilon^4}, \frac{k}{\delta^2\epsilon^2}$	$c, w, r = \frac{k^2\eta^2}{\epsilon^8}, \frac{k^2\eta^2}{\epsilon^8}, \frac{k}{\delta^2\epsilon^2}$

and use $O(m+n)$ additional RAM. The second algorithm, the CONSTANTTIMECUR algorithm of section 4, is similar except that it approximates the matrix C by randomly sampling a constant number w of rows of C. Thus, our second algorithm requires only constant additional RAM but uses a third pass over the data and has additional error. To achieve an additional error (beyond the best rank-k approximation) that is at most $\epsilon \|A\|_F$, both algorithms take time which is a low-degree polynomial in k, $1/\epsilon$, and $1/\delta$, where $\delta > 0$ is a failure probability; the first algorithm takes time linear in $\max(m, n)$ and the second takes time independent of m and n. See Table 1 for a summary of the dependence of the sampling complexity on k and ϵ, δ, and $\eta = 1 + \sqrt{8 \log(1/\delta)}$. (Note that the two algorithms we present in this paper are most interesting when the matrix A is well approximated by a rank-k matrix, where k is assumed to be constant with respect to m and n. In this case, c, r, and w are also constant with respect to m and n. This is assumed throughout the remainder of this paper.)

The proofs for the error bounds make important use of linear algebra and matrix perturbation theory; see [19, 22, 28, 6] for an overview of these topics. In particular, the proofs use the approximate matrix multiplication results of [10] and the approximate SVD results of [11]. As with those previous works, the probability distribution over the columns, the probability distribution over the rows, and the respective rescaling are crucial features of the algorithms which must be chosen judiciously. In addition, as a by-product of the CUR decomposition, we can estimate the top k singular values of A.

Our CUR approximations may be viewed as a "dimension reduction" technique. Two common techniques for dimension reduction are the SVD and multidimensional scaling; see [19, 25]. Another method that has attracted renewed interest recently is the traditional "random projection" method where one projects the problem into a randomly chosen low-dimensional subspace [24, 29, 23]. These methods do not share properties (i) and (ii) and thus are not suited for very large problems. Our algorithms achieve (i) and (ii) at the cost of the $\epsilon \|A\|_F$ error. In addition, our CUR approximations may be viewed as an approximate decomposition of a matrix $A \approx CUR$; as with other decompositions, the CUR decomposition both reveals information about the structure of the matrix and allows computations to be performed more efficiently. For example, in applications of the LINEARTIMECUR algorithm, A' could be stored in RAM since it can be stored in $O(m+n)$ space (instead of $O(mn)$ space); in addition, A' could then be operated upon by, e.g., applying $A' = CUR$ to a query vector $x \in \mathbb{R}^n$, which is an operation that can be performed in $O(m+n)$ space (instead of $O(mn)$ space if A is used).

Our CUR approximations have been used in applications such as the reconstruction of a matrix given a sample of the matrix in a recommendation systems context and for "similarity query" problems which are widely used in areas such as information retrieval [14]. In this application, after A has been preprocessed, one gets "query" vectors x and must find the similarity of x to each row of A. Here, the similarity of

two vectors is defined to be their dot product or their normalized dot product; our technique can handle both. See [7] for related discussion. Note that the measure $\|A\|_2$ is a worst-case measure and that this is more useful in many contexts than an average-case measure like $\|A\|_F$, since the relevant query x often comes from a small-dimensional subspace and is not random. See also [2, 1] for a nice discussion of these issues. Our CUR approximations have also been used in theoretical applications such as designing and analyzing approximation algorithms for the max-cut problem [13]. In these applications, the use of the constant additional space and time framework is essential.

In other related work, Achlioptas and McSherry have also computed succinctly described matrix approximations using somewhat different sampling techniques [2, 1]. Also included in [2, 1] is a comparison of their methods with those of [8, 9, 18] and thus with the results we present here. When compared with our LinearTimeCUR algorithm, they achieve the same results for the Frobenius norm bound and slightly better results (with respect to $1/\epsilon$) for the spectral norm bound [1]; in their work, however, there is no analogue of our ConstantTimeCUR algorithm.

After this introduction, we provide in section 2 reviews of linear algebra, the pass-efficient model, and of several previous results from [10, 11] that are used in this paper. In section 3 we describe and analyze the LinearTimeCUR algorithm which computes an approximate CUR decomposition of a matrix A using linear (in m and n) additional space and time, and in section 4 we describe and analyze the ConstantTimeCUR algorithm which computes a description of an approximate CUR decomposition of a matrix A using only constant additional space and time. Finally, in section 5 we provide a discussion and conclusion.

Finally, note that c and r enter into the asymptotic analysis; for improved clarity, however, we generally take them to be constants that do not vary.

2. Review of relevant background.

2.1. Review of linear algebra. This section contains a review of linear algebra that will be useful throughout the paper; for more details, see [19, 22, 28, 6] and the references therein.

For a vector $x \in \mathbb{R}^n$ we let $|x| = (\sum_{i=1}^n |x_i|^2)^{1/2}$ denote its Euclidean length. For a matrix $A \in \mathbb{R}^{m\times n}$ we let $A^{(j)}$, $j = 1, \ldots, n$, denote the jth column of A as a column vector and $A_{(i)}$, $i = 1, \ldots, m$, denote the ith row of A as a row vector. We denote matrix norms by $\|A\|_\xi$, using subscripts to distinguish between various norms. Of particular interest will be the Frobenius norm, the square of which is $\|A\|_F^2 = \sum_{i=1}^m \sum_{j=1}^n A_{ij}^2$, and the spectral norm, which is defined by $\|A\|_2 = \sup_{x\in\mathbb{R}^n,\ x\neq 0} \frac{|Ax|}{|x|}$. These norms are related to each other as $\|A\|_2 \leq \|A\|_F \leq \sqrt{n}\,\|A\|_2$.

If $A \in \mathbb{R}^{m\times n}$, then there exist orthogonal matrices $U = [u^1 u^2 \ldots u^m] \in \mathbb{R}^{m\times m}$ and $V = [v^1 v^2 \ldots v^n] \in \mathbb{R}^{n\times n}$, where $\{u^t\}_{t=1}^m \in \mathbb{R}^m$ and $\{v^t\}_{t=1}^n \in \mathbb{R}^n$ are such that

$$U^T A V = \Sigma = \mathbf{diag}(\sigma_1, \ldots, \sigma_\rho),$$

where $\Sigma \in \mathbb{R}^{m\times n}$, $\rho = \min\{m, n\}$, and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_\rho \geq 0$. Equivalently, $A = U\Sigma V^T$. The three matrices U, V, and Σ constitute the SVD of A. The σ_i are the singular values of A and the vectors u^i, v^i are the ith left and the ith right singular vectors, respectively. If $k \leq r = \mathrm{rank}(A)$ and we define $A_k = U_k \Sigma_k V_k^T = \sum_{t=1}^k \sigma_t u^t v^{t^T}$, then the distance (as measured by both $\|\cdot\|_2$ and $\|\cdot\|_F$) between A and

any rank-k approximation to A is minimized by A_k, i.e.,

$$\min_{D\in\mathbb{R}^{m\times n}:\mathrm{rank}(D)\leq k} \|A-D\|_2 = \|A-A_k\|_2 = \sigma_{k+1}(A), \tag{1}$$

$$\min_{D\in\mathbb{R}^{m\times n}:\mathrm{rank}(D)\leq k} \|A-D\|_F^2 = \|A-A_k\|_F^2 = \sum_{t=k+1}^{r} \sigma_t^2(A). \tag{2}$$

2.2. Review of the pass-efficient model. The pass-efficient model of data-streaming computation is a computational model that is motivated by the observation that in modern computers the amount of disk storage, i.e., sequential access memory, has increased very rapidly, while RAM and computing speeds have increased at a substantially slower pace [10, 9]. In the pass-efficient model the three scarce computational resources are number of passes over the data and the additional RAM space and additional time required by the algorithm. The data are assumed to be stored on a disk, to consist of elements whose sizes are bounded by a constant, and to be presented to an algorithm on a read-only tape. See [10] for more details.

2.3. Review of approximate matrix multiplication. The BasicMatrixMultiplication algorithm to approximate the product of two matrices is presented and analyzed in [10]. When this algorithm is given as input two matrices, $A\in\mathbb{R}^{m\times n}$ and $B\in\mathbb{R}^{n\times p}$, a probability distribution $\{p_i\}_{i=1}^n$, and a number $c\leq n$, it returns as output two matrices, C and R, such that $CR\approx AB$; $C\in\mathbb{R}^{m\times c}$ is a matrix whose columns are c randomly chosen columns of A (suitably rescaled) and $R\in\mathbb{R}^{c\times p}$ is a matrix whose rows are the c corresponding rows of B (also suitably rescaled). An important aspect of this algorithm is the probability distribution $\{p_i\}_{i=1}^n$ used to choose column-row pairs. Although one could always use a uniform distribution, superior results are obtained if the probabilities are chosen judiciously. In particular, a set of sampling probabilities $\{p_i\}_{i=1}^n$ are the *optimal probabilities* (with respect to approximating the product AB) if they are of the form (3); for an explanation and discussion, see [10], where we prove the following.

Theorem 1. *Suppose $A\in\mathbb{R}^{m\times n}$, $B\in\mathbb{R}^{n\times p}$, $c\in\mathbb{Z}^+$ such that $1\leq c\leq n$, and $\{p_i\}_{i=1}^n$ are such that $p_i\geq 0$ and $\sum_{i=1}^n p_i = 1$. Construct C and R with the* BasicMatrixMultiplication *algorithm of* [10] *and let CR be an approximation to AB. If the probabilities $\{p_i\}_{i=1}^n$ are such that*

$$p_k = \frac{\left|A^{(k)}\right|\left|B_{(k)}\right|}{\sum_{k'=1}^n \left|A^{(k')}\right|\left|B_{(k')}\right|}, \tag{3}$$

then

$$\mathbf{E}\left[\|AB-CR\|_F\right] \leq \frac{1}{\sqrt{c}}\|A\|_F\|B\|_F. \tag{4}$$

If, in addition, we let $\delta\in(0,1)$ and $\eta = 1+\sqrt{8\log(1/\delta)}$, then with probability at least $1-\delta$,

$$\|AB-CR\|_F \leq \frac{\eta}{\sqrt{c}}\|A\|_F\|B\|_F. \tag{5}$$

Furthermore, if the probabilities $\{p_i\}_{i=1}^n$ are such that

$$p_k = \frac{\left|B_{(k)}\right|^2}{\|B\|_F^2}, \tag{6}$$

then

$$\mathbf{E}\left[\,\|AB - CR\|_F\right] \leq \frac{1}{\sqrt{c}}\,\|A\|_F\,\|B\|_F\,. \tag{7}$$

Note that with probabilities of the form (6), we do not get a bound of the form $\|AB - CR\|_F \leq \frac{1}{\sqrt{c}}\,\|A\|_F\,\|B\|_F$ by sampling $O(\log(1/\delta))$ columns without making additional, awkward assumptions on the input matrices; see [10]. Of course, by Markov's inequality we can (and will) obtain such a bound by sampling $O(1/\delta)$ columns.

2.4. Review of approximate SVD. The LINEARTIMESVD algorithm is presented in [11]. It is an algorithm which, when given a matrix $A \in \mathbb{R}^{m\times n}$, uses $O(m+n)$ additional space and time to compute an approximation to the top k singular values and the corresponding left singular vectors of A by randomly choosing c columns of A and rescaling each appropriately to construct a matrix $C \in \mathbb{R}^{m\times c}$, computing the top k singular values and corresponding right singular vectors of C, and using them to construct a matrix $H_k \in \mathbb{R}^{m\times k}$ consisting of approximations to the top k left singular vectors of A. In [11] we prove the following.

THEOREM 2. *Suppose $A \in \mathbb{R}^{m\times n}$ and let H_k be constructed from the* LINEARTIMESVD *algorithm of* [11]. *Then*

$$\left\|A - H_k H_k^T A\right\|_F^2 \leq \|A - A_k\|_F^2 + 2\sqrt{k}\left\|AA^T - CC^T\right\|_F, \tag{8}$$

$$\left\|A - H_k H_k^T A\right\|_2^2 \leq \|A - A_k\|_2^2 + 2\left\|AA^T - CC^T\right\|_2. \tag{9}$$

The CONSTANTTIMESVD algorithm is also presented in [11]. It is an algorithm which, when given a matrix $A \in \mathbb{R}^{m\times n}$, uses constant additional space and time to compute a description of an approximation to the top k singular values and the corresponding left singular vectors of A. It does so in a manner similar to that of the LINEARTIMESVD algorithm except that it performs a second level of sampling in order to estimate (rather than compute exactly) the top k singular values and corresponding singular vectors of C. The γ in the following theorem is a parameter of the CONSTANTTIMESVD algorithm of [11] that is related to the second level of sampling; it also appears in our CONSTANTTIMECUR algorithm, and is thus discussed in section 4. In [11] we prove the following.

THEOREM 3. *Suppose $A \in \mathbb{R}^{m\times n}$; let a description of $\tilde{H}_\ell$ be constructed from the* CONSTANTTIMESVD *algorithm of* [11] *by sampling c columns of A with probabilities $\{p_i\}_{i=1}^n$ and w rows of C with probabilities $\{q_j\}_{j=1}^m$ where $p_i = |A^{(i)}|^2/\|A\|_F^2$ and $q_j = |C_{(j)}|^2/\|C\|_F^2$. Let $\eta = 1 + \sqrt{8\log(2/\delta)}$ and $\epsilon > 0$.*

If a Frobenius norm bound is desired, and hence the CONSTANTTIMESVD *algorithm is run with $\gamma = \epsilon/100k$, then by choosing $c = \Omega(k^2\eta^2/\epsilon^4)$ columns of A and $w = \Omega(k^2\eta^2/\epsilon^4)$ rows of C we have that with probability at least $1 - \delta$,*

$$\left\|A - \tilde{H}_\ell \tilde{H}_\ell^T A\right\|_F^2 \leq \|A - A_k\|_F^2 + \epsilon\,\|A\|_F^2. \tag{10}$$

If a spectral norm bound is desired, and hence the CONSTANTTIMESVD *algorithm is run with $\gamma = \epsilon/100$, then by choosing $c = \Omega(\eta^2/\epsilon^4)$ columns of A and $w = \Omega(\eta^2/\epsilon^4)$ rows of C we have that with probability at least $1 - \delta$,*

$$\left\|A - \tilde{H}_\ell \tilde{H}_\ell^T A\right\|_2^2 \leq \|A - A_k\|_2^2 + \epsilon\,\|A\|_F^2. \tag{11}$$

3. The linear time *CUR* decomposition. In this section we describe and analyze the LINEARTIMECUR algorithm, which computes an approximate CUR decomposition of a matrix $A \in \mathbb{R}^{m \times n}$ using linear (in m and n) additional space and time. In section 4 we describe and analyze the CONSTANTTIMECUR algorithm which computes a description of an approximate CUR decomposition of a matrix A using only constant additional space and time. Both algorithms will make extensive use of the corresponding results from [11] for approximating the SVD of a matrix as well as results from [10] on approximating the product of two matrices. As with the SVD algorithms, the CONSTANTTIMECUR algorithm has a similar flavor to the LINEARTIMECUR algorithm, but is technically more complex due to the second level of sampling required. Thus, in this section we provide an extensive description of the LINEARTIMECUR algorithm and the motivation and intuition behind it, and in section 4 we highlight the differences between the linear additional time framework and the constant additional time framework.

3.1. The algorithm. Given a matrix $A \in \mathbb{R}^{m \times n}$, we wish to compute a succinctly described, easily computed matrix A' that is decomposable as $A' = CUR \in \mathbb{R}^{m \times n}$ and that satisfies properties (i)–(v) of section 1. The LINEARTIMECUR algorithm, which is presented in Figure 1, accomplishes this by first forming a matrix $C \in \mathbb{R}^{m \times c}$ by rescaling a randomly chosen subset of c columns of A; the columns are chosen in c independent identical trials where in each trial the αth column of A is chosen with probability q_α, and if the αth column is chosen, it is rescaled by $1/\sqrt{cq_\alpha}$ before inclusion in C. The algorithm then forms a matrix $R \in \mathbb{R}^{r \times n}$ by rescaling a randomly chosen subset of r rows of A; the rows are chosen in r independent identical trials where in each trial the αth row of A is chosen with the probability p_α, and if the αth row is chosen, it is rescaled by $1/\sqrt{rp_\alpha}$ before inclusion in R. Using the same randomly chosen rows to construct R from A the algorithm also constructs a matrix Ψ from C in an identical manner. Thus, $\Psi \in \mathbb{R}^{r \times c}$ and $\Psi_{ij} = A_{i_{t_1} j_{t_2}} / \sqrt{crp_{i_{t_1}} q_{j_{t_2}}}$, where i_{t_1} is the element of $\{1, \ldots, m\}$ selected in the t_1th row sampling trial and j_{t_2} is the element of $\{1, \ldots, n\}$ selected in the t_2th column sampling trial.

The following sampling matrix formalism provides a convenient representation of our ideas, and will be used extensively in this section and the next. (See [10] for another use of this sampling matrix formalism.) Let us define the column sampling matrix $S_C \in \mathbb{R}^{n \times c}$ to be the zero-one matrix where $(S_C)_{ij} = 1$ if the ith column of A is chosen in the jth independent random trial, and $S_{ij} = 0$ otherwise; let us also define the associated rescaling matrix $D_C \in \mathbb{R}^{c \times c}$ to be the diagonal matrix with $(D_C)_{tt} = 1/\sqrt{cp_{i_t}}$, where i_t is the element of $\{1, \ldots, n\}$ chosen in the tth sampling trial. Let us similarly define $S_R \in \mathbb{R}^{r \times m}$ and $D_R \in \mathbb{R}^{r \times r}$ to be the row sampling matrix and associated diagonal rescaling matrix, respectively. In this notation,

$$C = AS_C D_C \qquad \text{and} \qquad R = D_R S_R A, \tag{12}$$

where $S_C D_C$ postmultiplies (and thus samples and rescales columns of) A to form C, and where $D_R S_R$ premultiplies (and thus samples and rescales rows of) A to form R. Thus, in this notation,

$$\Psi = D_R S_R C = D_R S_R A S_C D_C. \tag{13}$$

Given C, the LINEARTIMECUR algorithm computes the top k singular values, $\sigma_t^2(C)$, $t = 1, \ldots, k$, and the corresponding singular vectors, y^t, $t = 1, \ldots, k$, of $C^T C$. Note that these are also the squares of the singular values and the corresponding right

LINEARTIMECUR Algorithm.

Input: $A \in \mathbb{R}^{m \times n}$, $r, c, k \in \mathbb{Z}^+$ such that $1 \leq r \leq m$, $1 \leq c \leq n$, and $1 \leq k \leq \min(r, c)$, $\{p_i\}_{i=1}^m$ such that $p_i \geq 0$ and $\sum_{i=1}^m p_i = 1$, and $\{q_j\}_{j=1}^n$ such that $q_j \geq 0$ and $\sum_{j=1}^n q_j = 1$.

Output: $C \in \mathbb{R}^{m \times c}$, $U \in \mathbb{R}^{c \times r}$, and $R \in \mathbb{R}^{r \times n}$.

1. For $t = 1$ to c,
   (a) Pick $j_t \in \{1, \ldots, n\}$ with $\mathbf{Pr}[j_t = \alpha] = q_\alpha$, $\alpha = 1, \ldots, n$.
   (b) Set $C^{(t)} = A^{(j_t)} / \sqrt{cq_{j_t}}$.
2. Compute $C^T C$ and its SVD; say $C^T C = \sum_{t=1}^c \sigma_t^2(C) y^t y^{t^T}$.
3. If $\sigma_k(C) = 0$, then let $k = \max\{k' : \sigma_{k'}(C) \neq 0\}$.
4. For $t = 1$ to r,
   (a) Pick $i_t \in \{1, \ldots, m\}$ with $\mathbf{Pr}[i_t = \alpha] = p_\alpha$, $\alpha = 1, \ldots, m$.
   (b) Set $R_{(t)} = A_{(i_t)} / \sqrt{rp_{i_t}}$.
   (c) Set $\Psi_{(t)} = C_{(i_t)} / \sqrt{rp_{i_t}}$.
5. Let $\Phi = \sum_{t=1}^k \frac{1}{\sigma_t^2(C)} y^t y^{t^T}$ and let $U = \Phi \Psi^T$.
6. Return C, U, and R.

FIG. 1. *The* LINEARTIMECUR *algorithm.*

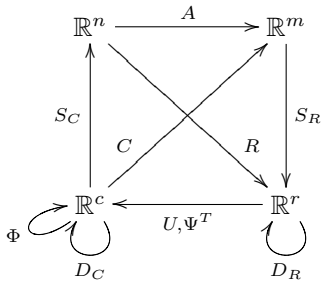


FIG. 2. *Diagram for the* LINEARTIMECUR *algorithm.*

singular vectors of C. Using these quantities, a matrix $\Phi \in \mathbb{R}^{c \times c}$ may be defined as

$$\Phi = \sum_{t=1}^{k} \frac{1}{\sigma_t^2(C)} y^t y^{t^T}, \tag{14}$$

from which $U \in \mathbb{R}^{c \times r}$ is constructed as $U = \Phi \Psi^T$. We could, of course, have defined a matrix Φ (and thus constructed a matrix U) from the singular vectors and singular values of RR^T in a manner analogous to that described above; in that case, the roles of the row sampling and column sampling would be reversed relative to the discussion below.

Figure 2 presents a diagram illustrating the action of the LINEARTIMECUR algorithm. The matrix A is shown as operating between the high-dimensional spaces $\mathbb{R}^n$ and $\mathbb{R}^m$. In addition, the matrix C is shown as operating between $\mathbb{R}^c$ and $\mathbb{R}^m$ and the matrix R is shown as operating between $\mathbb{R}^n$ and $\mathbb{R}^r$. Intuitively, one may think of $\mathbb{R}^c$ and $\mathbb{R}^r$ as being the most significant parts of $\mathbb{R}^n$ and $\mathbb{R}^m$, respectively, in terms of the action of A. Indeed, this will be the case when the sampling probabilities $\{p_i\}_{i=1}^m$

and $\{q_j\}_{j=1}^n$ satisfy certain conditions, as stated, e.g., in Theorem 4. The diagram also illustrates that $C = AS_CD_C$, $R = D_RS_RA$, and the matrix U which can be seen to be $U = \Phi\Psi^T = \Phi C^T(D_RS_R)^T$.

The matrix A is thus approximated by a matrix $A' = CUR$, where C is an $m \times c$ matrix consisting of c randomly chosen columns of A, R is an $r \times n$ matrix consisting of r randomly chosen rows of A, and $U = \Phi\Psi^T$ is a $c \times r$ matrix computed from C and R. As we shall see, if the column and row sampling probabilities are chosen judiciously, then c and r can be chosen to be constants (independent of m and n but depending on k and ϵ). Thus, (pictorially) we have that

$$\left(\begin{array}{c} \\ A \\ \\ \end{array}\right) \approx \left(\begin{array}{c} \\ C \\ \\ \end{array}\right) \left(\begin{array}{c} \\ U \\ \\ \end{array}\right) \left(\begin{array}{c} \\ R \\ \\ \end{array}\right). \tag{15}$$

The length of our succinct representation is $O(m+n)$. Note that if C and R are not explicitly needed, the length of the representation is a constant $O(1)$; this is because U is of constant size and only a constant number of bits are needed to specify which columns and rows of A are kept (along with their associated rescaling factors) in the construction of C and R, respectively.

Before proving the theorem, we would like to give some intuition as to why, if $\{p_i\}_{i=1}^m$ and $\{q_j\}_{j=1}^n$ satisfy certain conditions, the product CUR, as computed from the LinearTimeCUR algorithm, then becomes a good approximation to A in the sense of requirements (iv) and (v) of section 1. Let $h^t = Cy^t/\sigma_t(C)$ be the left singular vectors of C. Thus, if $H_k = (h^1\ h^2 \dots h^k) \in \mathbb{R}^{m\times k}$, then $H_kH_k^TA$ is the projection of A onto the subspace spanned by the top k left singular vectors of C. If the column sampling probabilities $\{q_j\}_{j=1}^n$ satisfy certain conditions, then the top k of the h^t's are approximations to the top k left singular vectors of A in the sense that their projection can be shown to "capture" *almost* as much of A as the projection of A onto the space spanned by its own top k left singular vectors. Indeed, the content of the SVD results of [11] (which in turn depend on the matrix multiplication results of [10]) is that if the column sampling probabilities $\{q_j\}_{j=1}^n$ are chosen judiciously, then the error in $\|A - H_kH_k^TA\|_\xi$ beyond the error for the best rank-k approximation can be made arbitrarily small both in expectation and with high probability.

Although $H_kH_k^TA$ is an approximation to A, one might wonder whether the approximation $H_kH_k^TA$ can be approximated so as to satisfy properties (i)–(v). Indeed, this is exactly what the CUR decomposition does! Using our sampling matrix formalism, let us define

$$\widetilde{H_k^T} = H_k^T(D_RS_R)^T \qquad \text{and} \qquad \widetilde{A} = D_RS_RA \tag{16}$$

to be the column-sampled and rescaled version of H_k^T and row-sampled and rescaled version of A, respectively. (We will see that $\widetilde{H_k^T}\widetilde{A} \approx H_k^TA$ by Theorem 1.) Lemma 1 states that

$$\begin{aligned} CUR &= H_kH_k^T(D_RS_R)^TD_RS_RA \\ &= H_k\widetilde{H_k^T}\widetilde{A}. \end{aligned} \tag{17}$$

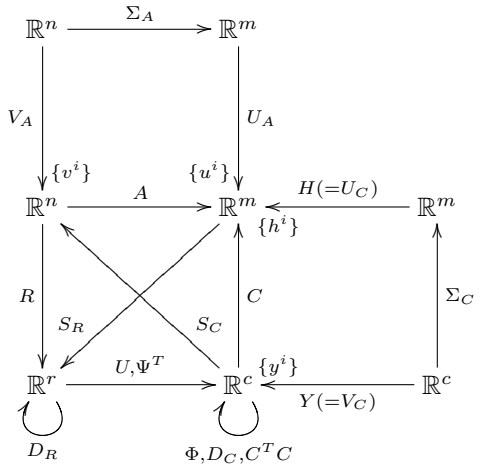


FIG. 3. *Another diagram for the* LINEARTIMECUR *algorithm.*

Thus, in order to provide a bound for $\|A - CUR\|_\xi$ for $\xi = 2, F$ we can first note that by the triangle inequality

$$\|A - CUR\|_\xi \leq \left\|A - H_k H_k^T A\right\|_\xi + \left\|H_k H_k^T A - CUR\right\|_\xi, \tag{18}$$

and then we can bound the two terms separately. The first term in (18) can be bounded using the SVD results of [11] if the column sampling probabilities satisfy certain conditions; in particular we will require that they be the optimal probabilities. Since $H_k^T H_k = I_k$, Lemma 2 states that

$$\begin{aligned} \left\|H_k H_k^T A - CUR\right\|_F &= \left\|H_k^T A - H_k^T (D_R S_R)^T D_R S_R A\right\|_F \\ &= \left\|H_k^T A - \widetilde{H_k^T} \widetilde{A}\right\|_F. \end{aligned} \tag{19}$$

Thus, the second term in (18) can be bounded by the matrix multiplication results of [10]; it will follow that if the sampling probabilities $\{p_i\}_{i=1}^m$ satisfy certain conditions, then $\widetilde{H_k^T} \widetilde{A} \approx H_k^T A$ in the sense that the error in $\|H_k^T A - \widetilde{H_k^T} \widetilde{A}\|_F$ can be bounded. Note that since the optimal probabilities depend on both H^T and A, and since we do not have access to H^T, our probabilities will not be optimal; nevertheless, although we will not obtain bounds with very high probability, as in [10] and [11], we will be able to apply Markov's inequality and thus achieve the bounds we desire.

A diagram illustrating the method (just described) that will be used to prove the correctness of the CUR algorithm is presented in Figure 3. In this figure, the locations of $\mathbb{R}^c$ and $\mathbb{R}^r$ and thus the directions of R, S_R, C, S_C, and U have been switched relative to their location in Figure 2. This presentation has several advantages: first, the SVD of C and the SVD of A can both be presented in the same figure as the CUR decomposition of A; second, one can see that bounding $\|A - H_k H_k^T A\|_\xi$ well in terms of $\|AA^T - CC^T\|_\xi$ depends on the probabilities used to sample the columns of A; and third, one can also see that bounding $\|H_k^T A - \widetilde{H_k^T} \widetilde{A}\|_F$ well depends on the probabilities used to sample the (columns of H_k^T and the corresponding) rows of A. See the corresponding figures in [10] and [11] for a comparison.

3.2. Analysis of the implementation and running time. In the LINEARTIMECUR algorithm the sampling probabilities $\{p_i\}_{i=1}^m$ and $\{q_j\}_{j=1}^n$ (if they are chosen to be of the form used in Theorems 4 and 5) can be computed in one pass and $O(c+r)$ additional space and time using the SELECT algorithm of [10]. Given the elements to be sampled, the matrix C can then be constructed in one additional pass; this requires additional space and time that is $O(mc)$. Similarly, the matrix R can then be constructed in the same pass using additional space and time that is $O(nr)$. Given $C \in \mathbb{R}^{m\times c}$, computing C^TC requires $O(mc)$ additional space and $O(mc^2)$ additional time, and computing the SVD of C^TC requires $O(c^3)$ additional time. The matrix Ψ can be computed in the same second pass by sampling the same r rows of C that were used to construct R from A; this requires additional space and time that is $O(cr)$. The matrix Φ can be explicitly computed using $O(c^2k)$ additional time, and then the matrix $U = \Phi\Psi^T$ can be computed using $O(c^2r)$ additional time. Thus, since c, r, and k are assumed to be a constant, overall $O(m+n)$ additional space and time are required by the LINEARTIMECUR algorithm, and requirements (i)–(iii) of section 1 are satisfied. Note that the "description" of the solution that is computable in the allotted additional space and time is the explicit matrices C, U, and R.

3.3. Analysis of the sampling step. Before stating and proving the main theorem of this section, we will first prove two useful lemmas. Lemma 1 will establish (17) and Lemma 2 will establish (19).

LEMMA 1.

$$CUR = H_k\widetilde{H_k^T}\widetilde{A}.$$

Proof. Note that the SVD of C is $C = \sum_{t=1}^c \sigma_t(C)h^t y^{t^T}$, that the matrix $\Psi = D_RS_RC$, and that $U = \Phi\Psi^T$, where Φ is given by (14). Thus, we have that

$$\begin{aligned} CUR &= C\left(\sum_{t=1}^k \frac{1}{\sigma_t^2(C)} y^t y^{t^T}\right) C^T (D_RS_R)^T R \\ &= \left(\sum_{t_1} \sigma_{t_1}(C) h^{t_1} y^{t_1 T}\right)\left(\sum_{t_2=1}^k \frac{1}{\sigma_{t_2}^2(C)} y^{t_2} y^{t_2 T}\right)\left(\sum_{t_3} \sigma_{t_3}(C) y^{t_3} h^{t_3 T}\right)(D_RS_R)^T R \\ &= \left(\sum_{t=1}^k h^t h^{t^T}\right)(D_RS_R)^T R. \end{aligned}$$

The lemma follows since $\sum_{t=1}^k h^t h^{t^T} = H_kH_k^T$, since $R = D_RS_RA$, and from the definitions (16). ∎

LEMMA 2.

$$\left\|H_kH_k^TA - CUR\right\|_F = \left\|H_k^TA - \widetilde{H_k^T}\widetilde{A}\right\|_F.$$

Proof. From Lemma 1 we have that $CUR = H_k\widetilde{H_k^T}\widetilde{A}$. Let us define the matrix $\Omega \in \mathbb{R}^{k\times n}$ as

$$\Omega = H_k^TA - H_k^T(D_RS_R)^TD_RS_RA = H_k^TA - \widetilde{H_k^T}\widetilde{A}.$$

In addition, note that

$$\left\|H_kH_k^TA - H_k\widetilde{H_k^T}\widetilde{A}\right\|_F^2 = \|H_k\Omega\|_F^2 = \mathbf{Tr}\left(\Omega^TH_k^TH_k\Omega\right).$$

The lemma follows since $H_k^TH_k = I_k$ and since $\mathbf{Tr}\left(\Omega^T\Omega\right) = \|\Omega\|_F^2$. ∎

Here is our main theorem regarding the LINEARTIMECUR algorithm described in section 3.1. Note that in this theorem we restrict ourselves to sampling probabilities that are optimal in the sense of section 2.3.

THEOREM 4. *Suppose $A \in \mathbb{R}^{m\times n}$, and let C, U, and R be constructed from the* LINEARTIMECUR *algorithm by sampling c columns of A with probabilities $\{q_j\}_{j=1}^n$ and r rows of A with probabilities $\{p_i\}_{i=1}^m$. Assume that $p_i = |A_{(i)}|^2 / \|A\|_F^2$ and $q_j = |A^{(j)}|^2 / \|A\|_F^2$. Then*

$$\mathbf{E}\left[\,\|A - CUR\|_F\right] \le \|A - A_k\|_F + \left(\left(\frac{4k}{c}\right)^{1/4} + \left(\frac{k}{r}\right)^{1/2}\right)\|A\|_F\,, \tag{20}$$

$$\mathbf{E}\left[\,\|A - CUR\|_2\right] \le \|A - A_k\|_2 + \left(\left(\frac{4}{c}\right)^{1/4} + \left(\frac{k}{r}\right)^{1/2}\right)\|A\|_F\,. \tag{21}$$

In addition, if we let $\eta_c = 1 + \sqrt{8\log(1/\delta_c)}$ and let $\delta = \delta_r + \delta_c$, then with probability at least $1-\delta$,

$$\|A - CUR\|_F \le \|A - A_k\|_F + \left(\left(\frac{4k\eta_c^2}{c}\right)^{1/4} + \left(\frac{k}{\delta_r^2 r}\right)^{1/2}\right)\|A\|_F\,, \tag{22}$$

$$\|A - CUR\|_2 \le \|A - A_k\|_2 + \left(\left(\frac{4\eta_c^2}{c}\right)^{1/4} + \left(\frac{k}{\delta_r^2 r}\right)^{1/2}\right)\|A\|_F\,. \tag{23}$$

Proof. By the triangle inequality we have that

$$\|A - CUR\|_\xi \le \left\|A - H_k H_k^T A\right\|_\xi + \left\|H_k H_k^T A - CUR\right\|_\xi \tag{24}$$

for both $\xi = 2, F$; thus by Lemmas 1 and 2 we have that

$$\|A - CUR\|_\xi \le \left\|A - H_k H_k^T A\right\|_\xi + \left\|H_k^T A - H_k^T (D_R S_R)^T D_R S_R A\right\|_F \tag{25}$$

$$= \left\|A - H_k H_k^T A\right\|_\xi + \left\|H_k^T A - \widetilde{H_k^T}\,\widetilde{A}\right\|_F \tag{26}$$

for both $\xi = 2, F$, where (26) follows from the definitions (16). Then, note that the column sampling satisfies the requirements for the LINEARTIMESVD algorithm of [11]. Thus, by Theorem 2 it follows from (25) that

$$\|A - CUR\|_F \le \|A - A_k\|_F + (4k)^{1/4}\left\|AA^T - CC^T\right\|_F^{1/2} + \left\|H_k^T A - \widetilde{H_k^T}\,\widetilde{A}\right\|_F, \tag{27}$$

$$\|A - CUR\|_2 \le \|A - A_k\|_2 + \sqrt{2}\left\|AA^T - CC^T\right\|_F^{1/2} + \left\|H_k^T A - \widetilde{H_k^T}\,\widetilde{A}\right\|_F. \tag{28}$$

Note that from the LINEARTIMECUR algorithm the column sampling probabilities are of the form (3) with $B = A^T$; thus, they are optimal and $\mathbf{E}\left[\|AA^T - CC^T\|_F\right] \le \frac{1}{\sqrt{c}}\|A\|_F^2$. In addition, although the row sampling probabilities are not optimal, they are of the form (6); thus, since $\|H_k^T\|_F = \sqrt{k}$ we have that

$$\mathbf{E}\left[\left\|H_k^T A - \widetilde{H_k^T}\,\widetilde{A}\right\|_F\right] \le \sqrt{\frac{k}{r}}\,\|A\|_F\,. \tag{29}$$

Thus, by taking expectations of (27) and (28), by using Jensen's inequality and Theorem 1, (20) and (21) follow.

To establish (22) and (23) first let the events $\mathcal{E}_\xi, \xi = c, r$ be defined as follows:

$$\mathcal{E}_c : \left\|AA^T - CC^T\right\|_F \leq \frac{\eta_c}{\sqrt{c}} \|A\|_F^2 ,$$

$$\mathcal{E}_r : \left\|H_k^T A - \widetilde{H_k^T}\widetilde{A}\right\|_F \leq \frac{1}{\delta_r}\sqrt{\frac{k}{r}} \|A\|_F .$$

Thus, from Theorem 1 we have that $\mathbf{Pr}[\mathcal{E}_c] \geq 1-\delta_c$. By applying Markov's inequality to $\left\|H_k^T A - \widetilde{H_k^T}\widetilde{A}\right\|_F$ and using (29) we see that

$$\mathbf{Pr}\left[\left\|H_k^T A - \widetilde{H_k^T}\widetilde{A}\right\|_F \geq \frac{1}{\delta_r}\sqrt{\frac{k}{r}} \|A\|_F\right] \leq \delta_r$$

and thus that $\mathbf{Pr}[\mathcal{E}_r] \geq 1 - \delta_r$. The theorem then follows from (27) and (28) by considering the event $\mathcal{E}_c \bigcap \mathcal{E}_r$. □

Note that in the proof of Theorem 4 (and similarly in that of Theorem 6 in section 4) $\|A - CUR\|_\xi$ is bounded by bounding each of the terms $\|A - H_k H_k^T A\|_\xi$ and $\|H_k H_k^T A - CUR\|_\xi$ independently. In order to bound $\|A - H_k H_k^T A\|_\xi$, the SVD results for arbitrary probabilities from [11] are used, and then it is noted that the column sampling probabilities used in the LINEARTIMECUR algorithm are optimal for bounding $\|AA^T - CC^T\|_F$. Then, independently, $\|H_k H_k^T A - CUR\|_\xi$ is bounded by using the matrix multiplication results of [10]. Since the probabilities that are used for the row sampling are not optimal with respect to bounding $\|H_k^T A - \widetilde{H_k^T}\widetilde{A}\|_F$, we do not obtain that bound with high probability. Due to the use of Markov's inequality, we must sample a number of rows that is $O(1/\delta)$, whereas we only need to sample a number of columns that is $O(\log(1/\delta))$.

As a corollary of Theorem 4 we have the following theorem. In this theorem, in addition to using sampling probabilities that are optimal in the sense of section 2.3, we choose sufficiently many columns and rows to ensure that the additional error is less than $\epsilon' \|A\|_F$.

THEOREM 5. *Suppose $A \in \mathbb{R}^{m\times n}$, and let C, U, and R be constructed from the* LINEARTIMECUR *algorithm by sampling c columns of A with probabilities $\{q_j\}_{j=1}^n$ and r rows of A with probabilities $\{p_i\}_{i=1}^m$. Assume that $p_i = |A_{(i)}|^2/\|A\|_F^2$ and $q_j = |A^{(j)}|^2/\|A\|_F^2$ and let $\epsilon, \epsilon' > 0$ with $\epsilon = \epsilon'/2$.*

If $c \geq 4k/\epsilon^4$ and $r \geq k/\epsilon^2$, then

$$\mathbf{E}[\|A - CUR\|_F] \leq \|A - A_k\|_F + \epsilon' \|A\|_F , \tag{30}$$

and if $c \geq 4/\epsilon^4$ and $r \geq k/\epsilon^2$, then

$$\mathbf{E}[\|A - CUR\|_2] \leq \|A - A_k\|_2 + \epsilon' \|A\|_F . \tag{31}$$

In addition, if we let $\eta_c = 1 + \sqrt{8\log(1/\delta_c)}$ and let $\delta = \delta_r + \delta_c$, and if $c \geq 4k\eta_c^2/\epsilon^4$ and $r \geq k/\delta_r^2\epsilon^2$, then with probability at least $1 - \delta$,

$$\|A - CUR\|_F \leq \|A - A_k\|_F + \epsilon' \|A\|_F , \tag{32}$$

and if $c \geq 4\eta_c^2/\epsilon^4$ and $r \geq k/\delta_r^2\epsilon^2$, then with probability at least $1 - \delta$,

$$\|A - CUR\|_2 \leq \|A - A_k\|_2 + \epsilon' \|A\|_F . \tag{33}$$

The results of Theorems 4 and 5 for both the Frobenius norm and the spectral norm hold for all k and are of particular interest when A is well approximated by a matrix of low rank since then one may choose $k = O(1)$ and obtain a good approximation. In addition, since $\|A - A_t\|_2 \leq \|A\|_F / \sqrt{t}$ for all $t = 1, 2, \ldots, r$, the bounds with respect to the spectral norm have the following interesting property: from (31) we can see that

$$\mathbf{E}\left[\|A - CUR\|_2\right] \leq \left(1/\sqrt{k} + \epsilon'\right) \|A\|_F,$$

and similarly for (33). Thus, under the assumptions of Theorem 5 if we choose $k = 1/\epsilon'^2$ and let $\epsilon'' = 2\epsilon'$, then we have that

$$\mathbf{E}\left[\|A - CUR\|_2\right] \leq \epsilon'' \|A\|_F \tag{34}$$

and that

$$\|A - CUR\|_2 \leq \epsilon'' \|A\|_F \tag{35}$$

holds with probability at least $1 - \delta$.

4. The constant time $C\tilde{U}R$ decomposition.

4.1. The algorithm. The ConstantTimeCUR algorithm is very similar in spirit to the LinearTimeCUR algorithm; thus, we only highlight its main features with an emphasis on similarities and differences between the two algorithms. Given a matrix $A \in \mathbb{R}^{m \times n}$, we wish to compute a description of a succinctly described, easily computed matrix A' that is decomposable as $A' = C\tilde{U}R \in \mathbb{R}^{m \times n}$ and that satisfies requirements (i)–(v) of section 1, where the additional RAM space and time to compute $\tilde{U}$ is $O(1)$. The ConstantTimeCUR algorithm, which is presented in Figure 4, accomplishes this by forming a matrix $C \in \mathbb{R}^{m \times c}$ by rescaling a randomly chosen subset of c columns of A, forming a matrix $R \in \mathbb{R}^{r \times n}$ by rescaling a randomly chosen subset of r rows of A, and forming a matrix $\Psi \in \mathbb{R}^{r \times c}$ from C by choosing the same randomly chosen rows used to construct R from A and rescaling appropriately. Given C, the ConstantTimeCUR algorithm randomly chooses and rescales w rows of C to form a matrix $W \in \mathbb{R}^{w \times c}$ and then computes the top ℓ singular values, $\sigma_t^2(W)$, $t = 1, \ldots, \ell$, and the corresponding singular vectors, z^t, $t = 1, \ldots, \ell$, of $W^T W$. Note that these are also approximations to the (squares of the) singular values and the corresponding right singular vectors of C. Using these quantities, a matrix $\tilde{\Phi} \in \mathbb{R}^{c \times c}$ may be defined as

$$\tilde{\Phi} = \sum_{t=1}^{\ell} \frac{1}{\sigma_t^2(W)} z^t z^{t^T}, \tag{36}$$

from which $\tilde{U} \in \mathbb{R}^{c \times r}$ is constructed as $\tilde{U} = \tilde{\Phi}\Psi^T$. Note that in the constant additional space and time framework the actual matrices C and R are not explicitly computed; instead the constant-sized matrix $\tilde{U}$ is computed and only a constant number of bits are stored to specify which columns and rows of A are kept (along with their associated rescaling factors) in the construction of C and R, respectively. Thus, the length of the succinct representation of A is a constant.

Figure 2 of section 3 provides a diagram illustrating the action of LinearTimeCUR algorithm, but the diagram and associated discussion are also relevant for the ConstantTimeCUR algorithm. Figure 5 also provides a diagram illustrating the

ConstantTimeCUR Algorithm.

Input: $A \in \mathbb{R}^{m \times n}$, $r, c, k \in \mathbb{Z}^+$ such that $1 \leq r \leq m$, $1 \leq c \leq n$, and $1 \leq k \leq \min(r, c)$, $\{p_i\}_{i=1}^m$ such that $p_i \geq 0$ and $\sum_{i=1}^m p_i = 1$, and $\{q_j\}_{j=1}^n$ such that $q_j \geq 0$ and $\sum_{j=1}^n q_j = 1$.

Output: $\tilde{U} \in \mathbb{R}^{c \times r}$ and a "description" of $C \in \mathbb{R}^{m \times c}$ and $R \in \mathbb{R}^{r \times n}$.

1. For $t = 1$ to c,
   (a) Pick $j_t \in \{1, \ldots, n\}$ with $\mathbf{Pr}[j_t = \alpha] = q_\alpha$, and save $\{(j_t, q_{j_t}) : t = 1, \ldots, c\}$.
   (b) Set $C^{(t)} = A^{(j_t)}/\sqrt{cq_{j_t}}$. (Note that C is not explicitly constructed in RAM.)
2. Choose $\{\pi_i\}_{i=1}^m$ such that $\pi_i = \left|C_{(i)}\right|^2 / \|C\|_F^2$.
3. For $t = 1$ to w,
   (a) Pick $i_t \in 1, \ldots, m$ with $\mathbf{Pr}[i_t = \alpha] = \pi_\alpha$, $\alpha = 1, \ldots, m$.
   (b) Set $W_{(t)} = C_{(i_t)}/\sqrt{w\pi_{i_t}}$.
4. Compute $W^T W$ and its SVD; say $W^T W = \sum_{t=1}^c \sigma_t^2(W) z^t z^{t^T}$.
5. If a $\|\cdot\|_F$ bound is desired, set $\gamma = \epsilon/100k$,
   Else if a $\|\cdot\|_2$ bound is desired, set $\gamma = \epsilon/100$.
6. Let $\ell = \min\{k, \max\{t : \sigma_t^2(W) \geq \gamma \|W\|_F^2\}\}$.
7. Keep singular values $\{\sigma_t(W)\}_{t=1}^\ell$ and their corresponding singular vectors $\{z^t\}_{t=1}^\ell$.
8. For $t = 1$ to r,
   (a) Pick $i_t \in \{1, \ldots, m\}$ with $\mathbf{Pr}[i_t = \alpha] = p_\alpha$, and save $\{(i_t, p_{i_t}) : t = 1, \ldots, r\}$.
   (b) Set $R_{(t)} = A_{(i_t)}/\sqrt{rp_{i_t}}$. (Note that R is not explicitly constructed in RAM.)
   (c) Set $\Psi_{(t)} = C_{(i_t)}/\sqrt{rp_{i_t}}$.
9. Let $\tilde{\Phi} = \sum_{t=1}^\ell \frac{1}{\sigma_t^2(W)} z^t z^{t^T}$ and let $\tilde{U} = \tilde{\Phi}\Psi^T$.
10. Return $\tilde{U}$, c column labels $\{(j_t, q_{j_t}) : t = 1, \ldots, c\}$, and r row labels $\{(i_t, p_{i_t}) : t = 1, \ldots, r\}$.

Fig. 4. *The* ConstantTimeCUR *algorithm.*

action of the ConstantTimeCUR algorithm, and is the analogue for the constant time $C\tilde{U}R$ of Figure 3; Figure 5 also illustrates that the matrix Y $(= V_C)$ (of Figure 3) consisting of the top k right singular vectors of C is not exactly computed, but is instead approximated by the matrix Z $(= V_W)$, where Z is a matrix whose columns $Z^{(t)} = z^t$ consist of the right singular vectors of W. Thus, the matrix H_k consisting of the left singular vectors of C is not exactly computed but is only approximated by $\tilde{H}_\ell$, where $\tilde{H}_\ell^{(t)} = \tilde{h}^t = Cz^t/\sigma_t(W)$ for $t = 1, \ldots, \ell$. Since by construction it is still the case that $R = D_R S_R A$ (and also that $C = AS_C D_C$ and $\Psi = D_R S_R C$), where the sampling matrices and the diagonal rescaling matrices are defined as in section 3.1, it follows from Lemma 3 that

$$C\tilde{U}R = \tilde{H}_\ell \tilde{H}_\ell^T (D_R S_R)^T D_R S_R A. \tag{37}$$

In the linear time case we had that $H_k^T H_k = I_k$ since the columns of H_k were k of the left singular vectors of C. This allowed us to prove Lemma 2, as described

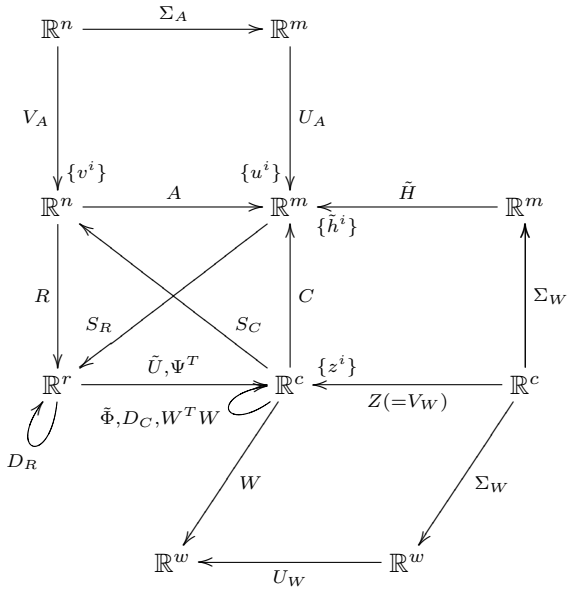


FIG. 5. *Diagram for the* CONSTANTTIMECUR *algorithm.*

in section 3.1. In the constant additional time setting, the columns of $\tilde{H}_\ell$ are only approximations to k of the left singular vectors of C, but we still have that $\tilde{H}_\ell^T \tilde{H}_\ell \approx I_\ell$. To quantify this, define $Z_{\alpha,\beta} \in \mathbb{R}^{c \times (\beta-\alpha+1)}$ to be the matrix whose columns are the αth through the βth singular vectors of $W^T W$ and $T \in \mathbb{R}^{\ell \times \ell}$ to be the diagonal matrix with elements $T_{tt} = 1/\sigma_t(W)$. If we define the matrix $\Delta \in \mathbb{R}^{\ell \times \ell}$ to be

$$\Delta = T Z_{1,\ell}^T (C^T C - W^T W) Z_{1,\ell} T, \tag{38}$$

then Lemma 5 will establish that

$$\left\| \tilde{H}_\ell \tilde{H}_\ell^T A - C\tilde{U}R \right\|_F \leq (1 + \|\Delta\|_F^{1/2}) \left\| \tilde{H}_\ell^T A - \tilde{H}_\ell^T (D_R S_R)^T D_R S_R A \right\|_F, \tag{39}$$

and Lemma 4 establishes that $\tilde{H}_\ell^T \tilde{H}_\ell = I_\ell + \Delta$. Thus, as in the linear time case, we can split

$$\left\| A - C\tilde{U}R \right\|_\xi \leq \left\| A - \tilde{H}_\ell \tilde{H}_\ell^T A \right\|_\xi + \left\| \tilde{H}_\ell \tilde{H}_\ell^T A - C\tilde{U}R \right\|_\xi$$

for $\xi = 2, F$ and then bound the two terms separately. The first term can be bounded by the constant time SVD results of [11], the second term can be bounded by the matrix multiplication results of [10], and bounding the overall error depends on both results. Due to the two levels of sampling, the additional error will be larger than in the linear additional time framework, but it can be made arbitrarily small by choosing a constant number of rows and columns.

4.2. Analysis of the implementation and running time. In the CONSTANTTIMECUR algorithm the sampling probabilities $\{p_i\}_{i=1}^m$ and $\{q_j\}_{j=1}^n$ (if they are chosen to be of the form used in Theorem 6) can be computed in one pass and $O(c + r)$ additional space and time using the SELECT algorithm of [10]. Given the columns of A to be sampled, we do not explicitly construct the matrix C but instead

perform a second level of sampling and select w rows of C with probabilities $\{\pi_i\}_{i=1}^m$ in order to construct the matrix W; this requires a second pass and $O(w)$ additional space and time. Then, in a third pass we explicitly construct W; this requires additional space and time that is $O(cw)$. Similarly, a description of the matrix R can then be constructed in the same third pass using additional space and time that is $O(r)$. Then, given W, computing W^TW requires $O(c^2w)$ additional time and computing the SVD of W^TW requires $O(c^3)$ additional time. The matrix Ψ can be computed in the same third pass by sampling the same r rows of C that were used to construct R from A; this requires additional time that is $O(cr)$. The matrix $\tilde{\Phi}$ can be explicitly computed using $O(c^2k)$ additional time and then the matrix $U = \tilde{\Phi}\Psi^T$ can be computed using $O(c^2r)$ additional time. Thus, since c, r, and k are assumed to be constants, overall $O(1)$ additional space and time are required by the ConstantTimeCUR algorithm, and requirements (i)–(iii) of section 1 are satisfied. Note that the "description" of the solution that is computable in the allotted additional space and time is the matrix $\tilde{U}$, with the labels $i_1, \ldots, i_r$ and $j_1, \ldots, j_c$ indicating the rows chosen to construct C and R as well as the corresponding probabilities $\{p_{i_t}\}_{t=1}^r$ and $\{q_{j_t}\}_{t=1}^c$; we note that we need to know p_i only for the sampled rows i and q_j only for the sampled columns j.

4.3. Analysis of the sampling step. Before stating and proving the main theorem of this section, we will first prove several useful lemmas. First, in Lemma 3 we will establish (37).

Lemma 3.

$$C\tilde{U}R = \tilde{H}_\ell \tilde{H}_\ell^T (D_R S_R)^T D_R S_R A.$$

Proof. Since $\tilde{h}^t = Cz^t/\sigma_t(W)$ for $t = 1, \ldots, \ell$, we have that $C = \sum_{t=1}^c \sigma_t(W)\tilde{h}^t z^{t^T}$. Thus, we have that

$$\begin{aligned} C\tilde{U}R &= C\left(\sum_{t=1}^{\ell} \frac{1}{\sigma_t^2(W)} z^t z^{t^T}\right) C^T (D_R S_R)^T R \\ &= \left(\sum_{t=1}^{\ell} \tilde{h}^t \tilde{h}^{t^T}\right) (D_R S_R)^T R. \end{aligned}$$

The lemma follows since $\sum_{t=1}^{\ell} \tilde{h}^t \tilde{h}^{t^T} = \tilde{H}_\ell \tilde{H}_\ell^T$ and since $R = D_R S_R A$. ∎

Next, Lemma 4 will characterize, in terms of Δ, the degree to which the columns of $\tilde{H}_\ell$ are not orthonormal. Note that it appeared in [11].

Lemma 4. *When written in the basis with respect to Z,*

$$\tilde{H}_\ell^T \tilde{H}_\ell = I_\ell + \Delta.$$

Furthermore, for $\xi = 2, F$,

$$\|\Delta\|_\xi \leq \frac{1}{\gamma \|W\|_F^2} \left\|C^T C - W^T W\right\|_\xi.$$

Proof. Recall that $\tilde{H}_\ell = CZ_{1,\ell}T$ and that $T^T Z_{1,\ell}^T W^T W Z_{1,\ell} T = I_\ell$, so that

$$\left\|\tilde{H}_\ell^T \tilde{H}_\ell - I_\ell\right\|_\xi = \left\|T^T Z_{1,\ell}^T C^T C Z_{1,\ell} T - T^T Z_{1,\ell}^T W^T W Z_{1,\ell} T\right\|_\xi \tag{40}$$

$$= \left\|T^T Z_{1,\ell}^T \left(C^T C - W^T W\right) Z_{1,\ell} T\right\|_\xi. \tag{41}$$

Using the submultiplicativity properties of the 2-norm, and in particular

$$\|AB\|_\xi \leq \|A\|_2 \|B\|_\xi , \tag{42}$$

$$\|AB\|_\xi \leq \|A\|_\xi \|B\|_2 , \tag{43}$$

for both $\xi = 2, F$, we get

$$\left\|\tilde{H}_\ell^T \tilde{H}_\ell - I_\ell\right\|_\xi \leq \left\|T^T Z_{1,\ell}^T\right\|_2 \left\|C^T C - W^T W\right\|_\xi \|Z_{1,\ell} T\|_2 \tag{44}$$

$$\leq \|T\|_2^2 \left\|C^T C - W^T W\right\|_\xi \tag{45}$$

$$\leq \max_{t=1,\ldots,\ell} \left(1/\sigma_t^2(W)\right) \left\|C^T C - W^T W\right\|_\xi , \tag{46}$$

since $\|Z_{1,\ell}\|_2 = 1$. The lemma follows since $\sigma_t^2(W) \geq \gamma \|W\|_F^2$ for all $t = 1, \ldots, \ell$ by the definition of ℓ. □

Next, in Lemma 5 we will establish (39).

LEMMA 5.

$$\left\|\tilde{H}_\ell \tilde{H}_\ell^T A - C\tilde{U}R\right\|_F \leq (1 + \|\Delta\|_F^{1/2}) \left\|\tilde{H}_\ell^T A - \tilde{H}_\ell^T (D_R S_R)^T D_R S_R A\right\|_F .$$

Proof. From Lemma 3 we have that $C\tilde{U}R = \tilde{H}_\ell \tilde{H}_\ell^T (D_R S_R)^T D_R S_R A$. Let us define the matrix $\Omega \in \mathbb{R}^{\ell \times n}$ as

$$\Omega = \tilde{H}_\ell^T A - \tilde{H}_\ell^T (D_R S_R)^T D_R S_R A.$$

Thus, since $\mathbf{Tr}\left(XX^T\right) = \|X\|_F^2$ for a matrix X, we have

$$\begin{aligned} \left\|\tilde{H}_\ell \Omega\right\|_F^2 &= \mathbf{Tr}(\Omega^T \tilde{H}_\ell^T \tilde{H}_\ell \Omega) \\ &= \mathbf{Tr}(\Omega^T (I_\ell + \Delta)\Omega) \qquad (47) \\ &= \|\Omega\|_F^2 + \mathbf{Tr}(\Omega^T \Delta \Omega) \\ &\leq \|\Omega\|_F^2 + \|\Delta\|_2 \|\Omega\|_F^2 , \qquad (48) \end{aligned}$$

where (47) follows from Lemma 4 and (48) follows since $\left|\mathbf{Tr}(\Omega^T \Delta \Omega)\right| \leq \|\Delta\|_2 \mathbf{Tr}(\Omega^T \Omega)$. Thus,

$$\left\|\tilde{H}_\ell \tilde{H}_\ell^T A - C\tilde{U}R\right\|_F \leq (1 + \|\Delta\|_2)^{1/2} \left\|\tilde{H}_\ell^T A - \tilde{H}_\ell^T (D_R S_R)^T D_R S_R A\right\|_F ,$$

and the lemma then follows. □

Finally, in Lemma 6 we show that $\|W\|_F = \|C\|_F = \|A\|_F$ when optimal probabilities are used. It also appeared in [11].

LEMMA 6. *Suppose $A \in \mathbb{R}^{m \times n}$, and run the* ConstantTimeCUR *algorithm by sampling c columns of A with probabilities $\{q_j\}_{j=1}^n$ (and then sampling w rows of C with probabilities $\{\pi_i\}_{i=1}^m$ to construct W) and r rows of A with probabilities $\{p_i\}_{i=1}^m$. Assume that $p_i = |A_{(i)}|^2 / \|A\|_F^2$ and $q_j = |A^{(j)}|^2 / \|A\|_F^2$. Then $\|W\|_F = \|C\|_F = \|A\|_F$.*

Proof. If $p_i = |A^{(i)}|^2 / \|A\|_F^2$, then we have that $\|C\|_F^2 = \sum_{t=1}^c |C^{(t)}|^2 = \sum_{t=1}^c \frac{|A^{(i_t)}|^2}{c p_{i_t}} = \|A\|_F^2$. Similarly, if $q_j = |C_{(j)}|^2 / \|C\|_F^2$, then we have that $\|W\|_F^2 = \sum_{t=1}^w \left|W_{(t)}\right|^2 = \sum_{t=1}^w \frac{|C_{(i_t)}|^2}{w q_{i_t}} = \|C\|_F^2$. The lemma follows. □

Here is our main theorem regarding the ConstantTimeCUR algorithm described in section 4.1. It is the constant time analogue of Theorem 5. Note that

in this theorem we restrict ourselves to sampling probabilities that are optimal in the sense of section 2.3, and to choosing sufficiently many columns and rows to ensure that the additional error is less than $\epsilon \|A\|_F$.

THEOREM 6. *Suppose $A \in \mathbb{R}^{m\times n}$, and let C, $\tilde{U}$, and R be constructed from the* CONSTANTTIMECUR *algorithm by sampling c columns of A with probabilities $\{q_j\}_{j=1}^n$ (and then sampling w rows of C with probabilities $\{\pi_i\}_{i=1}^m$ to construct W) and r rows of A with probabilities $\{p_i\}_{i=1}^m$. Assume that $p_i = |A_{(i)}|^2/\|A\|_F^2$ and $q_j = |A^{(j)}|^2/\|A\|_F^2$. Let $\eta = 1+\sqrt{8\log(3/\delta)}$ and $\epsilon > 0$.*

If a Frobenius norm bound is desired, and hence the CONSTANTTIMESVD *algorithm is run with $\gamma = \epsilon/100k$, then if we let $c = \Omega(k^2\eta^2/\epsilon^8)$, $w = \Omega(k^2\eta^2/\epsilon^8)$, and $r = \Omega(k/\delta^2\epsilon^2)$, then with probability at least $1-\delta$,*

$$\left\|A - C\tilde{U}R\right\|_F \leq \|A - A_k\|_F + \epsilon \|A\|_F . \tag{49}$$

If a spectral norm bound is desired, and hence the CONSTANTTIMESVD *algorithm is run with $\gamma = \epsilon/100$, then if we let $c = \Omega(\eta^2/\epsilon^8)$, $w = \Omega(\eta^2/\epsilon^8)$, and $r = \Omega(k/\delta^2\epsilon^2)$, then with probability at least $1-\delta$,*

$$\left\|A - C\tilde{U}R\right\|_2 \leq \|A - A_k\|_2 + \epsilon \|A\|_F . \tag{50}$$

Proof. Let us define the events:

$$\mathcal{E}_c : \left\|AA^T - CC^T\right\|_F \leq \frac{\eta}{\sqrt{c}} \|A\|_F^2 , \tag{51}$$

$$\mathcal{E}_w : \left\|C^TC - W^TW\right\|_F \leq \frac{\eta}{\sqrt{w}} \|A\|_F^2 , \tag{52}$$

$$\mathcal{E}_r : \left\|\tilde{H}_\ell^T A - \tilde{H}_\ell^T (D_R S_R)^T D_R S_R A\right\|_F \leq \frac{3}{\delta\sqrt{r}} \left\|\tilde{H}_\ell\right\|_F \|A\|_F . \tag{53}$$

Under the assumptions of this theorem, event $\mathcal{E}_c$ holds with probability greater than $1-\delta/3$ by Theorem 1, and similarly for event $\mathcal{E}_w$. Next, we claim that $\mathcal{E}_r$ holds with probability greater than $1-\delta/3$. To prove this it suffices to prove that

$$\mathbf{E}\left[\left\|\tilde{H}_\ell^T A - \tilde{H}_\ell^T (D_R S_R)^T D_R S_R A\right\|_F\right] \leq \frac{1}{\sqrt{r}} \left\|\tilde{H}_\ell^T\right\|_F \|A\|_F , \tag{54}$$

since the claim that $\mathbf{Pr}[\mathcal{E}_r] \geq 1-\delta/3$ follows immediately from (54) by Markov's inequality; but (54) follows from Theorem 1 since the probabilities $\{p_i\}_{i=1}^m$ used to sample the columns of $\tilde{H}_\ell$ and the corresponding rows of A are of the form (6). Thus, under the assumptions of the theorem,

$$\mathbf{Pr}[\mathcal{E}_\xi] \geq 1-\delta/3 \quad \text{for } \xi = c, w, r.$$

Next, from Lemma 4 and the Cauchy–Schwarz inequality, it follows that

$$\left\|\tilde{H}_\ell\right\|_F^2 = \sum_{t=1}^{\ell} \left|\tilde{h}^{t^T} \tilde{h}^t\right| = \sum_{t=1}^{\ell} 1 + \Delta_{tt} \leq k + \sqrt{k} \|\Delta\|_F . \tag{55}$$

Since $\sqrt{1+x} \leq 1 + \sqrt{x}$ for $x \geq 0$ it follows from (55) and (53) that under the event $\mathcal{E}_r$ we have

$$\left\|\tilde{H}_\ell^T A - \tilde{H}_\ell^T (D_R S_R)^T D_R S_R A\right\|_F \leq \frac{3}{\delta\sqrt{r}} \left(\sqrt{k} + k^{1/4} \|\Delta\|_F^{1/2}\right) \|A\|_F . \tag{56}$$

By combining (56) with Lemma 5 we have that

$$\begin{aligned}
\left\|\tilde{H}_\ell \tilde{H}_\ell^T A - C\tilde{U}R\right\|_F &\leq \frac{3}{\delta\sqrt{r}} \left(1 + \|\Delta\|_F^{1/2}\right) \left(\sqrt{k} + k^{1/4} \|\Delta\|_F^{1/2}\right) \|A\|_F \\
&\leq \left(\frac{9k}{\delta^2 r}\right)^{1/2} \left(1 + \|\Delta\|_F^{1/2}\right)^2 \|A\|_F \qquad (57) \\
&\leq \left(\frac{9k}{\delta^2 r}\right)^{1/2} \left(1 + 3\|\Delta\|_F^{1/2}\right) \|A\|_F \qquad (58) \\
&\leq \left(\frac{9k}{\delta^2 r}\right)^{1/2} \left(1 + \frac{3\left\|C^T C - W^T W\right\|_F^{1/2}}{\sqrt{\gamma}\|W\|_F}\right) \|A\|_F , \qquad (59)
\end{aligned}$$

where (57) follows since $k^{1/4} \leq \sqrt{k}$, (58) follows by multiplying out terms and since $\|\Delta\|_F \leq 1$ under the assumptions of the theorem, and (59) follows from the bound on $\|\Delta\|_F$ in Lemma 4.

Let us first consider establishing the Frobenius norm bound of (49). Recall that in this case we have set $\gamma = \epsilon/100k$. We will use the triangle inequality to get

$$\left\|A - C\tilde{U}R\right\|_F \leq \left\|A - \tilde{H}_\ell \tilde{H}_\ell^T A\right\|_F + \left\|\tilde{H}_\ell \tilde{H}_\ell^T A - C\tilde{U}R\right\|_F \tag{60}$$

and will bound each term separately. First, using the probabilities $\{q_j\}_{j=1}^n$ and the values of $c, w = \Omega(k^2\eta^2/\epsilon^8)$, then under the event $\mathcal{E}_c \bigcap \mathcal{E}_w$ we have that

$$\left\|A - \tilde{H}_\ell \tilde{H}_\ell^T A\right\|_F \leq \|A - A_k\|_F + \frac{\epsilon}{2}\|A\|_F \tag{61}$$

by Theorem 3; see also [11]. In addition, using the sampling probabilities $\{p_i\}_{i=1}^m$ and values of $r = \Omega(k/\delta^2\epsilon^2)$ and $w = \Omega(k^2\eta^2/\epsilon^8)$, and noting Lemma 6, it follows from (59) that under the event $\mathcal{E}_r \bigcap \mathcal{E}_w$,

$$\left\|\tilde{H}_\ell \tilde{H}_\ell^T A - C\tilde{U}R\right\|_F \leq \frac{\epsilon}{2}\|A\|_F . \tag{62}$$

Thus, under the event $\mathcal{E}_c \bigcap \mathcal{E}_w \bigcap \mathcal{E}_r$, which has probability at least $1-\delta$, by combining (61) and (62) we see that (49) follows.

Let us next consider establishing the spectral norm bound of (50). Recall that in this case we have set $\gamma = \epsilon/100$ and that

$$\left\|A - C\tilde{U}R\right\|_2 \leq \left\|A - \tilde{H}_\ell \tilde{H}_\ell^T A\right\|_2 + \left\|\tilde{H}_\ell \tilde{H}_\ell^T A - C\tilde{U}R\right\|_F \tag{63}$$

by the submultiplicitivity of $\|\cdot\|_2$ and since $\|\cdot\|_2 \leq \|\cdot\|_F$. First, using the probabilities $\{q_j\}_{j=1}^n$ and the values of $c, w = \Omega(\eta^2/\epsilon^8)$, under the event $\mathcal{E}_c \bigcap \mathcal{E}_w$ we have that

$$\left\|A - \tilde{H}_\ell \tilde{H}_\ell^T A\right\|_2 \leq \|A - A_k\|_2 + \frac{\epsilon}{2}\|A\|_F \tag{64}$$

by Theorem 3; see also [11]. In addition, using the sampling probabilities $\{p_i\}_{i=1}^m$ and values of $r = \Omega(k/\delta^2\epsilon^2)$ and $w = \Omega(\eta^2/\epsilon^8)$, and noting Lemma 6, it follows from (59) that under the event $\mathcal{E}_r \bigcap \mathcal{E}_w$,

$$\left\|\tilde{H}_\ell \tilde{H}_\ell^T A - C\tilde{U}R\right\|_F \leq \frac{\epsilon}{2} \|A\|_F . \tag{65}$$

Thus, under the event $\mathcal{E}_c \bigcap \mathcal{E}_w \bigcap \mathcal{E}_r$, which has probability at least $1-\delta$, by combining (64) and (65) we see that (50) follows. □

As in the linear additional time framework, the results of Theorem 6 hold for all k and are of particular interest when A is well approximated by a matrix of low rank since then one may choose $k = O(1)$ and obtain a good approximation. In addition, it follows from (50) that

$$\left\|A - C\tilde{U}R\right\|_2 \leq (1/\sqrt{k} + \epsilon') \|A\|_F .$$

Thus, under the assumptions of Theorem 6 if we choose $k = 1/\epsilon'^2$ and let $\epsilon'' = 2\epsilon'$, then we have that

$$\|A - CUR\|_2 \leq \epsilon'' \|A\|_F \tag{66}$$

holds with probability at least $1 - \delta$.

We note the following lemma. This result is not needed in this paper, but is included for future reference [13].

LEMMA 7.

$$\left\|\tilde{U}\right\|_2 \leq \frac{O(1)}{\gamma \|A\|_F}.$$

Proof. First note that $\|\tilde{U}\|_2 = \|\tilde{\Phi}\Psi^T\|_2 \leq \|\tilde{\Phi}\|_2 \|\Psi^T\|_2$. Since $\tilde{\Phi} = \sum_{t=1}^{\ell} \frac{1}{\sigma_t^2(W)} z^t z^{tT}$, we see that $\|\tilde{\Phi}\|_2 = \frac{1}{\sigma_\ell^2(W)} \leq \frac{1}{\gamma \|W\|_F^2}$. The lemma follows since, when using the probabilities of Theorem 6, we have that $\|\Psi^T\|_F \leq O(1)\|A\|_F$ and that $1/\|W\|_F^2 \leq O(1)/\|A\|_F^2$. □

5. Discussion and conclusion. To put the CUR decomposition into context, it will be useful to contrast it with the SVD. The SVD of A expresses A as $A = \sum_{t=1}^{\rho} \sigma_t(A) u^t v^{tT}$. Keeping the first k terms of this expansion, i.e., keeping $A_k = \sum_{t=1}^{k} \sigma_t(A) u^t v^{tT} = U_k \Sigma_k V_k^T$, gives us the "optimal" rank-k approximation to A with respect to both the spectral norm and the Frobenius norm [19, 22]. Thus, computing the SVD gives us a good succinct approximation, since it only takes space $O(k(m+n))$ to write down U_k, Σ_k, V_k. However, the computational problem of finding the SVD cannot be carried out in a small constant number of passes. Our theorems say that weaker bounds, which are similar in spirit, may be achieved by CUR. Note, however, that although the SVD may be thought of as a rotation followed by a rescaling followed by a rotation, the CUR decomposition is quite different; both C and R perform an action like A and thus U must involve a pseudoinverse-like operation. Note also that the last upper bound, i.e., (v), is much smaller than $\|A\|_F$ when A has a good low-rank approximation. This is indeed the case for matrices occurring in many contexts, such as matrices for which principal component analysis is used.

Recent work has focused on developing new techniques for proving lower bounds on the number of queries a sampling algorithm is required to perform in order to

approximate a given function accurately with a low probability or error [3, 4]. In [4] these methods have been applied to the low-rank matrix approximation problem and to the matrix reconstruction problem. In the latter problem, the input is a matrix $A \in \mathbb{R}^{m \times n}$ and the goal is to find a matrix B that is close to A. In [4] it is shown that finding a B such that $\|A - B\|_F \leq \epsilon \|A\|_F$ requires $\Omega(mn)$ queries and that finding a B such that $\|A - B\|_2 \leq \epsilon \|A\|_F$ requires $\Omega(m + n)$ queries. Thus, our algorithm is optimal for constant ϵ.

During the time since this manuscript was submitted for journal publication we have become aware of other low-rank matrix decompositions of the form $A \approx CUR$, where C is a matrix consisting of a small number of columns of A, R is a matrix consisting of a small number of rows of A, and U is an appropriately defined low-dimensional matrix. Work by Stewart [26, 27, 5] and independently work by Goreinov, Tyrtyshnikov, and Zamarashkin [21, 20] have considered matrix decompositions with structural (but not algorithmic) properties quite similar to the CUR decompositions we have considered in this paper. Drineas and Mahoney have extended the CUR decompositions of this paper to kernel-based statistical learning [15, 16] and large tensor-based data [17]. These latter papers also contain a discussion of the relationship between the work presented in this paper and the related work of Stewart and Goreinov, Tyrtyshnikov, and Zamarashkin.

Acknowledgments. We would like to thank Dimitris Achlioptas for fruitful discussions and the National Science Foundation for partial support of this work. In addition, we would like to thank an anonymous reviewer for carefully reading the paper and making numerous useful suggestions; in particular, the reviewer provided an elegant, short proof for Lemma 4.

REFERENCES

[1] D. Achlioptas and F. McSherry, *Fast computation of low rank matrix approximations*, J. ACM, to appear.

[2] D. Achlioptas and F. McSherry, *Fast computation of low rank matrix approximations*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, 2001, pp. 611–618.

[3] Z. Bar-Yossef, *The Complexity of Massive Data Set Computations*, Ph.D. thesis, University of California, Berkeley, 2002.

[4] Z. Bar-Yossef, *Sampling lower bounds via information theory*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, 2003, pp. 335–344.

[5] M. W. Berry, S. A. Pulatova, and G. W. Stewart, *Computing Sparse Reduced-Rank Approximations to Sparse Matrices*, Tech. Report UMIACS TR-2004-32 CMSC TR-4589, University of Maryland, College Park, MD, 2004.

[6] R. Bhatia, *Matrix Analysis*, Springer-Verlag, New York, 1997.

[7] E. Cohen and D. D. Lewis, *Approximating matrix multiplication for pattern recognition tasks*, J. Algorithms, 30 (1999), pp. 211–252.

[8] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay, *Clustering in large graphs and matrices*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1999, pp. 291–299.

[9] P. Drineas and R. Kannan, *Pass efficient algorithms for approximating large matrices*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2003, pp. 223–232.

[10] P. Drineas, R. Kannan, and M. W. Mahoney, *Fast Monte Carlo algorithms for matrices* I: *Approximating matrix multiplication*, SIAM J. Comput., 36 (2006), pp. 132–157.

[11] P. Drineas, R. Kannan, and M. W. Mahoney, *Fast Monte Carlo algorithms for matrices* II: *Computing a low-rank approximation to a matrix*, SIAM J. Comput., 36 (2006), pp. 158–183.

[12] P. Drineas, R. Kannan, and M. W. Mahoney, *Fast Monte Carlo Algorithms for Matrices* III: *Computing a Compressed Approximate Matrix Decomposition*, Tech. Report

YALEU/DCS/TR-1271, Department of Computer Science, Yale University, New Haven, CT, 2004.

[13] P. Drineas, R. Kannan, and M. W. Mahoney, *Sampling Sub-problems of Heterogeneous Max-Cut Problems and Approximation Algorithms*, Tech. Report YALEU/DCS/TR-1283, Department of Computer Science, Yale University, New Haven, CT, 2004.

[14] P. Drineas, I. Kerenidis, and P. Raghavan, *Competitive recommendation systems*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, 2002, pp. 82–90.

[15] P. Drineas and M. W. Mahoney, *Approximating a Gram matrix for improved kernel-based learning*, in Proceedings of the 18th Annual Conference on Learning Theory, 2005, pp. 323–337.

[16] P. Drineas and M. W. Mahoney, *On the Nyström method for approximating a Gram matrix for improved kernel-based learning*, J. Machine Learning, 6 (2005), pp. 2153–2175.

[17] P. Drineas and M. W. Mahoney, *A Randomized Algorithm for a Tensor-Based Generalization of the Singular Value Decomposition*, Tech. Report YALEU/DCS/TR-1327, Department of Computer Science, Yale University, New Haven, CT, 2005.

[18] A. Frieze, R. Kannan, and S. Vempala, *Fast Monte-Carlo algorithms for finding low-rank approximations*, in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, 1998, pp. 370–378.

[19] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, 1989.

[20] S. A. Goreinov and E. E. Tyrtyshnikov, *The maximum-volume concept in approximation by low-rank matrices*, Contemp. Math., 280 (2001), pp. 47–51.

[21] S. A. Goreinov, E. E. Tyrtyshnikov, and N. L. Zamarashkin, *A theory of pseudoskeleton approximations*, Linear Algebra Appl., 261 (1997), pp. 1–21.

[22] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, New York, 1985.

[23] P. Indyk, *Stable distributions, pseudorandom generators, embeddings and data stream computation*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, 2000, pp. 189–197.

[24] J. Kleinberg, *Two algorithms for nearest-neighbor search in high dimensions*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, 1997, pp. 599–608.

[25] K. V. Mardia, J. T. Kent, and J. M. Bibby, *Multivariate Analysis*, Academic Press, London, 1979.

[26] G. W. Stewart, *Four algorithms for the efficient computation of truncated QR approximations to a sparse matrix*, Numer. Math., 83 (1999), pp. 313–323.

[27] G. W. Stewart, *Error Analysis of the Quasi-Gram-Schmidt Algorithm*, Tech. Report UMIACS TR-2004-17 CMSC TR-4572, University of Maryland, College Park, MD, 2004.

[28] G. W. Stewart and J. G. Sun, *Matrix Perturbation Theory*, Academic Press, New York, 1990.

[29] S. Vempala, *Random projection: A new approach to VLSI layout*, in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, 1998, pp. 389–395.

SIAM J. COMPUT.
Vol. 36, No. 1, pp. 207–229

A COMPLETE CLASSIFICATION OF THE COMPLEXITY OF PROPOSITIONAL ABDUCTION*

NADIA CREIGNOU† AND BRUNO ZANUTTINI‡

Abstract. Abduction is the process of explaining a given query with respect to some background knowledge. For instance, p is an explanation for the query q given the knowledge $p \to q$. This problem is well known to have many applications, particularly in artificial intelligence (AI), and has been widely studied from both an AI and a complexity-theoretic point of view. In this paper we completely classify the complexity of propositional abduction in Schaefer's famous framework. We consider the case where knowledge bases are taken from a class of formulas in generalized conjunctive normal form. This means that the propositional formulas considered are conjunctions of constraints taken from a fixed finite language. We show that according to the properties of this language, deciding whether at least one explanation exists is either polynomial, NP-complete, or Σ_2P-complete. Our results are stated for a query consisting of a single, positive literal and for assumption-based solutions, i.e., the solutions must be formed upon a distinguished subset of the variables that is part of the input. We show, however, that our results can be interpreted "dually" for negative queries, and thus also for unrestricted (positive or negative) queries.

Key words. abduction, propositional logic, complexity, Boolean constraints

AMS subject classifications. 68Q17, 68Q25, 68T27

DOI. 10.1137/S0097539704446311

1. Introduction.

Abduction is a fundamental mode of reasoning which consists in searching for an *explanation*, or *plausible cause*, of a given *observed manifestation* and given some *background knowledge base*. For instance, given the knowledge base $p \to q$, the fact p is an explanation for the observation q, which means that p may have caused q. Viewing abduction as an inference problem, the conclusion p can be inferred from $p \to q$ and q; q is usually called the *query*, since it is the varying part of the input of the problem, while the background knowledge base is intended to be more persistent, as a description of how the universe behaves. Finally, an abduction problem is usually constrained by a set of *hypotheses* from which the explanations must be taken; in our framework as in many others, the set of hypotheses will be given *in intension* by a set of *abducible variables* upon which the explanations will have to be formed.

Note that the abduction process is nonmonotonic, i.e., logically strengthening (or weakening) the knowledge base may rule out or add some solutions of the problem. It is also not truth-preserving, in the sense that observing some query and inferring some explanation for it does not guarantee that the explanation did indeed occur.

From an artificial intelligence (AI) point of view, abduction has many applications. The most natural one is *medical diagnosis* (see, e.g., [5, section 6]), where a physician attempts to identify a patient's disease from some observed symptoms: here the query models the symptoms, the knowledge base describes the relations between diseases and symptoms, and finally the abducible variables model the diseases (since we do

*Received by the editors October 18, 2004; accepted for publication (in revised form) December 20, 2005; published electronically May 26, 2006.
http://www.siam.org/journals/sicomp/36-1/44631.html
†LIF, UMR CNRS 6166, Université de la Méditerranée, 163, avenue de Luminy, 13288 Marseille Cedex 9, France (creignou@lif.univ-mrs.fr).
‡GREYC, UMR CNRS 6072, Université de Caen, Boulevard du Maréchal Juin, 14032 Caen Cedex, France (zanutti@info.unicaen.fr).

not want to explain a symptom by another symptom). In the same vein, *system diagnosis* [6, 27], where one wants to discover the faulty components of a system that does not behave as desired (e.g., an electronic device), can be modeled as an abduction problem. Another application is *configuration* [1], and still another one is text interpretation [16]. Abduction has also been studied with temporal knowledge bases [4]. Closer to the reasoning processes themselves, abduction is a fundamental part of the CMS/ATMS [24], and it has many relations with default reasoning [26].

From a complexity-theoretic point of view, abduction is in general very hard, since in its full generality propositional abduction as a decision problem is Σ_2P-complete; that is to say, it is at the second level of the polynomial hierarchy. However, many restrictions on the syntactic form of the knowledge base have been identified that make the problem easier, even polynomial for some of them: 2-CNF and monotone knowledge bases (see, e.g., [20]), definite Horn CNF [26, 11] and affine [28] bases, CNF bases with unit-refutable pseudocompletion [14] or with bounded kernel width [10], Horn bases given by their characteristic models [19], and some classes of DNF bases [28]. Among those restrictions some impose *local* properties on the knowledge bases, such as being in definite Horn CNF, and some impose *global* properties, such as being in Horn-renamable DNF or having a unit-refutable pseudocompletion.

Our work takes place in the framework of propositional classes of formulas defined by *local* properties. More precisely, we investigate the computational complexity of propositional abduction in Schaefer's framework. This now famous framework consists in considering propositional formulas in generalized conjunctive form. These formulas are conjunctions of constraints built over a fixed finite set L of Boolean relations. In his seminal paper [25], Schaefer studied the computational complexity of the satisfiability problem in this setting, SAT(L), and established a dichotomy theorem which asserts that for each finite set of relations L, SAT(L) is either NP-complete or in P. In a nutshell, Schaefer's dichotomy theorem asserts that there are exactly four (nontrivial) polynomial-time cases of SAT(L): every relation in L is bijunctive; every relation in L is Horn; every relation in L is dual Horn; every relation in L is affine. Moreover, this dichotomy is polynomial-time decidable, that is, there is a polynomial-time algorithm that, given a finite set of Boolean relations L, decides whether SAT(L) is NP-complete or in P.

Our main contribution is a trichotomy theorem for the computational complexity of propositional abduction (seen as a decision problem) in Schaefer's framework. We consider that hypotheses are the terms formed upon a given set of variables, and that queries are given by a single positive literal. In this framework, we prove that depending on the set of relations L the abduction problem is either polynomial or NP-complete or Σ_2P-complete. More precisely, we show that the abduction problem with positive queries is polynomial-time solvable if every relation in the language is bijunctive, affine, definite Horn, IHS-$B-$, or IHS-$B+$;[1] otherwise, NP-complete if the language is Horn or dual Horn; and, finally, Σ_2P-complete in every other case. The first essential difference from Schaefer's dichotomy theorem is that, unlike the satisfiability problem, the abduction problem for Horn relations and for dual Horn relations need not to be in P. Instead, the abduction problem is in P for certain classes of Horn relations (namely, definite Horn, IHS-$B-$) and for certain classes of dual Horn relations (namely, IHS-$B+$), while it is NP-complete for all other classes of Horn relations and all other classes of dual Horn relations. Furthermore, the abduction problem is Σ_2P-complete for every finite set of relations for which the satisfiability is

[1] See [18] for the definition of IHS-$B-$ and IHS-$B+$ relations.

NP-complete.

From this result we derive two additional trichotomy results by considering the variants of abduction in which the query to be explained is a negative literal, or an unrestricted (positive or negative) literal.

In order to obtain these results, on one hand we exhibit new tractable languages for abduction (namely, IHS-$B-$ and IHS-$B+$ languages), and on the other hand we identify new minimal sets of relations that make abduction within a given language NP- or Σ_2P-hard, in the sense that any language able to express these relations defines a hard problem.

In addition to these new results, we revisit known ones and unify all of them into the same framework, while they were originally stated within (slightly) different formalizations of the abduction process, thus making our classification self-contained. We also wish to emphasize that our results are *constructive*, in the sense that we give efficient algorithms for the polynomial cases and exhibit effective reductions for the hard ones.

For use in AI applications, our results allow the designers of knowledge-based systems to choose among different languages for their knowledge bases; this choice can be made according to the importance of the abduction process for the precise application and maybe to other constraints.

The paper is organized as follows. Sections 2 and 3 present the basic definitions and the abduction problem. Sections 4 to 7 are devoted to the proof of our main result (Theorem 31), which completely classifies the computational complexity of the abduction problem with a positive query in Schaefer's framework. In section 8 we extend these results to negative and unrestricted queries. Finally, in section 9 we conclude and identify interesting future work.

2. Preliminaries. We introduce in this section the basic definitions and the main tools that we will use throughout the paper.

2.1. Propositional formulas. Let V be a set of variables. A *propositional formula over* V is any well-formed formula built upon the variables in V and the connectives $\neg$ (negation), $\vee$ (disjunction), and $\wedge$ (conjunction); given a propositional formula φ we denote by $\mathit{Vars}(\varphi)$ the set of all the variables that occur in φ.

A *literal* is either a variable x (*positive* literal) or the negation $\neg x$ of one (*negative* literal); we write $\mathit{Lits}(V)$ for the set of all literals over V, and we write $\overline{\ell}$ for the opposite of literal ℓ. A *clause* is a finite disjunction of literals of the form $C = (\ell_1 \vee \cdots \vee \ell_k)$ $(k \geq 0)$; slightly abusing notation we write, e.g., $\ell_1 \in C$. A formula is said to be in *conjunctive normal form* (CNF) if it is written as a conjunction of clauses, and, dually, in *disjunctive normal form* (DNF) if it is written as a disjunction of terms (conjunctions of literals).

An *assignment* to a set of variables V is a mapping from V to $\{0,1\}$; we also see a mapping from a superset of V to $\{0,1\}$ as an assignment to V by considering its restriction to V. When the variables under consideration and their order are clear from the context, we will sometimes denote an assignment m to, say, $\{x_1,\ldots,x_n\}$ by the word $m(x_1)\ldots m(x_n)$. A *model* m of a propositional formula φ is an assignment to $\mathit{Vars}(\varphi)$ that satisfies φ with the usual rules for the connectives; we also write $m \models \varphi$. Finally, a formula is said to be *satisfiable* if it has at least one model, and two formulas φ, φ' are said to be *logically equivalent*, written $\varphi \equiv \varphi'$, if their sets of models are equal.

If V is some set of variables, φ is a propositional formula over V, and V' is a subset of V, we will sometimes use the notation $\exists V'\varphi$. This defines a propositional

formula over $V\backslash V'$; its models are these assignments m to $V\backslash V'$ such that there is an assignment m' to V' with $mm' \models \varphi$; note that this corresponds to projecting the set of models of φ over $V\backslash V'$, in the sense of relational algebra. The other notions are extended in a straightforward manner.

2.2. Relations and *L*-formulas. An *n-ary (logical) relation* R is a Boolean relation of arity n, i.e., a subset of $\{0,1\}^n$. Thus the elements of an n-ary relation R are n-ary Boolean vectors of the form $m = (m_1, \ldots, m_n) \in \{0,1\}^n$; we also write vectors as words, e.g., 011 for the ternary vector $(0,1,1)$.

Let V be a set of variables. A *constraint over* V is an application of an n-ary relation R to an n-tuple $(x_1, \ldots, x_n)$ of variables from V, which we write $R(x_1, \ldots, x_n)$; note in particular that repetition of variables is allowed. An assignment m to V is said to *satisfy* a constraint $C = R(x_1, \ldots, x_n)$ if $(m(x_1), \ldots, m(x_n)) \in R$ holds; we also write $m \models C$, and m is said to be a *model* of C.

Example 1. We will often use some standard relations. First of all, the unary relations $\mathsf{F} = \{(0)\}$ and $\mathsf{T} = \{(1)\}$, which force the value of a variable to 0 and 1, respectively. We will also use the k-ary relation $\mathsf{OR}_{k,j}$, which is defined $\forall k \geq 2, j \leq k$ by

$$\mathsf{OR}_{k,j} = \{0,1\}^k \backslash \{1 \ldots 1 0 \ldots 0\} \ (j \text{ times } 1).$$

Note that the constraint $\mathsf{OR}_{k,j}(x_1, \ldots, x_k)$ has the same models as the formula $(\neg x_1 \vee \cdots \vee \neg x_j \vee x_{j+1} \vee \cdots \vee x_k)$. In particular, we will use the binary relations $\mathsf{OR}_{2,0} = \{0,1\}^2\backslash\{00\}$, $\mathsf{OR}_{2,1} = \{0,1\}^2\backslash\{10\}$, and $\mathsf{OR}_{2,2} = \{0,1\}^2\backslash\{11\}$. Finally, we will use the ternary relation $\mathsf{SymOR}_{2,1}$, which is defined by

$$\mathsf{SymOR}_{2,1}(x,y,z) \equiv (\mathsf{OR}_{2,1}(x,y) \wedge \mathsf{T}(z)) \vee (\mathsf{OR}_{2,1}(y,x) \wedge \mathsf{F}(z)).$$

Example 2. Given the ternary relations $R_{\text{nae}} = \{0,1\}^3\backslash\{000, 111\}$ and $R_{1/3} = \{001, 010, 100\}$, an assignment m to $\{x,y,z\}$ satisfies the constraint $R_{\text{nae}}(x,y,z)$ if m does not map all three variables to the same value, and it satisfies the constraint $R_{1/3}(x,y,z)$ if it maps exactly one of them to 1.

Now let L be a finite set of Boolean relations, not necessarily of the same arities; we also call L a *(finite) language*. An *L-formula (over V)* is a finite conjunction of constraints over V with all relations taken from L. An assignment m to V is said to *satisfy* an L-formula φ if it satisfies every constraint in φ; m is then said to be a *model* of φ, written $m \models \varphi$. The notions of satisfiability and logical equivalence are defined as for propositional formulas, and we also define logical equivalence between a propositional and an L-formula, with obvious meaning.

2.3. Properties of relations. Throughout this paper we study different types of Boolean relations and languages following Schaefer's terminology [25]. Let R be an n-ary relation and $(x_1, \ldots, x_n)$ a tuple of pairwise distinct variables; we say that a propositional formula φ over $\{x_1, \ldots, x_n\}$ *describes* R if φ is logically equivalent to the $\{R\}$-formula $R(x_1, \ldots, x_n)$. Then R is said to be

- *bijunctive* if it can be described by a *bijunctive* formula, i.e., a formula in CNF having at most two literals in each clause;
- *affine* if it can be described by an *affine* formula, i.e., a system of linear equations over the two-element field $\{0,1\}$;
- *Horn* (resp., *dual Horn*) if it can be described by a *Horn* (resp., *dual Horn*) formula, i.e., a formula in CNF having at most one positive (resp., negative) literal in each clause;

- *definite Horn* (resp., *definite dual Horn*) if it can be described by a *definite Horn* (resp., *definite dual Horn*) formula, i.e., a formula in CNF having exactly one positive (resp., negative) literal in each clause; note that any definite (dual) Horn relation (resp., formula) is (dual) Horn;
- *IHS-B−* (for *implicative hitting set-bounded−* [18]) if it can be described by an *IHS-B−* formula, i.e., a formula in CNF whose clauses are all of one of the following types: (x_i), or $(\neg x_{i_1} \vee x_{i_2})$, or $(\neg x_{i_1} \vee \cdots \vee \neg x_{i_k})$ for some $k \geq 0$ (which is not necessarily the same for every clause); note that any IHS-$B-$ relation (resp., formula) is Horn;
- *IHS-B+* if it can be described by an *IHS-B+* formula, i.e., a formula in CNF whose clauses are all of one of the following types: $(\neg x_i)$, or $(\neg x_{i_1} \vee x_{i_2})$, or $(x_{i_1} \vee \cdots \vee x_{i_k})$ for some $k \geq 0$; note that any IHS-$B+$ relation (resp., formula) is dual Horn;
- *complementive* if for all $(m_1, \ldots, m_n) \in R$ the vector $(m_1 \oplus 1, \ldots, m_n \oplus 1)$ is also in R;
- 1-*valid* (resp., 0-valid) if the vector $(1, \ldots, 1)$ (resp., $(0, \ldots, 0)$) is in R.

Some of these properties have equivalent definitions in terms of *closure properties*. Extend any operation *op* on $\{0,1\}$ to the vectors in $\{0,1\}^n$ by

$$(m_1, \ldots, m_n)\ op\ (m'_1, \ldots, m'_n) = (m_1\ op\ m'_1, \ldots, m_n\ op\ m'_n).$$

Then a relation R is

- *bijunctive* if and only if $\forall m, m', m'' \in R, (m \wedge m') \vee (m \wedge m'') \vee (m' \wedge m'') \in R$;
- *affine* if and only if $\forall m, m', m'' \in R, m \oplus m' \oplus m'' \in R$;
- *Horn* if and only if $\forall m, m' \in R, m \wedge m' \in R$;
- *dual Horn* if and only if $\forall m, m' \in R, m \vee m' \in R$.

The first equivalence is proved in, e.g., [25], the second one is a classical result from linear algebra, the third one is proved in, e.g., [9], and the fourth one is dual.

We will use them several times in the paper and wish to emphasize that they are the basis of many results that we will use from, in particular, [7].

We also generalize the above definitions to languages. For instance, a language L is said to be *Horn* if every relation in L is Horn. Additionally, we refer to a language L as a *Schaefer language* if L is bijunctive, affine, Horn, or dual Horn.

The properties listed above play a central role in determining whether a given computational task is easy, versus when it is hard (see [7] for a survey). For instance, Schaefer's theorem [25] states that the satisfiability problem for the class of all L-formulas is in P if L is Schaefer or if it is 0- or 1-valid, and that it is NP-complete otherwise.

A very important problem is the one of deciding whether a given language has a given property, in particular for identifying the tractable cases of PQ-ABDUCTION(L). This problem is called *structure identification* in [9]. It turns out that all the properties listed above can be recognized in polynomial time for a given set of relations written as sets of tuples (in extension).

PROPOSITION 3 (recognizing properties of relations). *One can decide in polynomial time whether a relation given in extension is bijunctive, affine, (dual) Horn, definite (dual) Horn, IHS-B−, IHS-B+, complementive,* 1*-valid, or* 0*-valid.*

Proof. This is obvious for bijunctive, affine, and (dual) Horn languages given the closure properties above; more efficient algorithms are also given in [9, 29, 15]. As for definite Horn languages, simply observe that a relation is definite Horn if and only if it is Horn and contains the vector $(1, \ldots, 1)$, which can again be tested efficiently; the

test is dual for definite dual Horn languages. Complementive, 1- and 0-valid relations can also be recognized efficiently by applying their definition. Finally, IHS-$B-$ and IHS-$B+$ relations can be recognized efficiently by testing closure under the mappings $(m, m', m'') \mapsto m \wedge (m' \vee m'')$ and $(m, m', m'') \mapsto m \vee (m' \wedge m'')$, respectively; this is proved in [8] and can also be derived from [3].[2] □

2.4. Complexity classes. This paper concerns the *computational complexity* of the abduction problem and, more precisely, of the associated *decision* problem. We assume knowledge of the basic notions of complexity but briefly recall the definition of the classes we are interested in; for more details we refer the reader to Papadimitriou's book [22]. First of all, the class P is the class of all decision problems that are solvable in deterministic polynomial time. The class NP is that of all decision problems that are solvable in *nondeterministic* polynomial time, or, equivalently, for which every positive instance has a witness of polynomial size and that can be checked in *deterministic* polynomial time. A problem A is said to be NP-*hard* if there is a polynomial-time reduction from every problem in NP to A, and NP-*complete* if both NP-hard and in NP. Finally, the class $\Sigma_2\mathrm{P} = \mathrm{NP}^{\mathrm{NP}}$ is the class of all decision problems that can be solved in nondeterministic polynomial time with an oracle for a problem in NP, or, equivalently, such that every positive instance has a witness of polynomial size and that can be checked in *nondeterministic* polynomial time; as for NP-hardness, Σ_2P-hardness is established by polynomial-time reductions from already known Σ_2P-complete problems. Note that the prototypical Σ_2P-complete problem is the one of deciding the validity of a formula of the form $\exists V\ \forall V' \psi$, where V and V' are disjoint sets of variables and ψ is a propositional formula in DNF over $V \cup V'$.

3. The abduction problem. We formalize in this section the abduction problem that we will study, and we give some general, well-known results about its complexity, for use later in the paper.

3.1. Definition. We are interested in the *abduction problem* that is defined below. Recall from Example 1 that T is the relation $\{1\}$ and F is the relation $\{0\}$. Given a set of literals E, we write $\bigwedge E$ for the $\{\mathsf{T}, \mathsf{F}\}$-formula $\bigwedge_{x \in E} \mathsf{T}(x) \wedge \bigwedge_{\neg x \in E} \mathsf{F}(x)$.

We define and study the problem first for *positive literal queries* (whence "PQ"), i.e., the query to be explained consists of a single, positive literal. In section 8 we will briefly state the complexity of the problem for *negative* and *unrestricted* literal queries.

PROBLEM 4 (PQ-ABDUCTION(L)). *Let L be a language.* PQ-ABDUCTION(L) *is the following decision problem:*

- *Input: An L-formula φ, a set of variables $A \subseteq Vars(\varphi)$, and a variable $q \in Vars(\varphi) \backslash A$.*
- *Question: Is there a set $E \subseteq Lits(A)$ such that the $L \cup \{\mathsf{T}, \mathsf{F}\}$-formula $\varphi \wedge \bigwedge E$ is satisfiable but the $L \cup \{\mathsf{T}, \mathsf{F}\}$-formula $\varphi \wedge \bigwedge E \wedge \mathsf{F}(q)$ is not?*

If one exists, such a set E is called a solution *of the abduction problem.*

This definition corresponds to the intuition of an abduction problem, where the goal is to find an explanation for some observed phenomenon, in the following manner. The phenomenon to observe is modeled by the query of the problem and the explanation by the set E. The first condition on E ensures that the explanation

[2] We wish to point out that [3] and [8] are posterior to the submission of this article, and that the results presented here actually motivated the study in [8].

may indeed have occurred, and the second condition, which can be read "φ and $\bigwedge E$ together entail q," ensures that E may indeed have caused q.

Example 5. Consider the language $L = \{\mathsf{OR}_{3,1}, \mathsf{OR}_{2,1}\}$ and recall that the constraint $\mathsf{OR}_{3,1}(x, y, z)$ can be read $x \to y \vee z$, while $\mathsf{OR}_{2,1}(x, y)$ can be read $x \to y$. Let $V = \{x, y, z, t, q\}$ be a set of variables and let φ be the following L-formula over V:

$$\mathsf{OR}_{3,1}(x, t, q) \wedge \mathsf{OR}_{2,1}(t, y) \wedge \mathsf{OR}_{2,1}(z, q) \wedge \mathsf{OR}_{2,1}(z, x).$$

Let $A = \{x, y, z\}$; then (φ, A, q) is an instance of the problem PQ-ABDUCTION(L). It can be seen that $\{x, \neg y\}$ and $\{z\}$ are both solutions for this problem. On the other hand, $\{\neg x, \neg z\}$ is not a solution, since, e.g., the assignment 00000 to V satisfies $\varphi \wedge \mathsf{F}(x) \wedge \mathsf{F}(z) \wedge \mathsf{F}(q)$. The set $\{\neg x, z\}$ is not a solution either, since the formula $\varphi \wedge \mathsf{F}(x) \wedge \mathsf{T}(z)$ is not satisfiable ($\mathsf{F}(x) \wedge \mathsf{T}(z)$ contradicts the constraint $\mathsf{OR}_{2,1}(z, x)$), and finally $\{x, \neg t\}$ is not a solution because it is not included in $Lits(A)$.

Note that the definition implies that a solution E for an abduction problem does not contain both x and $\neg x$ for any variable x. Indeed, if this were the case, then obviously the formula $\varphi \wedge \bigwedge E$ would not be satisfiable, thus violating the first condition on solutions.

Note also that the problem as defined above is a *decision problem*, whereas abduction is seen in general as an *inference* problem, where the goal is to compute a solution or assert that there is none. It is, however, easily seen that whenever the decision problem is hard, then so is the inference one, and conversely, as the proofs in the paper will show, whenever it is easy the inference problem is easy as well. In the same vein, note that in general the goal of abduction is to compute a *preferred* solution, i.e., a solution that is best with respect to some criteria; but obviously, there is a preferred solution for an abduction problem if and only if there is at least one (unrestricted) solution, and that is why we ignore that point in our definition.

Finally, we wish to emphasize that the language L *parameterizes* the problem; it is not part of the input. Thus, since L is finite, an algorithm for the problem PQ-ABDUCTION(L) may include a preliminary step for transforming a input formula φ into any desirable form, at least by enumerating the (finite number of) translations of every relation in L into the desired form; what we will use most often is translation into CNF. In all cases it is easily seen that this preliminary step will only take linear time, since the size of every translated form will be bounded (because L is fixed and finite).

3.2. Prime implicates. An important notion that we will use many times in the paper is that of a *prime implicate*. Let φ be a propositional formula. A clause $C = \bigvee_{i \in I} \ell_i$ is said to be a *prime implicate* of φ if φ entails C but entails no proper subset of C, i.e., if the formula $\varphi \wedge \bigwedge\{\neg \ell_i \mid i \in I\}$ is unsatisfiable but $\forall i_0 \in I$ the formula $\varphi \wedge \bigwedge\{\neg \ell_i \mid i \in I, i \neq i_0\}$ is satisfiable. The notion is defined similarly for an L-formula φ (given some language L), considering the formula $\varphi \wedge \bigwedge_{x \in C} \mathsf{F}(x) \wedge \bigwedge_{(\neg x) \in C} \mathsf{T}(x)$. We will often use the fact that if $C = \bigvee_{i \in I} x_i \vee \bigvee_{i' \in I'} \neg x_{i'}$ is a prime implicate of φ, then there is no model of φ that maps every x_i ($i \in I$) to 0 and every $x_{i'}$ ($i' \in I'$) to 1, but for every $i_0 \in I$ (resp., $i'_0 \in I'$) there is one that maps every x_i to 0 ($i \in I$) except for x_{i_0} and every $x_{i'}$ ($i' \in I'$) to 1 (resp., every x_i to 0 and every $x_{i'}$ to 1 except for $x_{i'_0}$).

We will also often use the well-known fact that all the prime implicates of a given formula φ in CNF can be generated by repeatedly applying *resolution* to φ [23]. Recall that resolution is the process of adding the clause $C_1 \vee C_2$ to φ if there is a variable x such that both clauses $(x \vee C_1)$ and $(\neg x \vee C_2)$ are in φ and C_1, C_2 do not contain two opposite literals.

We will use prime implicates mainly through the following well-known result.

LEMMA 6. *An abduction problem (φ, A, q) has a solution if and only if there is a prime implicate of φ of the form $(\ell_1 \vee \cdots \vee \ell_k \vee q)$, where $\forall i$, ℓ_i is a literal built upon a variable in A.*

Proof. Let $E = \{\ell_1, \ldots, \ell_n\}$ be a solution for (φ, A, q). Then, by definition of a solution, $\varphi \wedge \bigwedge E \wedge \neg q$ is unsatisfiable but $\varphi \wedge \bigwedge E$ is satisfiable. We conclude that (without loss of generality) there is a $k \leq n$ such that $(\neg\ell_1 \vee \cdots \vee \neg\ell_k \vee q)$ is a prime implicate of φ.

Conversely, if $(\ell_1 \vee \cdots \vee \ell_k \vee q)$ is a prime implicate of φ, write $E = \{\neg\ell_i \mid i = 1, \ldots, k\}$. Then, by definition of a prime implicate, the formula $\varphi \wedge \bigwedge E \wedge \neg q$ is unsatisfiable and (in particular) the formula $\varphi \wedge \bigwedge E$ is satisfiable. Thus E is a solution for (φ, A, q). □

3.3. Preliminary results. We will make heavy use of some results about the abduction problem for propositional formulas. We define the abduction problem PQ-ABDUCTION($\mathcal{C}$) for a class $\mathcal{C}$ of propositional formulas similarly to the problem PQ-ABDUCTION(L); in this particular context $\bigwedge E$ denotes the conjunction of all the literals in E.

The following result concerns the class of all propositional CNF formulas. Note that this class captures the whole complexity of abduction since, as can be seen in the proof, the abduction problem for unrestricted propositional formulas is in Σ_2P.

PROPOSITION 7 (general problem [11]). *If $\mathcal{C}$ is the class of all propositional CNF formulas, then* PQ-ABDUCTION *($\mathcal{C}$) is Σ_2P-complete.*

Proof. (Membership.) By definition of the problem, and since the formulas $\varphi \wedge \bigwedge E$ and $\varphi \wedge \bigwedge E \wedge \neg q$ are in CNF as soon as φ is in CNF, verifying a solution E for an instance (φ, A, q) can be done in polynomial time with an oracle for the satisfiability problem of $\mathcal{C}$. Since this problem is in NP and E is obviously of size polynomial in the one of A, we have the result.

(Hardness.) Let $\Psi = \exists V\ \forall V' \psi$, where V and V' are disjoint sets of variables and ψ is a propositional formula in DNF over $V \cup V'$. Let q be a new variable (i.e., $q \notin V \cup V'$) and let φ be a formula in CNF that is logically equivalent to the formula $\neg\psi \vee q$. Since ψ is in DNF, such a φ can be computed in polynomial time from Ψ with De Morgan's laws. We show that Ψ is valid if and only if the abduction problem (φ, V, q) has a solution, which will prove the result since deciding the validity of such a Ψ is a Σ_2P-complete problem.

Assume first that Ψ is valid, and let E be an assignment to V that witnesses this fact. On one hand, it is easily seen that the formula $\neg\psi \wedge \bigwedge E$ is unsatisfiable, and thus that the formula $(\neg\psi \vee q) \wedge \bigwedge E \wedge \neg q$, which is logically equivalent to $\varphi \wedge \bigwedge E \wedge \neg q$, is unsatisfiable as well. On the other hand, since q is a new variable, any assignment to $V \cup V' \cup \{q\}$ that satisfies $\bigwedge E \wedge q$ satisfies $(\neg\psi \vee q) \wedge \bigwedge E$, and thus the formula $\varphi \wedge \bigwedge E$ is satisfiable. We conclude that E is a solution for the problem (φ, V, q).

Conversely, let E be a solution for the problem (φ, V, q). In order to obtain a contradiction suppose that the formula $\neg\psi \wedge \bigwedge E$ is satisfiable; write m for one of its models. So, it is easily seen that the assignment m' to $V \cup V' \cup \{q\}$ defined by $m(q) = 0$ and $\forall x \in V \cup V', m'(x) = m(x)$, satisfies $\neg\psi \wedge \bigwedge E$ as well, since q is a new variable, and thus m' satisfies $\varphi \wedge \bigwedge E \wedge \neg q$; this contradicts the fact that E is a solution. Hence, we deduce that the formula $\neg\psi \wedge \bigwedge E$ is unsatisfiable. Therefore, E witnesses the fact that the formula $\exists V\ \forall V' \psi = \Psi$ is valid. □

We now turn to "easier" cases. Recall that a propositional formula in CNF is said to be *Horn* (resp., *dual Horn*) if each of its clauses has at most one positive (resp., negative) literal.

PROPOSITION 8 ((dual) Horn abduction [26]). *If $\mathcal{C}$ is the class of all propositional Horn formulas, then* PQ-ABDUCTION $(\mathcal{C})$ *is* NP*-complete. The same holds if $\mathcal{C}$ is the class of all propositional dual Horn formulas.*

Proof. We first consider the Horn case. Membership in NP follows from the fact that the satisfiability of a Horn formula can be decided in linear time, and from the fact that if φ is Horn, then so are $\varphi \wedge \bigwedge E$ and $\varphi \wedge \bigwedge E \wedge \neg q$. As for hardness, we reduce the satisfiability problem for propositional CNF formulas to this one. Let $\Psi = \bigwedge_{i \in I} C_i$ $(I \neq \emptyset)$ be a CNF formula where $\forall i \in I$, C_i is a clause. For all $i \in I$, let c_i be a new variable (i.e., $c_i \notin \mathit{Vars}(\Psi)$). For every variable $x \in \mathit{Vars}(\Psi)$, let p_x, n_x, v_x be three new variables (intuitively, p stands for "positive" and n for "negative"), and finally let q be a new variable. Let φ be the propositional formula

$$\bigwedge_{x \in \mathit{Vars}(\Psi)} ((\neg p_x \vee \neg n_x) \wedge (\neg p_x \vee v_x) \wedge (\neg n_x \vee v_x))$$
$$\wedge \bigwedge_{x \in C_i} (\neg p_x \vee c_i) \wedge \bigwedge_{(\neg x) \in C_i} (\neg n_x \vee c_i) \wedge \left(\bigvee_{x \in \mathit{Vars}(\Psi)} \neg v_x \vee \bigvee_{i \in I} \neg c_i \vee q \right).$$

It is easily seen that φ is Horn and can be built in polynomial time from Ψ. Now let $A = \{p_x, n_x \mid x \in \mathit{Vars}(\Psi)\}$; we show that the abduction problem (φ, A, q) has a solution if and only if Ψ is satisfiable.

Assume first that Ψ is satisfiable, write m for one of its models and define E to be $\{p_x \mid m(x) = 1\} \cup \{n_x \mid m(x) = 0\}$. Let m' be the assignment to $\mathit{Vars}(\varphi)$ defined by

$$\begin{cases} \forall x \in \mathit{Vars}(\Psi),\ m'(p_x) = m(x), \\ \forall x \in \mathit{Vars}(\Psi),\ m'(n_x) = m(x) \oplus 1, \\ \forall x \in \mathit{Vars}(\Psi),\ m'(v_x) = 1, \\ \forall i \in I,\ m'(c_i) = 1, \\ m'(q) = 1. \end{cases}$$

It is easily seen that m' is a model of the formula $\varphi \wedge \bigwedge E$, which is therefore satisfiable. To obtain a contradiction suppose that the formula $\varphi \wedge \bigwedge E \wedge \neg q$ is satisfiable, and write m' for one of its models. Since m' satisfies $\bigwedge E$, because of the clauses $(\neg p_x \vee v_x)$ and $(\neg n_x \vee v_x)$ we get $\forall x \in \mathit{Vars}(\Psi), m'(v_x) = 1$. Now, since m satisfies Ψ it satisfies at least one literal in each clause of Ψ, and because of the clauses $(\neg p_x \vee c_i)$ and $(\neg n_x \vee c_i)$ we deduce $\forall i \in I, m'(c_i) = 1$. Since by assumption $m'(q) = 0$, we conclude $m' \not\models (\bigvee_{x \in \mathit{Vars}(\Psi)} \neg v_x \vee \bigvee_{i \in I} \neg c_i \vee q)$, which is a contradiction.

Conversely, let E be a solution for the abduction problem (φ, A, q). Then $\varphi \wedge \bigwedge E$ is satisfiable; write m for one of its models. Because of the clauses $(\neg p_x \vee \neg n_x)$ we know that $\forall x \in \mathit{Vars}(\Psi)$, either $m(p_x) = 0$ or $m(n_x) = 0$. In order to obtain a contradiction suppose that for some x_0, $m(p_{x_0}) = m(n_{x_0}) = 0$, and define the assignment m' to $\mathit{Vars}(\varphi)$ as follows:

$$\begin{cases} \forall x \in \mathit{Vars}(\Psi),\ m'(p_x) = m(p_x), \\ \forall x \in \mathit{Vars}(\Psi),\ m'(n_x) = m(n_x), \\ m'(v_{x_0}) = 0, \\ \forall x \in \mathit{Vars}(\Psi),\ x \neq x_0 \Rightarrow m'(v_x) = 1, \\ \forall i \in I,\ m(c_i) = 1, \\ m(q) = 0. \end{cases}$$

It is easily seen that m' satisfies the formula $\varphi \wedge \bigwedge E \wedge \neg q$, a contradiction. Thus $\forall x \in \mathit{Vars}(\Psi)$, $m(p_x) \neq m(n_x)$. Define the assignment m_0 to $\mathit{Vars}(\Psi)$ by $\forall x \in \mathit{Vars}(\Psi), m_0(x) = m(p_x)$ or, equivalently, $m_0(x) = m(n_x) \oplus 1$. We show that m_0 satisfies Ψ. To obtain a contradiction, suppose that $m_0 \not\models \Psi$. Then there is a clause C_{i_0} in Ψ such that $\forall x \in C_{i_0}, m_0(x) = 0$ and $\forall(\neg x) \in C_{i_0}, m_0(x) = 1$. We deduce $\forall x \in C_{i_0}, m(p_x) = 0$ and $\forall(\neg x) \in C_{i_0}, m(n_x) = 0$. Then it is easily seen that the assignment m' defined to be equal to m except on c_{i_0} ($m'(c_{i_0}) = 0$) and on q ($m'(q) = 0$) satisfies the formula $\varphi \wedge \bigwedge E \wedge \neg q$, a contradiction. Thus m_0 satisfies Ψ, which concludes the proof for the Horn case since the satisfiability problem for the class of all formulas in CNF is NP-complete.

As for the dual Horn case, the proof is similar to the one for the Horn case by replacing every variable except q with its negation, and simplifying double negations, in the formula φ built there. This yields the following formula (for sake of clarity, n_x—resp., p_x—is now better read "x is positive—resp., negative—" and c_i, "Clause C_i is not satisfied"):

$$\bigwedge_{x \in \mathit{Vars}(\Psi)} ((p_x \vee n_x) \wedge (p_x \vee \neg v_x) \wedge (n_x \vee \neg v_x)) \wedge \bigwedge_{x \in C_i} (p_x \vee \neg c_i) \wedge \bigwedge_{(\neg x) \in C_i} (n_x \vee \neg c_i) \wedge \left(\bigvee_{x \in \mathit{Vars}(\Psi)} v_x \vee \bigvee_{i \in I} c_i \vee q \right).$$

Note that this formula is IHS-$B+$. Obviously, such a renaming of variables (excluding q) does not affect the existence of a solution for a given abduction problem. □

4. Polynomial languages. In this section we identify all the properties of a language L that make the problem PQ-ABDUCTION(L) polynomial. We wish to emphasize that all the results given here are constructive, in the sense that efficient algorithms for the problem PQ-ABDUCTION(L) can be easily derived from the proofs.

We first recall the result for the affine case, which has been considered in [28]. Note that [28] considers the class of *all* affine propositional formulas instead of classes of L-formulas, but recall from section 3.1 that since L parameterizes the problem, an affine formula equivalent to a given L-formula φ can be built in linear time from φ.

PROPOSITION 9 (affine [28]). *If L is an affine language, then the problem* PQ-ABDUCTION(L) *is in* P.

Proof. Given an instance (φ, A, q), first compute an affine formula ψ equivalent to φ in time linear in the size of φ. Next, compute a projection of the (still affine) formula $\psi \wedge (q = 0)$ onto A, i.e., an affine formula ψ' over A whose models are exactly the restrictions of the models of $\psi \wedge (q = 0)$ to A. As shown in [28], this step can be performed in polynomial time by first upper-triangulating $\psi \wedge (q = 0)$ with the variables in A put rightmost, and second selecting only those resulting equations that are formed over A. Finally, decide whether the formula $\psi \wedge \neg\psi'$ has a model by deciding whether the affine formula $\psi \wedge (\bigoplus \ell = a \oplus 1)$ has a model for at least one equation $(\bigoplus \ell = a)$ of ψ'. Indeed, such a model exists if and only if the abduction problem (φ, A, q) has a solution. Since satisfiability of affine formulas can be decided in polynomial time with Gaussian elimination, the whole algorithm is polynomial. For more details we refer the reader to [28]. □

We now use the characterization of explanations through prime implicates (Lemma 6) for deriving the following four results. Note that the first two of them are well known, although hard to trace back in the literature. Marquis [20] and Eiter and Gottlob [11], however, prove, respectively, the first and second ones in a slightly different formalization.

PROPOSITION 10 (bijunctive). *If L is a bijunctive language, then the problem* PQ-ABDUCTION(L) *is in* P.

Proof. If every relation in L is bijunctive, then given an L-formula one can compute in linear time an equivalent bijunctive propositional formula. Since all the prime implicates of a given CNF formula can be generated by resolution alone, it is easily seen that the only prime implicates that a 2-CNF formula can have are bijunctive clauses. Therefore Lemma 6 implies that there is a solution for an instance of PQ-ABDUCTION(L) if and only if there is one containing zero or one literal. Hence, if there are n abducible variables, then one can simply test the $O(n)$ literals built upon them for being a solution. This test can be performed in linear time for each literal since the satisfiability problem for bijunctive CNF formulas is linear and that class is stable under conjunction of unary clauses. □

PROPOSITION 11 (definite Horn). *If L is a definite Horn language, then the problem* PQ-ABDUCTION(L) *is in* P.

Proof. Again because the prime implicates of a given formula can be generated by resolution alone, it is easily seen that every prime implicate of a definite Horn formula is a definite Horn clause. Since q is assumed to be positive, the other literals in the clause have to be negative. It follows that if L is definite Horn, any instance (φ, A, q) of PQ-ABDUCTION(L) has a solution if and only if it has a positive solution. Since the assignment E of 1 to every variable in A is consistent with a definite Horn CNF formula φ, it is enough to decide whether $\varphi \wedge \bigwedge E \wedge \neg q$ is unsatisfiable (which can be done in linear time since φ is Horn). Indeed, if $\varphi \wedge \bigwedge E \wedge \neg q$ is unsatisfiable, then E is a solution for the abduction problem, and if it is satisfiable, then $\forall E' \subseteq E$ the formula $\varphi \wedge \bigwedge E' \wedge \neg q$ is also satisfiable (with the same model), and hence (φ, A, q) has no positive solution and, by the remark above, no solution at all. □

Finally, the next two propositions examine IHS-B languages. Those languages can be seen in some sense as maximal (dual) Horn languages that do not have the full expressive power of the class of all propositional (dual) Horn formulas, which explains their special properties. As far as we know these results are new, although easy. Note, however, that the class of IHS-B+ languages includes that of *positive* languages (i.e., sets of relations that can be described by a formula in CNF with only positive literals), and that it is already known that abduction can be performed efficiently with positive formulas (this is proved in, e.g., [20]).

Note finally that the proof for the IHS-B+ case can be used for the IHS-B− case as well, but that the algorithm described for the latter case is more efficient.

PROPOSITION 12 (IHS-B−). *If L is an IHS-B− language, then the problem* PQ-ABDUCTION(L) *is in* P.

Proof. Since an instance φ of this problem is IHS-B− and all the prime implicates of a given formula can be generated by resolution, it is easily seen that the only prime implicates of φ that can contain at least one positive literal are unary or binary. Thus, as for the bijunctive case (Proposition 10), any solution for the problem contains zero or one literal, and such candidates can be tested efficiently since φ is Horn. □

PROPOSITION 13 (IHS-B+). *If L is an IHS-B+ language, then the problem* PQ-ABDUCTION(L) *is in* P.

Proof. Since an instance φ of this problem is dual Horn, one can generate all its unary and binary prime implicates in polynomial time by a generate-and-test algorithm. Now observe that once this is done, the remaining prime implicates of φ cannot be obtained by resolving two positive clauses together, so the only new prime implicates can be obtained by resolving a binary or unary clause with a positive one.

This can only yield a positive clause of size at most that of the original positive one. Hence, if there are n variables occurring in φ and the largest positive clause in the CNF form of the relations in L is of arity k, then φ has $O(n^k)$ prime implicates, which again can be generated efficiently by a generate-and-test algorithm.

Observe that the polynomial complexity obtained here strongly relies on the fact that L, and thus k, is fixed and finite. □

Remark 14. The algorithm given in the proof of Proposition 11 shows that the following problem is in P:

- *Input:* A definite Horn formula φ in CNF, a set of variables $A \subseteq Vars(\varphi)$, and a variable $q \in Vars(\varphi)\backslash A$.
- *Question:* Is there a set $E \subseteq Lits(A)$ such that the propositional formula $\varphi \wedge \bigwedge E$ is satisfiable but the formula $\varphi \wedge \bigwedge E \wedge \neg q$ is not?

In the same manner, the corresponding problem is polynomial for affine formulas (this is proved in [28]), for bijunctive formulas in CNF (Proposition 10), and for IHS-$B-$ formulas in CNF (Proposition 12). On the contrary, the corresponding problem for IHS-$B+$ formulas in CNF is NP-complete, as the proof of Proposition 8 shows. This asymmetric behavior of the abduction problem is due to the fact that q is always a positive literal; we will observe this asymmetry again for the definite dual Horn case.

5. Useful tools for proving hardness results. We present in this section the main tools for getting hardness results, which we will use in the two following sections.

5.1. Implementations and reductions. We first introduce the definition of implementation, which has often been used for exhibiting reductions between various satisfiability problems (see [7, Definition 5.1]).

DEFINITION 15 (implementation). *A language L is said to* implement *a Boolean relation R if for all sets of variables V there exist a set of variables V' disjoint from V and an L-formula φ over $V \cup V'$ such that $R(V) \equiv \exists V' \varphi$.*

Example 16. Let $L = \{\mathsf{OR}_{3,2}\}$. Then for any $k \geq 2$, L implements the relation $\mathsf{OR}_{k,k-1}$. Indeed, for $k \geq 3$ it is easily seen that for all variables $x_1, x_2, \ldots, x_k$ the constraint $\mathsf{OR}_{k,k-1}(x_1, \ldots, x_k)$ is logically equivalent to the formula

$$\exists y_1, y_2, \ldots, y_{k-3} \mathsf{OR}_{3,2}(x_1, x_2, y_1) \wedge \left(\bigwedge_{i=2}^{k-3} \mathsf{OR}_{3,2}(y_{i-1}, x_{i+1}, y_i) \right) \wedge \mathsf{OR}_{3,2}(y_{k-3}, x_{k-1}, x_k)$$

and for $k = 2$ we have $\mathsf{OR}_{2,1}(x, y) = \mathsf{OR}_{3,2}(x, x, y)$.

It can also be seen that the language $L \cup \{\mathsf{F}\}$ implements $\mathsf{OR}_{2,2}$, since for any x, y we have $\mathsf{OR}_{2,2}(x, y) \equiv \exists z \mathsf{OR}_{3,2}(x, y, z) \wedge \mathsf{F}(z)$.

Finally, it can be seen that for any $k \geq 2$, the language $\{\mathsf{OR}_{3,1}, T\}$ implements $\mathsf{OR}_{k,0}$. Indeed, for $k \geq 2$ and any variables $x_1, \ldots, x_k$ the constraint $\mathsf{OR}_{k,0}(x_1, \ldots, x_k)$ is logically equivalent to the formula

$$\exists y_0, y_1, \ldots, y_{k-2} \mathsf{T}(y_0) \wedge \left(\bigwedge_{i=0}^{k-3} \mathsf{OR}_{3,1}(y_i, x_{i+1}, y_{i+1}) \right) \wedge \mathsf{OR}_{3,1}(y_{k-2}, x_{k-1}, x_k).$$

As is easily seen, the composition of two implementations is still an implementation. The following lemma shows a connection between implementations and reductions.

LEMMA 17. *Let φ, φ', A, q be such that the projection of the models of φ and φ' onto $A \cup \{q\}$ are the same, i.e., writing $Vars(\varphi)\backslash(A \cup \{q\}) = \{x_1, \ldots, x_m\}$ and*

$Vars(\varphi')\backslash(A \cup \{q\}) = \{x'_1, \ldots, x'_n\}$, such that the formulas $\exists x_1, \ldots, x_m, \varphi$ and $\exists x'_1, \ldots, x'_n, \varphi'$ are equivalent. Then the instances (φ, A, q) and (φ', A, q) for the abduction problem have the same solutions.

Proof. By symmetry, let E be a solution for (φ, A, q). Then there is an assignment m to $Vars(\varphi)$ that satisfies E. Consider the assignment m' to $A \cup \{q\}$ defined by $\forall x \in A \cup \{q\}, m'(x) = m(x)$. By construction m' satisfies both E and the formula $\exists x_1, \ldots, x_m, \varphi$. Thus, by assumption m' satisfies both E and the formula $\exists x'_1, \ldots, x'_m \varphi'$, and hence the formula $\varphi' \wedge \bigwedge E$ is satisfiable. The proof is dual for showing that if the formula $\varphi' \wedge \bigwedge E \wedge \neg q$ is satisfiable, then so is the formula $\varphi \wedge \bigwedge E \wedge \neg q$. This shows that every solution E for (φ, A, q) is a solution for (φ', A, q) as well, and by symmetry of the roles of φ and φ' this concludes the proof. □

As for the CNF forms of relations, if every relation in L can be implemented by L', an algorithm can look up these implementations in a catalog, and given an L-formula one can build in linear time an equivalent L'-formula (modulo existentially quantified variables). This allows us to state the following corollary of Lemma 17.

COROLLARY 18. *Let L, L' be two languages. If* PQ-ABDUCTION(L) *is* NP*-hard (resp., Σ_2P-hard) and every relation of L can be implemented by L', then* PQ-ABDUCTION$(L)'$ *is also* NP*-hard (resp., Σ_2P-hard).*

Proof. Given an instance (φ, A, q) for the problem PQ-ABDUCTION(L), let V' be a set of new variables and φ' a L'-formula such that the formulas $\exists V' \varphi'$ and φ are equivalent. Lemma 17 shows that the problems (φ, A, q) and (φ', A, q) have the same solutions. Since L' implements L and L is finite and parameterizes the problem PQ-ABDUCTION(L), one can compute φ' in polynomial time given φ (at least by including the finite number of implementations as steps of an algorithm), and thus we have a polynomial-time reduction from the problem PQ-ABDUCTION(L) to the problem PQ-ABDUCTION$(L)'$, which completes the proof. □

5.2. Useful reductions. We now state three lemmas which allow us to get rid of some unary constraints that we will introduce in our reductions. Since these lemmas do not directly add to the understanding of the complexity of abduction, their rather technical proofs are postponed to the appendices.

LEMMA 19. *Let L be a language. The problem* PQ-ABDUCTION$(L \cup \{\mathsf{T}\})$ *is polynomial-time reducible to the problem* PQ-ABDUCTION$(L \cup \{\mathsf{OR}_{2,1}\})$.

LEMMA 20. *Let L be a language. The problem* PQ-ABDUCTION$(L \cup \{\mathsf{F}\})$ *is polynomial-time reducible to the problem* PQ-ABDUCTION$(L \cup \{\mathsf{OR}_{3,1}\})$.

Note that there is a lack of symmetry in the statement of these two lemmas. This is due to the lack of symmetry with respect to the query literal (q is always a positive literal).

For the next lemma, recall from section 2 that the ternary relation $\mathsf{SymOR}_{2,1}$ is defined by

$$\mathsf{SymOR}_{2,1}(x, y, z) \equiv (\mathsf{OR}_{2,1}(x, y) \wedge \mathsf{T}(z)) \vee (\mathsf{OR}_{2,1}(y, x) \wedge \mathsf{F}(z)).$$

LEMMA 21. *Let L be a language. The problem* PQ-ABDUCTION$(L \cup \{\mathsf{F}\})$ *is polynomial-time reducible to the problem* PQ-ABDUCTION$(L \cup \{\mathsf{SymOR}_{2,1}\})$.

6. NP-complete languages. The following hardness results for basic Horn and dual Horn problems will be the cornerstones of the identification of NP-hard cases.

LEMMA 22. PQ-ABDUCTION$(\{\mathsf{OR}_{3,2}, \mathsf{F}\})$ *is* NP*-complete.*

Proof. As shown in Example 16, $\{\mathsf{OR}_{3,2}, \mathsf{F}\}$ implements $\mathsf{OR}_{2,2}$ and $\mathsf{OR}_{k,k-1}$ $\forall k \geq 2$; moreover, it can be seen that the implementation of $\mathsf{OR}_{k,k-1}$ is linear in k. Thus

the problem (φ, A, q) built in the proof of Proposition 8 for the Horn case can be reduced in polynomial time to PQ-ABDUCTION($\{\mathsf{OR}_{3,2}, \mathsf{F}\}$). Since the former problem is shown to be NP-complete there, the latter one also is. □

For dual Horn relations we can get an even stronger result, which emphasizes the asymmetric behavior of Horn and dual Horn relations.

LEMMA 23. PQ-ABDUCTION($\{\mathsf{OR}_{3,1}\}$) *is* NP*-complete.*

Proof. We first show that PQ-ABDUCTION($\{\mathsf{OR}_{3,1}, \mathsf{T}\}$) is NP-complete with a proof similar to the one of Lemma 22. Indeed, as shown in Example 16, $\{\mathsf{OR}_{3,1}, \mathsf{T}\}$ implements $\mathsf{OR}_{k,0}$ $\forall k \geq 2$, and this implementation is linear in k; since it also implements $\mathsf{OR}_{2,1}$ ($\mathsf{OR}_{2,1}(x, y) = \mathsf{OR}_{3,1}(x, y, y)$), we can reduce the problem built in the proof of Proposition 8 for the dual Horn case to PQ-ABDUCTION($\{\mathsf{OR}_{3,1}, \mathsf{T}\}$). The conclusion then follows from Lemma 19. □

Based upon these lemmas, the next propositions identify the NP-complete languages. We wish to point out that a Horn (resp., dual Horn) relation that is not IHS-$B-$ (resp., IHS-$B+$) cannot be either bijunctive, affine, or dual Horn (resp., Horn). Indeed, if R is both Horn and bijunctive, then by [29, Proposition 3] every prime implicate of any formula describing R is both Horn and bijunctive, thus IHS-$B-$, and thus R itself is IHS-$B-$, contradicting the hypothesis. Now if R is Horn and affine, then by the closure properties given in the preliminaries we get $\forall m, m', m'' \in R, (m \wedge m') \oplus (m \wedge m'') \oplus (m' \wedge m'') = (m \wedge m') \vee (m \wedge m'') \vee (m' \wedge m'') \in R$, which characterizes bijunctive relations, and R is IHS-$B-$ by the reasoning above. Finally, if R is both Horn and dual Horn, then by [29, Proposition 3] and the same reasoning as above, R is once again bijunctive.

PROPOSITION 24 (dual Horn languages). *If L is a dual Horn language that is not IHS-$B+$, then the problem* PQ-ABDUCTION(L) *is* NP*-complete.*

Proof. Membership in NP follows from Proposition 8. We first observe that L is not complementive. Indeed, if L were complementive, then since it is Horn it would be dual Horn, which can be concluded easily from the closure properties given in the preliminaries and from the equality $\forall b, b' \in \{0, 1\}, b \vee b' = ((b \oplus 1) \wedge (b' \oplus 1)) \oplus 1$. Now being Horn and dual Horn L would be bijunctive, as pointed out above, and finally it would be IHS-$B+$, contradicting the hypothesis. Thus L is not complementive. We now distinguish two cases.

First, if L is not 1-valid, it follows from [7, Lemma 5.25 (2)] that L implements F. Then from [7, Lemma 5.27] it follows that L implements $\mathsf{OR}_{3,1}$ (the proof does not use the constant relation T). The conclusion follows from Corollary 18 and Lemma 23.

Now assume that L is 1-valid. We first show that $L \cup \{\mathsf{OR}_{2,1}\}$ implements $\mathsf{OR}_{3,1}$, which will allow us to conclude that PQ-ABDUCTION($L \cup \{\mathsf{OR}_{2,1}\}$) is NP-hard with Lemma 23, and we will then show that L implements $\mathsf{OR}_{2,1}$, thus concluding the proof.

($L \cup \{\mathsf{OR}_{2,1}\}$ implements $\mathsf{OR}_{3,1}$.) First observe that by assumption L contains a relation R which is dual Horn but not IHS-$B-$. Considering a description ψ of R over $\{x_1, \ldots, x_n\}$ it is then easily seen that ψ has a prime implicate C of the form (without loss of generality) $C = (\neg x_1 \vee x_2 \vee \cdots \vee x_k)$ with the x_{i_j}'s pairwise distinct and $k \geq 3$. Let us consider the formula ψ' over $\{x_1, \ldots, x_k\}$ defined by $\psi' = \exists x_{k+1}, \ldots, x_n R(x_1, \ldots, x_k)$. Finally consider the formula ψ'' over $\{x_1, x_2, x_3\}$ defined by

$$\psi'' = \exists x_4, \ldots, x_k, \psi' \wedge \bigwedge_{i=4}^{k} \mathsf{OR}_{2,1}(x_i, x_2).$$

Every model of ψ'' that maps x_2 to 0 maps x_i to 0 $\forall i \in \{4, \ldots, k\}$ because of the

clauses $\mathsf{OR}_{2,1}(x_i, x_2)$. It follows that $100 \not\models \psi''$ by definition of a prime implicate. Now by definition of a prime implicate again, $110 \models \psi''$, since the clauses $\mathsf{OR}_{2,1}(x_i, x_2)$ do not impose any constraint on the value of any x_i. Finally, still by definition of a prime implicate, the assignments $0000\ldots0$ and $1010\ldots0$ to $\{x_1, x_2, \ldots, x_k\}$ satisfy ψ'; it is easily seen that $\forall i \in \{4, \ldots, k\}$ they satisfy the clause $\mathsf{OR}_{2,1}(x_i, x_2)$ as well, and thus 000 and 101 satisfy ψ''. Now denote by R'' the ternary relation described by ψ''; R'' satisfies exactly the assumptions of [7, Lemma 5.27], and it follows that $\{R, \mathsf{OR}_{2,1}\}$, and a fortiori $L \cup \{\mathsf{OR}_{2,1}\}$ implements $\mathsf{OR}_{3,1}$.

(L implements $\mathsf{OR}_{2,1}$.) We use once again the prime implicate C exhibited above. Let us consider the assignments to $\{x_1, x_2, \ldots, x_n\}$. By definition of a prime implicate we first get $10\ldots0 \notin R$. Since C is a prime implicate we also have $00\ldots0 \in R$ and $\forall i \geq 2$, $10\ldots010\ldots0 \in R$ (where i is the index of the second variable mapped to 1). Since R is dual Horn and thus closed under bitwise-or, we also get that for every set of indices $I \subseteq \{2, \ldots, k\}, I \neq \emptyset$, the assignment m_I that maps x_1 and every x_i with $i \in I$ to 1 and every other x_i to 0 is in R. In particular, $11\ldots1 \in R$, and for every i, $11\ldots101\ldots1 \in R$ (where i is the index of the only variable mapped to 0). Now we only need to distinguish two cases. First assume $01\ldots1 \in R$; then for every two variables x, y we have $R(x, y, \ldots, y) \equiv \mathsf{OR}_{2,1}(x, y)$. Now assume $01\ldots1 \notin R$; then since R is dual Horn and thus closed under bitwise-or, there is at least one index i such that $00\ldots010\ldots0 \notin R$ (where i is the index of the only variable mapped to 1); then $R(y, y, \ldots, y, x, y, \ldots, y) \equiv \mathsf{OR}_{2,1}(x, y)$. Hence, in both cases R implements $\mathsf{OR}_{2,1}$, thus concluding the proof. □

PROPOSITION 25 (Horn languages). *If L is a Horn language that is neither definite Horn nor IHS-B−, then the problem* PQ-ABDUCTION(L) *is* NP*-complete.*

Proof. A proof symmetric to that of Proposition 24 shows that L can implement the relation $\mathsf{OR}_{3,2}$. Since L is not definite Horn, L is not 1-valid. Moreover, as observed in the previous proposition, L is not complementive. Hence, according to [7, Lemma 5.25] L can implement F. Thus L implements $\{\mathsf{OR}_{3,2}, \mathsf{F}\}$. Finally, the conclusion follows from Corollary 18 and Lemma 22. □

7. Σ_2P-complete languages. Recall that a language L is said to be *Schaefer* if it is either affine, bijunctive, Horn, or dual Horn. If L is not Schaefer, then it does not fall in any of the cases explored in the two previous sections. We will prove in this section that if L is not Schaefer, then the problem PQ-ABDUCTION(L) is Σ_2P-complete.

We first need two lemmas.

LEMMA 26. *Let S_3 be the (finite) set of all ternary relations. The problem* PQ-ABDUCTION(S_3) *is* Σ_2P*-complete.*

Proof. It is well known that given any CNF formula ψ over some set of variables V, one can build in polynomial time a formula of the form $\exists V' \varphi$ such that φ is a propositional formula in CNF over $V \cup V'$, every clause of which is of length at most 3 and such that $\psi \equiv \exists V' \varphi$. Lemma 17 then shows that this construction yields a polynomial-time reduction from the general abduction problem (where the input is any CNF formula) to the problem PQ-ABDUCTION(S_3), and we can conclude from Proposition 7. □

LEMMA 27. *Let L be a non-Schaefer language.*

1. *If L is neither* 0- *nor* 1*-valid, nor complementive, then L implements every Boolean relation.*
2. *If L is neither* 0- *nor* 1*-valid but is complementive, then L implements* $\mathsf{SymOR}_{2,1}$.
3. *If L is* 0*-valid but not* 1*-valid, then L implements* $\mathsf{OR}_{2,1}$.

4. *If L is* 1*-valid but not* 0*-valid, then L implements* $\mathsf{OR}_{3,1}$.

5. *If L is both* 0*- and* 1*-valid, then L implements either* $\mathsf{SymOR}_{2,1}$ *or* $\mathsf{OR}_{2,1}$*, depending on whether L is complementive or not.*

Proof. For the sake of completeness we demonstrate or cite the implementations in the appendices, but the five points may be obtained by techniques from universal algebra and Post's lattice [17, 2]. □

We now make a case distinction depending on whether L is 0-valid (resp., 1-valid) or not.

PROPOSITION 28 (neither 0- nor 1-valid). *If L is neither Schaefer nor* 0*-valid nor* 1*-valid, then the problem* PQ-ABDUCTION(L) *is* Σ_2P*-complete.*

Proof. According to Lemma 27, if L is not complementive, then L implements the whole class of Boolean relations. We conclude with Lemma 26 (and Corollary 18).

Now if L is complementive, still according to Lemma 27 L implements $\mathsf{SymOR}_{2,1}$, and thus PQ-ABDUCTION(L) is as hard as PQ-ABDUCTION($L \cup \{\mathsf{F}\}$) by Lemma 21. Since $L \cup \{\mathsf{F}\}$ is neither 0-valid nor 1-valid nor complementive, PQ-ABDUCTION($L \cup \{\mathsf{F}\}$) is Σ_2P-hard, which concludes the proof. □

PROPOSITION 29 (1-valid xor 0-valid). *Let L be a* 0*-valid set of Boolean relations. If L is neither Schaefer nor* 1*-valid, then the problem* PQ-ABDUCTION(L) *is* Σ_2P*-complete. The same holds if L is* 1*-valid but neither Schaefer nor* 0*-valid.*

Proof. Membership in Σ_2P is obvious. Observe that L cannot be complementive since it is either 0-valid or 1-valid but not both.

We first consider the 0-valid case. Obviously $L \cup \{\mathsf{T}\}$ is neither Schaefer nor 0-valid nor 1-valid. We can thus conclude from Proposition 28 that the problem PQ-ABDUCTION($L \cup \{\mathsf{T}\}$) is Σ_2P-complete. From Lemma 19 it follows that the problem PQ-ABDUCTION($L \cup \{\mathsf{OR}_{2,1}\}$) is Σ_2P-complete. Now, according to Lemma 27 we know that L can implement $\mathsf{OR}_{2,1}$. From Corollary 18 we finally deduce that PQ-ABDUCTION(L) is Σ_2P-complete.

The 1-valid case can be handled in a similar way by first considering the set $L \cup \{\mathsf{F}\}$ and then successively using Lemmas 20 and 27 and Corollary 18. □

PROPOSITION 30 (0-valid and 1-valid). *Let L be a* 0*-valid and* 1*-valid set of Boolean relations. If L is not Schaefer, then the problem* PQ-ABDUCTION(L) *is* Σ_2P*-complete.*

Proof. First assume L is not complementive. Then $L \cup \{\mathsf{T}\}$ is 1-valid but neither Schaefer nor 0-valid. Thus, by Proposition 29 the problem PQ-ABDUCTION($L \cup \{\mathsf{T}\}$) is Σ_2P-complete, and we conclude as in the proof of the previous proposition.

Now if L is complementive, then according to Lemma 27 L can implement $\mathsf{SymOR}_{2,1}$. We then conclude with Lemma 21 after observing that $L \cup \{\mathsf{F}\}$ is 0-valid but neither Schaefer nor 1-valid. □

We have finally completely classified the complexity of abduction for the case when the query is a single, positive literal. Moreover, given any finite set of Boolean relations L one can efficiently decide whether the abduction problem PQ-ABDUCTION(L) is either polynomial, NP-complete, or Σ_2P-complete (see Proposition 3).

THEOREM 31 (main result). *Let L be a language.*

- *If L is bijunctive, affine, definite Horn, IHS-B+, or IHS-B−, the problem* PQ-ABDUCTION(L) *is polynomial.*
- *Otherwise, if L is Horn or dual Horn, the problem* PQ-ABDUCTION(L) *is* NP*-complete.*
- *In all other cases, the problem* PQ-ABDUCTION(L) *is* Σ_2P*-complete.*

Each of these conditions can be checked in polynomial time given a language written in extension.

8. Negative and unrestricted queries. We show briefly in this section how the previous results can be carried over to the case where the query q is a negative literal instead of a positive one, and to the case where it is unrestricted. Given some language L, write NQ-ABDUCTION(L) for the problem defined as the problem PQ-ABDUCTION(L) except that the query q in the instance is a *negative literal*, and ABDUCTION(L) for that defined for any literal query, i.e., q is either a positive or a negative literal.

PROPOSITION 32 (negative queries). *Let L be a language.*

- *If L is bijunctive, affine, definite dual Horn, IHS-B+, or IHS-B−, the problem* NQ-ABDUCTION(L) *is polynomial.*
- *Otherwise, if L is Horn or dual Horn, the problem* NQ-ABDUCTION(L) *is* NP*-complete.*
- *In all other cases, the problem* NQ-ABDUCTION(L) *is* Σ_2P*-complete.*

Each of these conditions can be checked in polynomial time given a language written in extension.

Proof. Given an n-ary relation R, define the n-ary relation $Ren(R)$ to be $\{(m_1 \oplus 1, \ldots, m_n \oplus 1) \mid (m_1, \ldots, m_n) \in R\}$. Given a language L, write $Ren(L) = \{Ren(R) \mid R \in L\}$, and given an L-formula $\varphi = \bigwedge_{i \in I} R_i(x_{i,1}, \ldots, x_{i,k_i})$, define the $Ren(L)$-formula $Ren(\varphi)$ to be $\bigwedge_{i \in I} Ren(R_i)(x_{i,1}, \ldots, x_{i,k_i})$. Then all the results can be derived from those for positive queries by observing that given a language L and an L-formula φ, the problem (φ, A, q) has a solution if and only if the problem $(Ren(\varphi), A, \neg q)$ has one. Since a relation R is Horn (resp., dual Horn) if and only if the relation $Ren(R)$ is dual Horn (resp., Horn), and similarly for the definite and IHS-B restrictions, and since all the other properties of interest are invariant under the operator Ren, we get the result. □

PROPOSITION 33 (unrestricted queries). *Let L be a language.*

- *If L is bijunctive, affine, IHS-B+, or IHS-B−, the problem* ABDUCTION(L) *is polynomial.*
- *Otherwise, if L is Horn or dual Horn, the problem* ABDUCTION(L) *is* NP*-complete.*
- *In all other cases, the problem* ABDUCTION(L) *is* Σ_2P*-complete.*

Each of these conditions can be checked in polynomial time given a language written in extension.

Proof. The result easily follows from that for positive queries and that for negative ones. □

9. Conclusion. We have completely classified the computational complexity of propositional abduction in Schaefer's framework of L-formulas. The problem appears to be either polynomial, NP-complete, or Σ_2P-complete depending on the properties satisfied by the language L. We have given the results for finite languages and for positive, negative, and unrestricted queries, which gives a fine comprehension of the influence of the polarity of the query on the complexity of the problem. Our results are summarized in Table 1.

As evoked in the introduction, various restrictions have been studied on the formalization of the abduction problem, mainly depending on the hypotheses and queries allowed. Our model is certainly one for which the problem is the easiest. At another extreme, Nordh and Zanuttini have recently studied the formalization in which the query is a term (conjunction of literals) and abducible *literals* are given instead of

TABLE 1
Summary of results for the problem PQ-ABDUCTION(L). *Membership of L in one class assumes it does not belong to any class upper in the table.*

Query	Pos.	Neg.	Any
L affine	P	P	P
L bijunctive	P	P	P
L IHS-$B-$	P	P	P
L IHS-$B+$	P	P	P
L definite Horn	P	NP-c.	NP-c.
L definite dual Horn	NP-c.	P	NP-c.
L Horn	NP-c.	NP-c.	NP-c.
L dual Horn	NP-c.	NP-c.	NP-c.
None of the above	Σ_2P-c.	Σ_2P-c.	Σ_2P-c.

variables; that is, the literals occurring in an explanation are restricted, instead of only the variables upon which they are formed. It turns out that a trichotomy has also been found for this problem [21]. Current work aims at classifying every intermediate model, considering various restrictions on the queries and hypotheses, with the goal of understanding what really makes the complexity of abduction jump from one level to another in the polynomial hierarchy.

Another interesting direction for future work is the study of other problems closely related to that of deciding the existence of a solution, and an attempt to establish a trichotomy in the complexity of them as well. Among these problems are those of deciding the relevance or the necessity of an abducible variable [11]. An abducible variable is said to be relevant (resp., necessary) to a given abduction problem if it is part of at least one of its solution (resp., of all its solutions). The necessity problem is quite close to that of deciding the existence of a solution, since α is necessary to $(\varphi, A \cup \{\alpha\}, q)$ if and only if (φ, A, q) has no solution, but no such relation is known for relevance. Other interesting issues concern the complexity of counting or enumerating the solutions. For this latter point we refer the reader to Eiter and Makino's work [12, 13]. We believe that for such studies one would first need to determine the properties that are preserved by implementations (e.g., the variables involved in a solution, the number of solutions, etc.), considering both preferred and unconstrained solutions.

Appendix A. Proof of Lemma 19. Let (φ, A, q) be an instance of PQ-ABDUCTION($L \cup \{\mathsf{T}\}$), and write

$$\varphi = \varphi_0 \wedge \bigwedge_{i \in I} \mathsf{T}(x_i),$$

where φ_0 is an L-formula.

We first assume $q \notin \{x_i \mid i \in I\}$. Define the formula

$$\varphi' = \varphi_0 \wedge \bigwedge_{i \in I} \mathsf{OR}_{2,1}(q, x_i)$$

and define furthermore A' to be $A \cup \{x_i \mid i \in I\}$. We show that the problem (φ', A', q) has a solution if and only if the problem (φ, A, q) has one.

(Solution of $(\varphi, A, q) \to$ solution of (φ', A', q).) Let E be a solution of the problem (φ, A, q), and define E' to be $E \cup \{x_i \mid i \in I\}$. It is easily seen that every model of

$\varphi' \wedge \bigwedge E'$ satisfies $\varphi \wedge \bigwedge E$ as well, and thus that $\varphi' \wedge \bigwedge E' \wedge \mathsf{F}(q)$ is unsatisfiable since by assumption so is $\varphi \wedge \bigwedge E \wedge \mathsf{F}(q)$. Now let m be a model of $\varphi \wedge \bigwedge E$. Because of the constraints $\mathsf{T}(x_i)$ in φ we know that for every $i \in I, m(x_i) = 1$, and it follows that m satisfies $\varphi' \wedge \bigwedge E'$. Thus E' is a solution for the problem (φ', A', q).

(Solution of $(\varphi', A', q) \rightarrow$ solution of (φ, A, q).) Let E' be a solution of the problem (φ', A', q), and define E to be $E' \backslash \{x_i, \neg x_i \mid i \in I, x_i \notin A\}$. Since $\varphi' \wedge \bigwedge E' \wedge \mathsf{F}(q)$ is unsatisfiable, we know that every model of $\varphi' \wedge \bigwedge E'$ maps q to 1 and thus, because of the constraint $\mathsf{OR}_{2,1}(q, x_i)$ $\forall i \in I$, that it maps x_i to 1 $\forall i \in I$. Since by definition of a solution there is at least one such model, $\varphi \wedge \bigwedge E'$ is satisfiable, and a fortiori $\varphi \wedge \bigwedge E$ is. For the same reason, we know that for every $i \in I$, E' does not contain $\neg x_i$. In order to obtain a contradiction suppose that $\varphi \wedge \bigwedge E \wedge \mathsf{F}(q)$ is satisfiable. It follows from the previous remark that $\varphi \wedge \bigwedge E' \wedge \mathsf{F}(q)$ is satisfiable and, since every model of φ satisfies φ', that $\varphi' \wedge \bigwedge E' \wedge \mathsf{F}(q)$ is satisfiable as well, which contradicts the fact that E' is a solution. Thus E is a solution for the problem (φ, A, q), which concludes the proof for the case $q \notin \{x_i \mid i \in I\}$.

We now consider the case $q \in \{x_i \mid i \in I\}$. The reduction above does not work anymore since the query is not allowed to be abducible in the definition of the problem. However, we observe that in this case the problem (φ, A, q) has a solution if and only if φ is satisfiable. Indeed, if this is the case, then $E = \emptyset$ is a solution, (because the constraint $\mathsf{T}(q)$ in φ forces the value of q to 1 in every model of φ), and if φ is unsatisfiable, then no E can be found such that $\varphi \wedge \bigwedge E$ is satisfiable, as required for a solution. Thus in this case, we introduce a new variable q' ($q' \notin \mathit{Vars}(\varphi)$) and reduce the problem (φ, A, q) to the problem $(\varphi', A' = \{q\}, q')$ with

$$\varphi' = \varphi_0 \wedge \bigwedge_{i \in I} \mathsf{OR}_{2,1}(q, x_i) \wedge \mathsf{OR}_{2,1}(q, q') \wedge \mathsf{OR}_{2,1}(q', q).$$

Note first that φ' is an $(L \cup \{\mathsf{OR}_{2,1}\})$-formula. Note also that $\mathsf{OR}_{2,1}(q, q') \wedge \mathsf{OR}_{2,1}(q', q)$ forces the values of q and q' to be the same in every model of φ'. We show that (φ', A', q') has a solution if and only if φ is satisfiable, which will conclude the proof by the remark above.

If φ is satisfiable, write m for one of its models. On one hand, by definition m satisfies φ_0 and $\mathsf{T}(x_i)$ $\forall i \in I$ (in particular, $\mathsf{T}(q)$). On the other hand, since q and q' are forced to the same value by φ', the formula $\varphi' \wedge \mathsf{T}(q) \wedge \mathsf{F}(q')$ is unsatisfiable. It follows that $E = \{q\}$ is a solution of (φ', A', q'). Conversely, if E is a solution of the problem (φ', A', q'), let m be a model of $\varphi' \wedge \bigwedge E \wedge \mathsf{T}(q')$. Because of the constraints $\mathsf{OR}_{2,1}(q, q')$ and $\mathsf{OR}_{2,1}(q', q)$ in φ', m maps q to 1. Thus every x_i ($i \in I$) is mapped to 1 as well because of the constraints $\mathsf{OR}_{2,1}(q, x_i)$. Therefore, m satisfies φ, which is thus satisfiable. □

Appendix B. Proof of Lemma 20. Let (φ, A, q) be an instance of PQ-ABDUCTION$(L \cup \{\mathsf{F}\})$, and write

$$\varphi = \varphi_0 \wedge \bigwedge_{i \in I} \mathsf{F}(x_i),$$

where φ_0 is an L-formula.

First of all, if $q \in \{x_i \mid i \in I\}$, then the problem cannot have a solution; hence it can be reduced soundly and in constant time to the constant problem $(\varphi' = \mathsf{OR}_{3,1}(q, q, q), A' = \emptyset, q)$.

Thus we assume without loss of generality that $q \notin \{x_i \mid i \in I\}$. Let q' be a new variable (i.e., $q' \notin \mathit{Vars}(\varphi)$), write $I = \{i_1, \ldots, i_k\}$, and define the formula

$$\varphi' = \varphi_0 \wedge \mathsf{OR}_{k+2,1}(q, x_{i_1}, \ldots, x_{i_k}, q').$$

It is easily seen that φ' can be built in polynomial time from φ and that the relation $\mathsf{OR}_{k+2,1}$ is implemented by $\{\mathsf{OR}_{3,1}\}$ in the usual way. Now define A' to be $A \cup \{x_i \mid i \in I\}$. We show that the problem (φ', A', q') has a solution if and only if the problem (φ, A, q) has one.

(Solution of $(\varphi, A, q) \rightarrow$ solution of (φ', A', q').) Assume that E is a solution for the problem (φ, A, q), and define E' to be $E \cup \{\neg x_i \mid i \in I\}$. It is easily seen that every model of the formula $\varphi' \wedge \bigwedge E'$ satisfies $\varphi \wedge \bigwedge E$ as well, and thus that $\varphi' \wedge \bigwedge E' \wedge \mathsf{F}(q)$ is unsatisfiable since $\varphi \wedge \bigwedge E \wedge \mathsf{F}(q)$ is also. It follows that $\varphi' \wedge \bigwedge E' \wedge \mathsf{OR}_{k+1,1}(q, x_{i_1}, \ldots, x_{i_k})$ is unsatisfiable, and thus that $\varphi' \wedge \bigwedge E' \wedge \mathsf{F}(q')$ is unsatisfiable as well. Now let m be a model of $\varphi \wedge \bigwedge E$. Because of the clauses $\mathsf{F}(x_i)$ in φ we know that $\forall i \in I, m(x_i) = 0$, and it follows that adding $m(q') = 1$ to m we get a model of $\varphi' \wedge \bigwedge E'$. Thus E' is a solution for the problem (φ', A', q').

(Solution of $(\varphi', A', q') \rightarrow$ solution of (φ, A, q).) Let E' be a solution for the abduction problem (φ', A', q'). Define E to be $E' \backslash \{x_i, \neg x_i \mid i \in I, x_i \notin A\}$. First of all, we know that $\varphi' \wedge \bigwedge E' \wedge \mathsf{F}(q')$ is unsatisfiable. It follows that $\varphi_0 \wedge \mathsf{OR}_{k+1,1}(q, x_{i_1}, \ldots, x_{i_k}) \wedge \mathsf{F}(q') \wedge \bigwedge E'$ is unsatisfiable and thus that $\varphi_0 \wedge \mathsf{OR}_{k+1,1}(q, x_{i_1}, \ldots, x_{i_k}) \wedge \bigwedge E'$ is unsatisfiable since $q' \notin \mathit{Vars}(\varphi_0)$. Since $\varphi_0 \wedge \bigwedge E'$ is satisfiable (because so is $\varphi' \wedge \bigwedge E'$), we derive that (i) for no $i \in I$, E' contains x_i, and thus that every model of $\varphi \wedge \bigwedge E$ satisfies $\varphi_0 \wedge \bigwedge E'$ as well, and (ii) $\varphi_0 \wedge \bigwedge E' \wedge \mathsf{F}(q)$ is unsatisfiable, since otherwise any of its models would satisfy $\mathsf{F}(q)$, thus $\mathsf{OR}_{k+1,1}(q, x_{i_1}, \ldots, x_{i_k})$ and finally $\varphi_0 \wedge \mathsf{OR}_{k+1,1}(q, x_{i_1}, \ldots, x_{i_k}) \wedge \bigwedge E'$. Finally, by point (i) every model of $\varphi \wedge \bigwedge E \wedge \mathsf{F}(q)$ satisfies $\varphi_0 \wedge \bigwedge E' \wedge \mathsf{F}(q)$, and thus the latter is unsatisfiable by point (ii). On the other hand, since every model of $\varphi_0 \wedge \bigwedge E' \wedge \mathsf{OR}_{k,0}(x_{i_1}, \ldots, x_{i_k})$ satisfies $\varphi_0 \wedge \bigwedge E' \wedge \mathsf{OR}_{k+1,1}(q, x_{i_1}, \ldots, x_{i_k})$, which is shown to be unsatisfiable above, it is unsatisfiable too, and thus we know that any model of $\varphi_0 \wedge \bigwedge E'$ satisfies $\varphi_0 \wedge \bigwedge E' \wedge \bigwedge_{i \in I} \mathsf{F}(x_i)$ and, since there is at least one such model, that $\varphi \wedge \bigwedge E$ is satisfiable, which concludes the proof. □

Appendix C. Proof of Lemma 21. Let (φ, A, q) be an instance of PQ-ABDUCTION$(L \cup \{\mathsf{F}\})$, and write

$$\varphi = \varphi_0 \wedge \bigwedge_{i \in I} \mathsf{F}(x_i),$$

where φ_0 is an L-formula.

First of all, if $q \in \{x_i \mid i \in I\}$, then the problem cannot have a solution; hence it can be reduced soundly and in constant time to the constant problem $(\varphi' = \mathsf{SymOR}_{2,1}(q, q, q), A' = \emptyset, q)$.

Thus we assume without loss of generality that $q \notin \{x_i \mid i \in I\}$. Let q', β be two new variables (i.e., $q', \beta \notin \mathit{Vars}(\varphi)$), and define the formula

$$\varphi' = \varphi_0 \wedge \mathsf{SymOR}_{2,1}(\beta, q, q') \wedge \bigwedge_{i \in I} (x_i = \beta).$$

Moreover, let $A' = A \cup \{\beta\} \cup \bigcup \{x_i \mid i \in I\}$. We show that the abduction problem (φ, A, q) has a solution if and only if the problem (φ', A', q') has one. This will conclude the proof since replacing every x_i $(i \in I)$ with β and removing all the equalities in

φ', one gets an $L \cup \{\mathsf{SymOR}_{2,1}\}$-formula φ'', and it is easily seen that the abduction problems (φ', A, q') and (φ'', A, q') have the same solution (β and all the x_i's play the same role with respect to A and q').

(Solution of $(\varphi, A, q) \rightarrow$ solution of (φ', A', q').) Let E be a solution for (φ, A, q), and define E' to be $E \cup \{\neg\beta\} \cup \{\neg x_i \mid i \in I\}$. Let m be a model of $\varphi \wedge \bigwedge E$. Then m satisfies φ_0 and q, and it is easily seen that, adding $m(\beta) = 0$ and $m(q') = 1$ to m, one gets a model of $\varphi' \wedge \bigwedge E'$. Now $\varphi' \wedge \bigwedge E' \wedge \mathsf{F}(q)$ is unsatisfiable because every model of $\varphi' \wedge \bigwedge E'$ satisfies $\varphi \wedge \bigwedge E$, and $\varphi \wedge \bigwedge E \wedge \mathsf{F}(q)$ is unsatisfiable by assumption. We conclude that every model of $\varphi' \wedge \bigwedge E'$ maps q to 1, and since $(\neg\beta) \in E'$ we know that no model of $\varphi' \wedge \bigwedge E'$ satisfies $\mathsf{OR}_{2,1}(q, \beta)$ and thus, because of the relation $\mathsf{SymOR}_{2,1}(\beta, q, q')$, that every model m of $\varphi' \wedge \bigwedge E'$ satisfies $m(q') = 1$, which finishes to show that E' is a solution for (φ', A', q').

(Solution of $(\varphi', A', q') \rightarrow$ solution of (φ, A, q).) Let E' be a solution for (φ', A', q'), and let $E = E' \backslash (\{x_i, \neg x_i \mid i \in I, x_i \notin A\} \cup \{\beta, \neg\beta\})$. Since $\varphi' \wedge \bigwedge E' \wedge \mathsf{F}(q')$ is unsatisfiable and q' occurs only in the constraint $\mathsf{SymOR}_{2,1}(\beta, q, q')$, we know that $\varphi' \wedge \bigwedge E' \wedge \mathsf{OR}_{2,1}(q, \beta)$ is unsatisfiable. Indeed, otherwise there would be a model of $\varphi' \wedge \bigwedge E'$ satisfying $\mathsf{OR}_{2,1}(q, \beta) \wedge \mathsf{F}(q')$ (precisely, a model m of $\varphi' \wedge \mathsf{OR}_{2,1}(q, \beta)$ with $m(q')$ changed to 0). We conclude that every model of $\varphi' \wedge \bigwedge E'$ satisfies $\mathsf{T}(q) \wedge \mathsf{F}(\beta)$, and thus $\mathsf{F}(x_i)$ for every $i \in I$. Hence on one hand, every model of $\varphi' \wedge \bigwedge E'$ is a model of $\varphi \wedge \bigwedge E$, which is thus satisfiable. On the other hand, in order to obtain a contradiction, suppose that $\varphi \wedge \bigwedge E \wedge \mathsf{F}(q)$ is satisfiable and let m be a model. Then m satisfies $\mathsf{F}(x_i)$ for every $i \in I$, and thus satisfies $\varphi_0 \wedge \mathsf{F}(q) \wedge \bigwedge E'$. It is easily seen that, adding $m(\beta) = m(q') = 0$ to m, one gets a model of $\varphi' \wedge \bigwedge E' \wedge \mathsf{F}(q')$, which contradicts the fact that E' is a solution. □

Appendix D. Proof of Lemma 27. The first point is Schaefer's result [25]. The third and fifth ones are given in [7, Lemma 5.41].

(Second claim.) First remark that $L \cup \{\mathsf{T}\}$ is neither 0-valid nor 1-valid nor complementive. Thus by the first claim $L \cup \{\mathsf{T}\}$ can implement every Boolean relation. Consider the ternary relation $R = \{000, 100, 110\}$; since $L \cup \{\mathsf{T}\}$ can implement every Boolean relation there is an L-formula φ_0 over $V \cup \{x, y, z, t\}$ with

$$R(x, y, z) \equiv \exists V \ \exists t, \varphi_0 \wedge \mathsf{T}(t).$$

Note that we assume without loss of generality that there is only one constraint involving the relation T, since we can replace all the variables that are in the scope of such a constraint with t (noting that none involves any of x, y, z because $000 \in R$). By construction the 4-ary relation R' defined by $R'(x, y, z, t) = \exists V \varphi_0$ contains 0001, 1001, and 1101; moreover, since it is complementive (because so is L) it also contains 1110, 0110, and 0010. Now assume that there is another 4-ary vector $m = (m_x, m_y, m_z, m_t)$ in R'; then if $m_t = 1$, we should have $(m_x, m_y, m_z) \in R$, which is false by assumption. Now if $m_t = 0$, by complementivity of L we should have $(m_x \oplus 1, m_y \oplus 1, m_z \oplus 1, m_t \oplus 1) \in R'$ and thus $(m_x \oplus 1, m_y \oplus 1, m_z \oplus 1) \in R$, which is false as well. It is finally easily seen that the formula $\exists t R'(x, y, z, t)$ is logically equivalent to the formula $\mathsf{SymOR}_{2,1}(x, y, z)$, which concludes the proof.

(Fourth claim.) Since L is not Schaefer there are three relations R, R', and R'' in L that are, respectively, non–dual Horn, non-Horn, and nonaffine. Thus because of the closure properties given in the preliminaries there are two vectors $m_1 = 1^a 1^b 0^c 0^d, m_2 = 1^a 0^b 1^c 0^d \in R$ such that $m_1 \vee m_2 = 1^a 1^b 1^c 0^d \notin R$, where, e.g., 1^a indicates that the value 1 is repeated a times; by identifying the components of those vectors we can see that R implements some 4-ary relation R_0 with $1100, 1010 \in R_0$

and $1110 \notin R_0$. (Note that we can assume $a, b, c, d \neq 0$ since we can introduce auxiliary, unconstrained variables and check that the assumptions remain true.) In the same manner, the non-Horn relation R' implements some 4-ary relation R'_0 with $1100, 1010 \in R'_0$ and $1000 \notin R'_0$, and finally the 1-valid but nonaffine relation R'' satisfies $\exists m_1, m_2 \in R'', m_1 \oplus m_2 \oplus 1 \ldots 1 \notin R''$ [7, Lemma 4.10], from which we derive as before that R'' implements some 4-ary relation R''_0 with $1100, 1010 \in R''_0$ and $1001 \notin R''_0$. Now consider the ternary relation R_1 defined by

$$R_1(x, y, z) = \exists t, R_0(t, x, y, z) \wedge R'_0(t, x, y, z) \wedge R''_0(t, x, y, z) \wedge \mathsf{T}(t).$$

By construction, $111, 100, 010 \in R_1$ and $110, 000, 001 \notin R_1$. Now consider the ternary relation R''' defined by

$$R'''(x, y, z) = \exists t, u, R_1(t, x, y) \wedge R_1(t, z, u).$$

It is easily seen that $111, 110, 001, 000, 101 \in R'''$ and that $100 \notin R'''$. Observe that if $010 \in R'''$, then so does 011. Indeed, if $010 \in R'''$, then either $001 \in R_1$ or $101 \in R_1$, and since $001 \notin R_1$ we have $101 \in R_1$, and it is then easily seen that $011 \in R'''$. Therefore, there remain only two cases to discuss: either $011 \in R'''$, or both $011 \notin R'''$ and $010 \notin R'''$. In the first case we have

$$\mathsf{OR}_{3,1}(x, y, z) \exists u_1 R'''(x, u_1, u_1) \wedge R'''(u_1, y, z),$$

in the second

$$\mathsf{OR}_{3,1}(x, y, z) \exists u_1 \exists u_2 \exists u_3 \exists u_4 R'''(x, u_1, u_2) \wedge R'''(y, u_1, u_3) \wedge R'''(z, u_2, u_4).$$

We conclude that $L \cup \{\mathsf{T}\}$ implements $\mathsf{OR}_{3,1}$. Since L is 1-valid but not 0-valid, L implements T, and we finally deduce that L implements $\mathsf{OR}_{3,1}$. □

REFERENCES

[1] J. Amilhastre, H. Fargier, and P. Marquis, *Consistency restoration and explanations in dynamic CSPS—Application to configuration*, Artificial Intelligence, 135 (2002), pp. 199–234.

[2] E. Böhler, N. Creignou, S. Reith, and H. Vollmer, *Playing with Boolean Blocks, Part* II: *Post's Lattice with Applications to Complexity Theory*, in Complexity Theory Column 43, ACM-SIGACT News, Vol. 35, ACM, New York, 2004, pp. 22–35.

[3] E. Böhler, S. Reith, H. Schnoor, and H. Vollmer, *Bases for Boolean co-clones*, Inform. Process. Lett., 96 (2005), pp. 59–66.

[4] M. Bouzid and A. Ligeza, *Temporal causal abduction*, Constraints, 5 (2000), pp. 303–319.

[5] T. Bylander, D. Allemang, M. Tanner, and J. Josephson, *Some results concerning the computational complexity of abduction*, in Proceedings of the 1st Annual International Conference on Principles of Knowledge Representation and Reasoning (KR'89), Morgan Kaufmann, San Francisco, CA, 1989, pp. 44–54.

[6] S. Coste-Marquis and P. Marquis, *Characterizing consistency-based diagnoses*, in Proceedings of the 5th Annual International Symposium on Artificial Intelligence and Mathematics (AIMATH'98), 1998; available online from http://rutcor.rutgers.edu/~amai/aimath98/.

[7] N. Creignou, S. Khanna, and M. Sudan, *Complexity Classifications of Boolean Constraint Satisfaction Problems*, SIAM Monogr. Discrete Math. Appl. 7, SIAM, Philadelphia, 2001.

[8] N. Creignou, P. Kolaitis, and B. Zanuttini, *Preferred representations of Boolean relations*, Electronic Colloquium on Computational Complexity (ECCC), 2005; available online from http://eccc.hpi-web.de/eccc-reports/2005/TR05-119/index.html.

[9] R. Dechter and J. Pearl, *Structure identification in relational data*, Artificial Intelligence, 58 (1992), pp. 237–270.

[10] A. del Val, *The complexity of restricted consequence finding and abduction*, in Proceedings of the 17th National Conference on Artificial Intelligence (AAAI'00), AAAI Press, Menlo Park, CA, 2000, pp. 337–342.

[11] T. Eiter and G. Gottlob, *The complexity of logic-based abduction*, J. ACM, 42 (1995), pp. 3–42.
[12] T. Eiter and K. Makino, *On computing all abductive explanations*, in Proceedings of the 18th Annual National Conference on Artificial Intelligence (AAAI'02), AAAI Press/MIT Press, 2002, pp. 62–67.
[13] T. Eiter and K. Makino, *Generating all abductive explanations for queries on propositional Horn theories*, in Proceedings of the 12th Annual Conference of the EACSL (CSL'03), Springer Lecture Notes in Comput. Sci., Springer-Verlag, New York, 2003, pp. 197–211.
[14] K. Eshghi, *A tractable class of abduction problems*, in Proceedings of the 13th Annual International Joint Conference on Artificial Intelligence (IJCAI'93), Morgan Kaufmann, San Francisco, CA, 1993, pp. 3–8.
[15] J.-J. Hébrard and B. Zanuttini, *An efficient algorithm for Horn description*, Inform. Process. Lett., 88 (2003), pp. 177–182.
[16] J. Hobbs, M. Stickel, D. Appelt, and P. Martin, *Interpretation as abduction*, Artificial Intelligence, 63 (1993), pp. 69–142.
[17] P. Jeavons, D. Cohen, and M. Gyssens, *Closure properties of constraints*, J. ACM, 44 (1997), pp. 527–548.
[18] S. Khanna, M. Sudan, and L. Trevisan, *Constraint satisfaction: The approximability of minimization problems*, in Proceedings of the 12th Annual IEEE Conference on Computational Complexity (CCC'97), IEEE Computer Society, Los Alamitos, CA, 1997, pp. 282–296.
[19] R. Khardon and D. Roth, *Reasoning with models*, Artificial Intelligence, 87 (1996), pp. 187–213.
[20] P. Marquis, *Consequence finding algorithms*, in Handbook of Defeasible Reasoning and Uncertainty Management Systems (DRUMS), Vol. 5, Kluwer Academic, Dordrecht, The Netherlands, 2000, pp. 41–145.
[21] G. Nordh and B. Zanuttini, *Propositional abduction is almost always hard*, in Proceedings of the 19th Annual International Joint Conference on Artificial Intelligence (IJCAI'05), IJCAI, Detroit, MI, 2005, pp. 534–539.
[22] C. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, MA, 1994.
[23] W. Quine, *On cores and prime implicants of truth functions*, Amer. Math. Monthly, 66 (1959), pp. 755–760.
[24] R. Reiter and J. de Kleer, *Foundations of assumption-based truth maintenance systems: Preliminary report*, in Proceedings of the 6th Annual National Conference on Artificial Intelligence (AAAI'87), AAAI Pres, Menlo Park, CA, 1987, pp. 183–188.
[25] T. Schaefer, *The complexity of satisfiability problems*, in Proceedings of the 10th Annual Symposium on the Theory of Computing (STOC'78), ACM, New York, 1978, pp. 216–226.
[26] B. Selman and H. Levesque, *Abductive and default reasoning: A computational core*, in Proceedings of the 8th Annual National Conference on Artificial Intelligence (AAAI'90), AAAI Press, Menlo Park, CA, 1990, pp. 343–348.
[27] M. Stumptner and F. Wotawa, *Diagnosing tree-structured systems*, Artificial Intelligence, 127 (2001), pp. 1–29.
[28] B. Zanuttini, *New polynomial classes for logic-based abduction*, J. Artificial Intelligence Res., 19 (2003), pp. 1–10.
[29] B. Zanuttini and J.-J. Hébrard, *A unified framework for structure identification*, Inform. Process. Lett., 81 (2002), pp. 335–339.

SIAM J. COMPUT.
Vol. 36, No. 1, pp. 230–246

FULL CONSTRAINT SATISFACTION PROBLEMS*

TOMÁS FEDER† AND PAVOL HELL‡

Abstract. Feder and Vardi have conjectured that all constraint satisfaction problems to a fixed structure (constraint language) are polynomial or NP-complete. This so-called dichotomy conjecture remains open, although it has been proved in a number of special cases. Most recently, Bulatov has verified the conjecture for conservative structures, i.e., structures which contain all possible unary relations. We explore three different implications of Bulatov's result. First, the above dichotomy can be extended to so-called inclusive structures, corresponding to conservative constraint satisfaction problems in which each variable comes with its own domain. (This has also been independently observed by Bulatov.) We prove a more general version, extending the dichotomy to so-called three-inclusive structures, i.e., structures which contain, with any unary relation R, all unary relations R' for subsets $R' \subseteq R$ with at most three elements. For the constraint satisfaction problems in this generalization we must restrict the instances to so-called 1-full structures, in which each variable is involved in a unary constraint. This leads to our second focus, which is on restrictions to more general kinds of "full" input structures. For any set W of positive integers, we consider a restriction to W-full input structures, i.e., structures in which, for each $w \in W$, any w variables are involved in a w-ary constraint. We identify a class of structures (the so-called W-set-full structures) for which the restriction to W-full input structures does not change the complexity of the constraint satisfaction problem, and hence the family of these restricted problems also exhibits dichotomy. The general family of three-inclusive constraint satisfaction problems restricted to W-full input structures contains examples which we cannot seem to prove either polynomial or NP-complete. Nevertheless, we are able to use our result on the dichotomy for three-inclusive constraint satisfaction problems, to deduce the fact that all three-inclusive constraint satisfaction problems restricted to W-full input structures are NP-complete or "quasi-polynomial" (of order $n^{O(\log n)}$). Our third focus deals with bounding the number of occurrences of a variable, which we call the degree. We conjecture that the complexity classification of three-inclusive constraint satisfaction problems extends to the case where all degrees are bounded by three. Using previous results, we are able to verify this conjecture in a number of special cases. Conservative, inclusive, and three-inclusive constraint satisfaction problems can be viewed as problems in which each variable is restricted to a "list" of allowed values. This point of view of lists is frequently encountered in the study of graph colorings, graph homomorphisms, and graph partitions. Our results presented here, in all three areas, were strongly motivated by these results on graphs.

Key words. constraint satisfaction problems, dichotomy conjecture, conservative constraint satisfaction problems, full constraint satisfaction problems, graph homomorphisms, list homomorphisms, matrix partitions, bounded degrees, NP-complete problems, quasi-polynomial algorithms

AMS subject classifications. 05C15, 68Q15, 68Q17

DOI. 10.1137/S0097539703427197

1. Introduction.

A large class of problems in artificial intelligence and other areas of computer science can be viewed as *constraint satisfaction problems* [8, 31, 32, 33, 34, 40]. These include problems in machine vision, belief maintenance, scheduling, temporal reasoning, type reconstruction, graph theory, and satisfiability.

The standard formulation of the constraint satisfaction problem goes back to Montanari [36] in 1974. This framework has proved its value over the years by its wide-ranging applicability [8, 39]. The constraint satisfaction in its full generality is NP-complete. Thus constraint satisfaction problems have been studied under various

*Received by the editors May 5, 2003; accepted for publication (in revised form) January 22, 2006; published electronically May 26, 2006.
http://www.siam.org/journals/sicomp/36-1/42719.html
†268 Waverley St., Palo Alto, CA 94301 (tomas@theory.stanford.edu).
‡School of Computing Science, Simon Fraser University, Burnaby, BC, Canada V5A 1S6 (pavol@cs.sfu.ca).

restrictions. The main model [20] considers constraint satisfaction problems with a fixed template determining the size of the domain and the set of allowed constraint types in an instance.

Formally, a *vocabulary* V is a set of pairs (R_i, ℓ_i), where the R_i are *relation names* and the ℓ_i are *relation arities*. A *structure* S *over the vocabulary* V consists of a set (called the *domain*) D, together with a collection of relations $R_i \subseteq D^{\ell_i}$, one ℓ_i-ary relation R_i for each pair (R_i, ℓ_i) in V. We say that the structure S is an *interpretation* of the vocabulary V and emphasize this, if necessary, by writing V^S instead of S. We also write the domains and relations as D^S and R_i^S, to indicate the structure in which they are being interpreted.

The *constraint satisfaction problem* $CSP(H)$ (or $CSP(V^H)$) for a fixed structure $H = V^H$ over a vocabulary V is given as follows. An *instance* of $CSP(H)$ is a structure V^G over the same vocabulary V. The *question* asked by $CSP(H)$ is whether or not there exists a *homomorphism* f of G to H, that is, a mapping $f : D^G \to D^H$ such that if $(x_1, \ldots, x_{\ell_i}) \in R_i^G$, then $(f(x_1), \ldots, f(x_{\ell_i})) \in R_i^H$ for all choices of $x_i \in D^G$ and $(R_i, \ell_i) \in V$. We shall speak of the elements x_i of D^G as *variables*, constrained by the relations R_i^G to be assigned suitable *values* $f(x_i)$ in D^H, as allowed by the constraints R_i^H.

This model was studied in its full generality by Feder and Vardi [20], who found evidence toward the following *dichotomy conjecture.*

Conjecture 1.1. *For each fixed structure H, the problem $CSP(H)$ is NP-complete or polynomial time solvable.*

(Recall that it is known that if P $\neq$ NP, there are problems in NP that are neither NP-complete nor polynomial [30].) This conjecture was supported by earlier classifications that exhibited dichotomies. Schaefer [38] classified the complexity of Boolean constraint satisfaction problems, i.e., problems $CSP(H)$ with a two-element domain $D^H = \{0, 1\}$. In essence, the polynomial cases consist of Horn or anti-Horn clauses, two-satisfiability, and linear equations modulo two. The remaining constraint satisfaction problems $CSP(H)$ for a two-element domain D^H are all NP-complete. Bulatov classified constraint satisfaction problems on a three-element domain [3]. Hell and Nešetřil [26] classified the complexity of constraint satisfaction problems $CSP(H)$ where $H = V^H$ is a graph (with loops allowed). In other words, V consists of one pair $(R, 2)$, and the relation R^H is symmetric. They showed that this so-called *H-coloring problem* is polynomial if H is bipartite or has a loop and is NP-complete otherwise. Such a result is not known for digraphs (structures V^H in which R^H is not necessarily symmetric) [1, 20]. In fact, Feder and Vardi [20] have shown that dichotomy for digraph H-coloring problems would imply the validity of the entire dichotomy conjecture.

For this paper, the most relevant special case where the conjecture has been proved is the following result. We say that a structure H is *conservative* if it contains a unary relation R for each subset R of the domain D^H. We shall denote the set of unary relation names (names R involved in pairs $(R, 1)$) of a vocabulary V by $\mathcal{U}(V)$, and also denote the set of the corresponding relations R^H in a structure $H = V^H$ by $\mathcal{U}(V^H)$ (or just $\mathcal{U}(H)$). (Thus the vocabulary V of a conservative structure V^H with $|D^H| = n$ has $|\mathcal{U}(V)| = 2^n$, and $\mathcal{U}(V^H)$ is precisely the power set of D^H.) Bulatov [2] proved that dichotomy holds for all conservative structures, i.e., that *for every conservative structure H, the problem $CSP(H)$ is NP-complete or polynomial time solvable.* His approach is based on tools and techniques of universal algebra [2, 3, 4, 5, 28, 29]. Problems $CSP(H)$, for conservative structures H, will be called

conservative constraint satisfaction problems.

An alternative interpretation of Bulatov's result is the following. Suppose H is a conservative structure; we may restrict $CSP(H)$ to those input structures G that contain each variable $v \in D^G$ in some unary relation. Such structures G will be called 1-*full*. The restriction of $CSP(H)$ to 1-full instances G will be denoted by $CSP_1(H)$. Since H is conservative this is not a real restriction, as there exists in $\mathcal{U}(V)$ a pair $(R, 1)$ with the corresponding unary relation R^H equal to D^H. Thus the relation R^G can be imposed an any variable in D^G without changing the problem, and so $CSP(H)$ and $CSP_1(H)$ are equivalent problems. On the other hand, $CSP_1(V^H)$ is easily seen to be equivalent to the following *list constraint satisfaction problem*. The *instance* is a structure V^G in which each variable $v \in D^G$ is equipped with a list $L(v) \subseteq D^H$, and the *question* is whether or not there exists a homomorphism f of G to H such that each $f(v)$ belongs to $L(v)$. List constraint satisfaction problems have been extensively studied in the case when H is a graph. The problem is polynomial if H is a so-called bi-arc graph and is NP-complete otherwise [12, 13, 14, 15, 16]. Let C be a fixed circle with two specified points p and q. A graph H (with loops allowed) is called a *bi-arc graph* if there exist pairs of arcs (N_x, S_x), $x \in V(H)$, with each N_x being an arc on C containing p but not q, and each S_x being an arc on C containing q but not p, so that for any $x, y \in V(H)$, not necessarily distinct, the following hold:

- either N_x intersects S_y and N_y intersects S_x, or N_x does not intersect S_y and N_y does not intersect S_x;
- N_x intersects S_y (and N_y intersects S_x) if and only if x and y are not adjacent in H.

This class of graphs conveniently generalizes both the class of interval graphs, and the class of (complements of) circular arc graphs of clique covering number two [14, 22].

List constraint satisfaction problems involving multiple binary relations were studied by Feder, Madelaine, and Stewart [19].

Let us say that a structure H is *inclusive* if for any $R \in \mathcal{U}(H)$ and any $R' \subseteq R$ we also have $R' \in \mathcal{U}(H)$. Note that each conservative structure is inclusive. It was independently observed by the present authors, and by Bulatov, that the above result for conservative structures H extends to all inclusive structures in the following sense. *For every inclusive structure H, the problem $CSP_1(H)$ is NP-complete or polynomial time solvable.* Since for conservative structures H the problems $CSP_1(H)$ and $CSP(H)$ are equivalent, this extends the original result of Bulatov [2]. It is easy to interpret this result as the dichotomy of a list constraint satisfaction problem where the variables range over multiple domains. In this context, Feder has previously classified a family of list problems involving multiple Boolean domains [10].

We prove in this paper a more general version of the dichotomy for inclusive structures. We say that a structure H is *three-inclusive* if for any $R \in \mathcal{U}(H)$ and all $R' \subseteq R$ with $|R'| \leq 3$ we also have $R' \in \mathcal{U}(H)$. Thus each inclusive (and hence each conservative) structure is also three-inclusive. We shall prove that *for any three-inclusive structure H, the problem $CSP_1(H)$ is NP-complete or polynomial time solvable.* Problems $CSP_1(H)$, for three-inclusive structures H, will be called *three-inclusive constraint satisfaction problems.*

Three-inclusive structures H are the main focus of this paper. Note that a structure H without unary relations is three-inclusive (in fact inclusive) and thus proving dichotomy for $CSP(H)$ would yield dichotomy for digraph H-coloring problems and hence the entire dichotomy conjecture. Therefore we restrict the instances G as well. The restriction $CSP_1(H)$ mentioned above is one natural way—restricting to 1-full

instances G amounts to assuming the inputs are equipped with lists, as described above. Thus requiring H to be three-inclusive and G to be 1-full gives the three-inclusive constraint satisfaction problem $CSP_1(H)$ that is the connecting thread of our paper. (We then consider further restrictions on both G and H.)

This suggests looking at other versions of "fullness." Let W be a set of positive integers such that $1 \in W$, and let V be a vocabulary which contains at least one pair (R, w) for each $w \in W$. We shall say that a structure $G = V^G$ is *W-full* if for each $w \in W$, and any distinct variables $x_1, x_2, \ldots, x_w$ in D^G, there exist a permutation σ of $\{1, 2, \ldots, w\}$ and a pair (R, w) in V, such that $(x_{\sigma(1)}, \ldots, x_{\sigma(k)}) \in R^G$. Note that the notion of a 1-full structure coincides with the notion of a W-full structure with $W = \{1\}$. We shall also focus on the next natural special case when $W = \{1, 2\}$; in this case we simply say that a W-full structure is *pairwise full* (remembering that, despite the suggestive name, it is assumed to be at the same time 1-full and 2-full). The problem $CSP_W(H)$ is the restriction of $CSP(H)$ to W-full instances G.

A W-full structure $G = V^G$ is *strictly W-full* if the above permutation σ and relation R are unique for each $w \in W, w \geq 2$, and any distinct variables $x_1, x_2, \ldots, x_w$ in D^G. The restriction of $CSP(H)$ to strictly W-full instances G will be denoted by $CSP^*_W(H)$. The notion of *strictly pairwise full* structure refers to, as would be expected, a strictly W-full structure with $W = \{1, 2\}$. A graph G (with loops allowed) can be viewed as a strictly pairwise full structure with two binary relations, corresponding to the edges and the nonedges of G. (To conform to the definition as given, we choose an arbitrary orientation of G, i.e., consider each edge uv of G as an oriented arc—either $\vec{uv}$ or $\vec{vu}$—and consider each nonedge in a similar manner.) Under this interpretation, the problems $CSP^*_{\{1,2\}}(H)$ have been studied as *list matrix partition problems* [6, 18, 17, 23, 27]. (The structures H are usually taken to be conservative structures with two binary relations whose union is $(D^H)^2$.)

After the work of Jeavons [28] and Jeavons, Cohen, and Gyssens [29], it is usual, when studying the complexity of $CSP(H)$, to consider the relations "definable in H" along with the relations of H. The properties of structures considered in this paper are somewhat fragile with respect to adding definable relations. Thus to convert a $CSP(H)$ problem to a full problem $CSP_W(H)$, one needs to add to H relations of the arities in W that hold for all tuples of elements in D. To convert a full problem $CSP_W(H)$ further to a strict problem $CSP^*_W(H)$ one needs to further add to H relations of each arity in W corresponding to intersection of relations of this arity in H.

This family of (strictly) full problems is not directly under the scope of the dichotomy conjecture, and many problems in this class resist attempts at classification [6, 18, 23]. Consider, for instance, the following problem: The vocabulary V consists of eight unary relation names U_i (in other words, V contains eight pairs $(U_i, 1)$) and three binary relation names R_0, R_1, R_2 (additional pairs $(R_0, 2), (R_1, 2), (R_2, 2)$ in V). Now consider the structure V^H with domain $D = D^H = \{0, 1, 2\}$, where the interpretations of the three binary relations are $R^H_j = D^2 - (j, j)$ for $j = 0, 1, 2$, and the eight unary relations correspond to the eight subsets of D. (Assume that U_7 is the relation name for which $U^H_7 = D^H$.) The complexity of $CSP^*_{\{1,2\}}(H)$ is not known. It will follow from Theorem 3.2 that the problem can be solved in quasi-polynomial time $n^{O(\log n)}$ (where $n = |D^G|$). We first illustrate the idea of the recursive algorithm from the proof of Theorem 3.2. For simplicity, we shall assume that the instance V^G has all variables only in the unary relation U^G_7 (having the lists D^H), which does not restrict them in any way. (It is easy to argue that this version is equivalent to the original problem.) Thus we want to solve the following combinatorial problem.

Given a complete graph G with edges colored $0, 1, 2$, can the vertices also be colored by $0, 1, 2$ so that there is no monochromatic edge (an edge of color i, both of whose endpoints are also colored i)?

For each vertex x of G we choose a *majority color* $m(x)$ from $0, 1, 2$, with the property that at least a third of the edges incident to x have color $m(x)$. Suppose G has n vertices. We reduce the problem for G to $n + 1$ subproblems as follows. In the first subproblem we assume no vertex x obtains its majority color. In the remaining n subproblems we assume, for each vertex x in turn, that (at least) x obtains its majority color. The first subproblem leaves us with two choices of color for each vertex, with all constraints expressible as clauses with two variables, and hence can be solved by a two-satisfiability algorithm. Each of the other n problems yields a vertex that has been colored, and hence can be removed from the graph; moreover, this implies, for all vertices y adjacent to x along edges of the color $m(x)$, that $m(x)$ is not a legal choice of color for y. In the general step, we have some vertices with lists (of allowed values) of size two and some with lists of size three; we define the majority color $m(x)$ to be the color of at least one-third of the edges from x to vertices with lists of size three. Therefore the time to solve a problem with p vertices with lists of size three is

$$T(p) \leq (1 + pT(2p/3)) \cdot T_2(n),$$

where $T_2(n)$ is the (polynomial) time to solve the two-satisfiability problem arising from n vertices. The recurrence implies that $T(n)$ is quasi-poynomial $n^{O(\log n)}$. Despite the simplicity of this algorithm we have not been able to find a polynomial time algorithm. We have, however, obtained an algorithm of time complexity $n^{O(\log n/\log\log n)}$ [18].

The analogous problem with colors $0, 1$ corresponds to the well-known graph problem of recognition of *split graphs*, which has a simple polynomial time algorithm [22] (cf. also [24, 27]). When there are more than three colors, the problem is easily seen to be NP-complete. (Our problem can be shown to be no easier than the so-called *stubborn problem* from [6], which also resists classification.)

We shall conclude, using the dichotomy of $CSP(H)$ for three-inclusive structures, that *for a three-inclusive structure H, each problem $CSP_W(H)$ (respectively, $CSP^*_W(H)$) is NP-complete or solvable in quasi-polynomial time $n^{O(\log n)}$.* Thus we obtain a new kind of dichotomy—not between problems that are NP-complete and those that are solvable in polynomial time, but between problems that are NP-complete and those that are solvable in quasi-polynomial time. For convenience and brevity we shall call such a result a *quasi-dichotomy*. Our interest in quasi-dichotomy is motivated by arguments similar to those for dichotomy. It is generally expected that no NP-complete problem admits a quasi-polynomial algorithm, and if one NP-complete problem admitted such an algorithm, then so would all others. The relationship between the dichotomy conjecture and quasi-dichotomies was previously studied in the case of list matrix partition problems by Feder, Hell, Klein, and Motwani [17].

Megiddo and Vishkin [35] studied a problem on tournaments that has a quasi-polynomial algorithm. The problem asks for a minimum size dominating set in a tournament on n vertices. A minimum size dominating set always has at most $\log n$ vertices, so the problem can be solved in time $n^{O(\log n)}$. Megiddo and Vishkin showed that the problem of solving a satisfiability instance with $\log^2 n$ variables and n clauses reduces to the dominating set problem in tournaments. Papadimitriou and Yannakakis [37] showed that the dominating set problem for tournaments is complete for

the class $LOGSNP$, with the containment of classes $LOGSNP \subseteq LOGNP$, and both classes reduce in polynomial time to problems in $NP[\log^2 n] \cap DSPACE(\log^2 n)$. Here $NP[\log^2 n]$ is the class of problems that can be solved in nondeterministic polynomial time with only $O(\log^2 n)$ bits of nondeterminism.

The classes $LOGNP$ and $LOGSNP$ are classes of problems in NP involving guessing only $\log n$ elements in an n-element input structure I. More formally, the classes are defined by setting $LOGNP_0$ to be defined by a formula $\{I : \exists S \in [n]^{\log n}\ \forall x \in [n]^p\ \exists y \in [n]^q\ \forall j \in [\log n]\phi(I, s_j, x, y, j)\}$ and setting $LOGSNP_0$ to be defined by a formula $\{I : \exists S \in [n]^{\log n}\ \forall x \in [n]^p\ \exists j \in [\log n]\phi(I, s_j, x, j)\}$, where the S in $LOGSNP_0$ means *strict*, I is an input relation, x, y are tuples of first order variables ranging over $[n] = \{1, \ldots, n\}$, and ϕ is a quantifier-free first order expression. We then define $LOGNP$ to be the class of all languages that can be polynomially reduced to a problem in $LOGNP_0$, and similarly for $LOGSNP$ and $LOGSNP_0$.

All our quasi-polynomial algorithms involve guessing $O(\log n)$ vertices and $O(\log n)$ additional bits of information (thus with $O(\log^2 n)$ bits of nondeterminism) and then solving a resulting problem in deterministic polynomial time. Our quasi-polynomial problems are thus in $NP[\log^2 n]$. However, some of these problems are complete for P and are thus unlikely to be in $DSPACE(\log^2 n)$.

We shall also observe that constraint satisfaction problems for three-inclusive H show an interesting structure when the degree of the variables, i.e., the number of occurrences of each variable, is bounded. With maximum degree two, Boolean constraint satisfaction can express matching and delta-matroid intersection [7, 9, 11], while with maximum degree three it is the same as with unbounded degree. Similar results on degree restrictions for the case of graphs can be found in [15, 21, 25].

We conjecture that every three-inclusive list constraint satisfaction problem that is NP-complete remains NP-complete when restricted to degree three instances. Using existing results we verify this conjecture in four particular families of three-inclusive list constraint satisfaction problems.

A structure V^H is *W-set-full* if it is three-inclusive and if for any $w \in W$ and any (not necessarily distinct) pairs $(S_1, 1), \ldots, (S_w, 1)$ in V such that each $|S_i^H| \leq 3$ for $1 \leq i \leq w$, there exists a pair (R, w) in V and a permutation σ on $\{1, \ldots, w\}$, such that $S^H_{\sigma(1)} \times \cdots \times S^H_{\sigma(w)} \subseteq R^H$. These W-set-full structures H define *W-set-full constraint satisfaction problems* $CSP_W(H)$ and $CSP^*_W(H)$, which will turn out to be equivalent to the corresponding constraint satisfaction problem $CSP_1(H)$ and hence exhibit dichotomy just like the three-inclusive constraint satisfaction problems. When $W = \{1, 2\}$, we call a W-set-full structure (and problem) *pairwise-set-full*.

The *degree of x in a tuple $t = (t_1, \ldots, t_k)$* is the number of t_i that are equal to x. The *degree of x in a relation R_i* is the sum of the degrees of x in the tuples in R_i. The *degree of x in a structure G* is the sum of the degrees of x in the relations R_i of arity $\ell_i \geq 2$ in G. The *degree of a structure G* is the maximum degree of the elements of D^G in G.

2. Three-inclusive constraint satisfaction problems. A *polymorphism* of a structure H with domain $D = D^H$ is a mapping $g : D^k \to D$ such that for all relations R_i^H and all tuples $(x_{1j}, \ldots, x_{\ell_i j}) \in R_i^H$, for $1 \leq j \leq k$, we have $(y_1, \ldots, y_{\ell_i}) \in R_i^H$ for $y_t = g(x_{t1}, \ldots, x_{tk})$. Jeavons [28] showed that the complexity of a constraint satisfaction problem $CSP(H)$ is characterized, up to polynomial time reductions, by the complete set of polymorphisms that H has.

Bulatov [2] classified conservative constraint satisfaction problems $CSP(H)$ as polynomial or NP-complete. The polynomial cases are characterized by the existence

of three polymorphisms of H, namely $g_1 : D^2 \to D$, $g_2 : D^3 \to D$, and $g_3 : D^3 \to D$, satisfying the following properties:

(1) g_1, g_2, g_3 are *conservative*; that is, $g_1(x, y) \in \{x, y\}$, $g_2(x, y, z) \in \{x, y, z\}$, and $g_3(x, y, z) \in \{x, y, z\}$ for all $x, y, z \in D$.

For each $a, b \in D$ with $a \neq b$, at least one of the following holds:

(2) g_1 is *commutative* on $\{a, b\}$; that is, $g_1(a, b) = g_1(b, a) \in \{a, b\}$;

(3) g_2 is *majority* on $\{a, b\}$; that is, $g_2(a, a, a) = g_2(a, a, b) = g_2(a, b, a) = g_2(b, a, a) = a$ and $g_2(b, b, b) = g_2(b, b, a) = g_2(b, a, b) = g_2(a, b, b) = b$; and

(4) g_3 is *minority* on $\{a, b\}$; that is, $g_3(a, a, a) = g_3(a, b, b) = g_3(b, a, b) = g_3(b, b, a) = a$ and $g_3(b, b, b) = g_3(b, a, a) = g_3(a, b, a) = g_3(a, a, b) = b$.

Using this classification, we can now prove dichotomy of three-inclusive constraint satisfaction problems $CSP_1(H)$.

THEOREM 2.1. *If H is three-inclusive, then the constraint satisfaction problem $CSP_1(H)$ is polynomial time solvable or NP-complete.*

Proof. Let H be a three-inclusive structure; we define three auxiliary structures H^*, H', H'' on the same domain $D^{H^*} = D^{H'} = D^{H''} = D^H = D$, as follows.

- The structure H^* is obtained from H by replacing each nonunary relation R_i of H by the relations $R_i^a = R_i \cap (T_1 \times \cdots \times T_{\ell_i})$ for each choice a of unary relations $T_1, \ldots, T_{\ell_i} \in \mathcal{U}(H)$.
- The structure H' is obtained from H by removing all unary relations $S \in \mathcal{U}(H)$ with $|S| > 3$, and replacing each nonunary relation R_i of H by the relations R_i^a for each choice a of unary relations of the new structure H' (thus all $|T_i| \leq 3$).
- The structure H'' is obtained from H' by adding all missing unary relations S, so that $\mathcal{U}(H'')$ is the power set of D.

We note that all these structures are three-inclusive and that H'' is conservative. (Also observe that the three structures are over different vocabularies.)

We first prove that $CSP_1(H)$ and $CSP(H^*)$ are polynomially equivalent problems. To reduce $CSP_1(H)$ to $CSP(H^*)$, we transform an instance G of $CSP_1(H)$ to an instance G^* of $CSP(H^*)$; the structure G^* is obtained from G in a way similar to how H^* is obtained from H. (Each nonunary relation R_i is replaced by the relations R_i^a.) It is easy to see that a homomorphism of G to H is also a homomorphism of G^* to H^*. On the other hand, it is also the case that a homomorphism of G^* to H^* is a homomorphism of G to H, since G is 1-full. Thus G has a solution in $CSP_1(H)$ if and only if G^* has a solution in $CSP(H^*)$. To reduce $CSP(H^*)$ to $CSP_1(H)$, we transform an instance G of $CSP(H^*)$ to an instance G' of $CSP_1(H)$ obtained from G as follows. If a variable x is not constrained in G (does not appear in any relation), we remove it. For each $(x_1, x_2, \ldots, x_{\ell_i}) \in R_i^a$ in G, we impose the constraint $(x_1, x_2, \ldots, x_{\ell_i}) \in R_i$ as well as the unary constraints $x_i \in T_i$, $i = 1, \ldots, \ell_i$, in G'. It follows that G' is 1-full, i.e., an instance of $CSP_1(H)$; it is again easy to check that G has a solution in $CSP(H^*)$ if and only if G' has a solution in $CSP_1(H)$.

Since H'' is conservative, we know by [2] that the problem $CSP(H'')$ is polynomial time solvable (if there are polymorphisms g_1, g_2, g_3 of H'' as described above) or NP-complete. If $CSP(H'')$ is polynomial, then the three polymorphisms g_1, g_2, g_3 of H'' are also polymorphisms of H', as all relations in H' are in H''. (In particular, $CSP(H')$ is also polynomial.) Moreover, in this case, g_1, g_2, g_3 are also polymorphisms of H^*, and hence $CSP(H^*)$ is also polynomial. Indeed, consider a relation R_i^a of H^*, where a is the choice of unary relations $T_1, T_2, \ldots, T_{\ell_i} \in \mathcal{U}(H)$. Suppose $x = (x_1, \ldots, x_{\ell_i})$, $y = (y_1, \ldots, y_{\ell_i})$, and $z = (z_1, \ldots, z_{\ell_i})$ are in R_i^a. Let a' be the choice

of unary relations $T'_1, T'_2, \ldots, T'_{\ell_i} \in \mathcal{U}(H)$, where each $T'_i = \{x_i, y_i, z_i\}$. Since each $T'_i \subseteq T_i$ and H is three-inclusive, each $T'_i \in \mathcal{U}(H)$ and hence also $T'_i \in \mathcal{U}(H')$. Thus x, y, z are also in $R_i^{a'}$, and, since g_1, g_2, g_3 are polymorphisms of H', we must also have $(g_1(x_1, y_1), \ldots, g_1(x_{\ell_i}, y_{\ell_i})) \in R_i^{a'}$, $(g_2(x_1, y_1, z_1), \ldots, g_2(x_{\ell_i}, y_{\ell_i}, z_{\ell_i})) \in R_i^{a'}$, and $(g_3(x_1, y_1, z_1), \ldots, g_3(x_{\ell_i}, y_{\ell_i}, z_{\ell_i})) \in R_i^{a'}$. Since $R_i^{a'} \subseteq R_i^a$, g_1, g_2, g_3 are also polymorphisms of H^*, whence $CSP(H^*)$ (and $CSP_1(H)$) is polynomial.

Suppose instead $CSP(H'')$ is NP-complete. We will reduce $CSP(H'')$ to $CSP(H')$ and then to $CSP(H^*)$, proving that $CSP(H^*)$ (and $CSP_1(H)$) is also NP-complete. Given an instance G of $CSP(H'')$, we may assume that each variable x is constrained by some nonunary relation of G. (If a variable x is only in unary relations of G, then either the intersection of all the subsets of H corresponding to these unary relations is empty, in which case there is no solution, or the intersection is nonempty, and assigning any value from the intersection to x allows us to remove x from consideration.) Thus each variable x of G occurs in some jth position in some R_i^a in H', where a is the choice of some unary relations $T_1, T_2, \ldots, T_{\ell_i}$ with $|T_j| \leq 3$. We form G_1 from G by replacing each unary relation T containing x which is in H'' but not in H' by the unary relation $T' = T \cap T_j$ for x. Note that T' is a relation of H', since $T' \subseteq T_j$. It is easy to see that G_1 has a solution for $CSP(H')$ if and only if G has a solution for $CSP(H'')$. Therefore $CSP(H')$ is NP-complete. The final reduction from $CSP(H')$ to $CSP(H^*)$ is trivial, as each instance G of $CSP(H')$ is also an instance of $CSP(H^*)$. Therefore $CSP(H^*)$ (and $CSP_1(H)$) is also NP-complete. □

Let us call a structure H *subunary* if each nonunary relation R_i of H is included in some product $T_1 \times T_2 \times \cdots \times T_{\ell_i}$ of unary relations $T_1, T_2, \ldots, T_{\ell_i} \in \mathcal{U}(H)$. The structure H^* in the above proof is subunary, and this was the only property used in the proof. Thus we have also proved the following fact.

COROLLARY 2.2. *If H is a subunary three-inclusive structure, then $CSP(H)$ is polynomial or NP-complete.*

The above proof also shows that two structures which differ only in their unary relations S with $|S| > 3$ have the same behavior for CSP_1.

COROLLARY 2.3. *If H and K are three-inclusive structures on the same domain $D^H = D^K$ with the same nonunary relations and the same unary relations S with $|S| \leq 3$, then $CSP_1(H)$ and $CSP_1(K)$ are both NP-complete or both polynomial.*

Proof. In the proof of Theorem 2.1, the problems $CSP_1(H)$ and $CSP_1(K)$ yield the same derived problems: $CSP(H') = CSP(K')$ and $CSP(H'') = CSP(K'')$. □

We can apply the theorem to the following *three-inclusive list H-coloring problem*, where H is a fixed graph (with loops allowed). *An instance* is a graph G, together with lists $L(v)$, $v \in V(G)$, of vertices of H, such that each $|L(v)| \leq 3$. The *question* is whether or not there exists a homomorphism f of G to H with each $f(v) \in L(v)$.

COROLLARY 2.4 (see [15]). *Each three-inclusive list H-coloring problem is NP-complete or polynomial time solvable.*

In fact, in [15] we classify exactly which three-inclusive list H-coloring problems are polynomial—they turn out to be the same bi-arc graphs defined above.

3. Restriction to pairwise full (and strictly pairwise full) instances. Our general goal is to study the restriction of three-inclusive constraint satisfaction problems to W-full instances. For simplicity we shall first state and prove our results in the special case where $W = \{1, 2\}$, i.e., for pairwise full instances G. The general case will be treated in the next section, mostly just by pointing out the extra effort required.

We begin by observing that each three-inclusive constraint satisfaction problem $CSP_1(H)$ can be viewed as the restricted problem $CSP_{\{1,2\}}(H')$, where H' is obtained from H by adding the binary relation $D^H \times D^H$ (and the binary relation $D^G \times D^G$ to any instance G, without changing the problem). We prove a strong converse to this observation; in particular, this will show that $CSP_{\{1,2\}}(H)$ and $CSP^*_{\{1,2\}}(H)$ enjoy dichotomy for pairwise-set-full structures H.

THEOREM 3.1. *Let H be a pairwise-set-full structure.*

The problems $CSP_{\{1,2\}}(H), CSP^*_{\{1,2\}}(H)$, and $CSP_1(H)$ are all polynomial or all NP-complete.

Proof. The problem $CSP_1(H)$ is polynomial or NP-complete by Theorem 2.1 because the pairwise-set-full structure H is by definition three-inclusive. Every instance of $CSP^*_{\{1,2\}}(H)$ is an instance of $CSP_{\{1,2\}}(H)$, and every instance of $CSP_{\{1,2\}}(H)$ is an instance of $CSP_1(H)$; thus if $CSP_1(H)$ is polynomial, then all three problems are polynomial.

If, on the other hand, $CSP_1(H)$ is NP-complete, then by Corollary 2.3 so is $CSP_1(K)$ where K is obtained from H by deleting all unary relations $S \in \mathcal{U}(H)$ with $|S| > 3$. Since each instance of $CSP_1(K)$ is 1-full, we can prove, along the lines of the first part of the proof of Theorem 2.1, that $CSP_1(K)$ is polynomially equivalent to $CSP_1(K')$ where K' is subunary. To simplify the notation, let us assume that K itself is subunary.

For every pair of variables x, y in an instance G of $CSP_1(K)$, we have x and y constrained by unary relations S^G and T^G, respectively, such that $|S^K|, |T^K| \leq 3$. Since H and hence also K are pairwise-set-full, there is a binary relation R^K such that $S^K \times T^K \subseteq R^K$. We may thus add the pair (x, y) to R^G without affecting the existence of a solution. Therefore, for every pair of variables x, y in this modified instance G', there exists a binary relation $R^{G'}$ such that $(x, y) \in R^{G'}$. Hence G' is an instance of $CSP_{\{1,2\}}(K)$ and also of $CSP_{\{1,2\}}(H)$, and thus $CSP_{\{1,2\}}(H)$ is NP-complete.

We can transform the instance G of $CSP_{\{1,2\}}(H)$ into an instance G' of $CSP^*_{\{1,2\}}(H)$, in which each pair (x, y) occurs in only one binary relation R_i, as one of $(x, y) \in R_i$ or $(y, x) \in R_i$. Let r be the number of binary relations in H. For the transformation we shall use an edge colored complete bipartite graph B with the following properties:

- B has s vertices in each part;
- each edge of B has one of $2r$ colors;
- between any $s/3$ vertices in one part of B and any $s/3$ vertices in the other part of B, there is an edge of each of the $2r$ colors.

Given B, we replace each variable x in the instance with s variables x_i. Two variables x and y are joined by some $k \leq 2r$ choices of binary relations out of the r binary relations with two possible choices of orientation (from x to y or from y to x) for these binary relations. We may thus join the x_i to the y_i with a bipartite graph obtained from B with edges of k colors, by replacing the remaining $2r - k$ colors with some of the k chosen colors. A variable x is constrained by a unary relation $S \subseteq D^H$ with $|S| \leq 3$, and since H is pairwise-set-full, there exists a binary relation R in H such that $S \times S \subseteq R$. We add the pairs (x_i, x_j) to R for all the copies x_i and x_j of x. We now have an instance G' of $CSP^*_{\{1,2\}}(H)$. In a solution, out of the s variables x_i, at least $s/3$ of them will have the same value from S, which can then be used as a value for the original variable x. The k binary relations involving x and y appear between the $s/3$ corresponding choices of x_i and the $s/3$ corresponding choices of y_i; a solution to G' gives a solution to G. This completes the reduction and shows that

$CSP^*_{\{1,2\}}(H)$ is NP-complete as well.

It remains to show that such a bipartite graph B exists for any r. (Note that we may choose s as large as we want.) Given a complete bipartite graph with s vertices on each side, assign a color out of $2r$ colors to each edge uniformly at random. The probability that two sets of size $s/3$ will miss one of the $2r$ colors is at most $2r(1-1/(2r))^{(s/3)^2}$, and the number of choices of such subsets is $\binom{s}{s/3}^2$; hence it suffices to choose s large enough so that $2r(1-1/(2r))^{(s/3)^2}\binom{s}{s/3}^2 < 1/2$ to guarantee that with probability at least $1/2$ the complete bipartite graph B will be assigned $2r$ edge colors with the desired property. (The quantity on the left is easily seen to converge to 0 with increasing s for any constant r.) □

Thus pairwise-set-full constraint satisfaction problems enjoy dichotomy when restricted to either the pairwise full or the strictly pairwise full instances G. We have not succeeded in obtaining a similar dichotomy for three-inclusive constraint satisfaction in general. We can, however, use Theorem 3.1 to obtain a quasi-dichotomy, because of the following result.

THEOREM 3.2. *Let H be a three-inclusive structure, and let $\mathcal{H}$ be the set of all pairwise-set-full structures H' with the same domain and the same nonunary relations as H, and such that each $S \in \mathcal{U}(H')$ with $|S| \le 3$ is also in $\mathcal{U}(H)$.*

If all $CSP_1(H')$ for $H' \in \mathcal{H}$ are polynomial, then both $CSP_{\{1,2\}}(H)$ and $CSP^*_{\{1,2\}}(H)$ are quasi-polynomial. If some $CSP_1(H')$ for $H' \in \mathcal{H}$ is NP-complete, then both $CSP_{\{1,2\}}(H)$ and $CSP^*_{\{1,2\}}(H)$ are NP-complete.

COROLLARY 3.3. *For each three-inclusive structure H, the problems $CSP_{\{1,2\}}(H)$ and $CSP^*_{\{1,2\}}(H)$ are both NP-complete or both quasi-polynomial.*

Proof. We shall focus on proving the theorem for $CSP_{\{1,2\}}(H)$; the proof for $CSP^*_{\{1,2\}}(H)$ is similar.

We shall transform $CSP_{\{1,2\}}(H)$ to problems $CSP_1(H')$ for $H' \in \mathcal{H}$ in such a way that if any of these problems is NP-complete, then so is $CSP_{\{1,2\}}(H)$. In the transformation, each instance G of $CSP_{\{1,2\}}(H)$ with n variables gives rise to $n^{O(\log n)}$ instances G' of problems $CSP_1(H')$, so that if all these problems are polynomial, then $CSP_{\{1,2\}}(H)$ is quasi-polynomial.

The problems $CSP_1(H')$ will have a pairwise-set-full structure H' obtained from H by repeatedly replacing certain unary relations S with all proper subsets $T \subset S$. In fact, each structure $H' \in \mathcal{H}'$ will be *strongly pairwise-set-full*, in the sense that it is three-inclusive and for any unary relations S_1, S_2 (not just those with $|S_i| \le 3$) there is a binary relation R with $S_1 \times S_2$ or $S_2 \times S_1$ contained in R.

To obtain these $n^{O(\log n)}$ instances G' (and the problems $CSP_1(H')$ to which they correspond), we proceed as follows. Let A and B be two subsets of D^H such that no binary relation of H contains $A \times B$ or $B \times A$. Since we want to make our structure strongly pairwise-set-full, we shall ensure that there is no $S \in \mathcal{U}(H)$ that contains A, or no $T \in \mathcal{U}(H)$ that contains B. (Later on, we shall keep ensuring this property for other pairs A, B of sets, on structures H' previously obtained from H.)

Let L be the set of variables (elements of D^G) which are in unary constraints S^G such that $A \subseteq S^H$, and let M be the set of variables (elements of D^G) in unary constraints T^G such that $B \subseteq T^H$. Let $\ell = |L|$, $m = |M|$, and let r again denote the number of binary relations in H. For every variable $v \in L$ there exists a binary relation R_i such that (v, w) or (w, v) is in R_i^G for at least $m/(2r)$ variables $w \in M$. We now transform the instance G of $CSP_{\{1,2\}}(H)$ (later on, the instance G' of the current $CSP_{\{1,2\}}(H')$) into $\ell + 1$ new problems. For each binary relation R_i we

choose variables $a_i, a_i' \in A$, $b_i, b_i' \in B$ so that $(a_i, b_i) \notin R_i^H, (b_i', a_i') \notin R_i^H$. In the first derived problem we change, for each variable $v \in L$, the unary constraint S^G to the unary constraint $S^G - a_i$ or $S^G - a_i'$, depending on whether R_i was chosen above for pairs (v, w) or (w, v). In the remaining ℓ derived problems, we change, for one variable $v \in L$, the constraint S^G to the constraint $\{a_i\}^G$ or $\{a_i'\}^G$, depending on whether R_i was chosen above for pairs (v, w) or (w, v). We also change, for all variables $w \in M$ such that (v, w) (respectively, (w, v)) is constrained by R_i^G, the constraint T^G to the constraint $T^G - b_i$ (respectively, $T^G - b_i'$). Recall that for each choice of v, the number of such variables w is at least $m/(2r)$. Thus an instance with $(|L|, |M|) = (\ell, m)$ has been replaced with one instance with $(|L|, |M|) = (0, m)$ and ℓ instances with $(|L|, |M|) = (\ell - 1, m(1 - 1/(2r)))$. Repeating this process $-\log m / \log(1 - 1/(2r)) = O(\log n)$ times, we obtain $n^{O(\log n)}$ instances that each have $\ell = 0$ or $m = 0$; that is, either no unary constraint contains A or no unary constraint contains B. The problem $CSP_{\{1,2\}}(H)$ is correspondingly replaced by problems $CSP_{\{1,2\}}(H')$. Note that no unary constraint in H' contains A or no unary constraint in H' contains B, and all the unary constraints in H' are subsets of unary constraints in H.

We repeat this process for each pair of sets (A, B) as above. In the end, we obtain $n^{O(\log n)}$ instances and corresponding structures H', such that H' is strongly pairwise-set-full. If all such $CSP_1(H')$ are polynomial, then we can solve each of these $n^{O(\log n)}$ resulting problems, obtaining a quasi-polynomial algorithm for $CSP_{\{1,2\}}(H)$. Otherwise, at least one such $CSP_1(H')$ is NP-complete. Let H'' be obtained from H' by removing all unary constraints $S^{H'}$ with $|S^{H'}| > 3$. By Corollary 2.3 the pairwise-set-full constraint satisfaction problem $CSP_1(H'')$ is also NP-complete; note that all the unary constraints of H'' are in $\mathcal{U}(H)$. By Theorem 3.1 the problem $CSP_{\{1,2\}}(H'')$ is also NP-complete, and its instances are also instances of the original problem $CSP_{\{1,2\}}(H)$; thus this problem is also NP-complete. □

Recall the list matrix partition problems $CSP_{1,2}^*(H)$ defined in the introduction and studied in [6, 17, 18, 24, 27]. We obtain the following corollaries. (The first corollary has been anticipated in [17].)

COROLLARY 3.4. *Each list matrix partition problem is NP-complete or quasi-polynomial.*

We call a list matrix partition problem $CSP_{1,2}^*(H)$ *three-inclusive* if the instances are graphs G with lists of size at most three.

COROLLARY 3.5. *Each three-inclusive list matrix partition problem is NP-complete or quasi-polynomial.*

The analogue of Corollary 2.3 follows from Theorems 3.1 and 3.2 by an application of Corollary 2.3.

COROLLARY 3.6. *If H and K are pairwise-set-full structures on the same domain $D^H = D^K$ with the same nonunary relations and the same unary relations S with $|S| \leq 3$, then $CSP_{\{1,2\}}(H)$ and $CSP_{\{1,2\}}(K)$ are both NP-complete or both polynomial.*

If H and K are three-inclusive structures on the same domain $D^H = D^K$ with the same nonunary relations and the same unary relations S with $|S| \leq 3$, then $CSP_{\{1,2\}}(H)$ and $CSP_{\{1,2\}}(K)$ are both NP-complete or both quasi-polynomial.

The same conclusions hold for the strict versions $CSP_{\{1,2\}}^*(H)$ and $CSP_{\{1,2\}}^*(K)$.

4. Restriction to W-full (and strictly W-full) instances. Theorems 3.1 and 3.2 generalize to W-full structures, and the proofs are similar. Here we briefly state the results and describe the additional effort required to prove them.

Here is the generalization of Theorem 3.1.

THEOREM 4.1. *Let H be a W-set-full structure.*

The problems $CSP_W(H)$, $CSP^*_W(H)$, and $CSP_1(H)$ are all polynomial or all NP-complete.

Proof. We prove the theorem for $W = \{1, w\}$ for a single $w \geq 2$. For general W we can carry out this proof for each value $w \in W$ with $w \geq 2$ in turn. Thus we assume $W = \{1, w\}$ and proceed as in Theorem 3.1. If $CSP_1(H)$ is polynomial, then trivially both $CSP^*_W(H)$ and $CSP_W(H)$ are polynomial. If $CSP_1(H)$ is NP-complete, then by Corollary 2.3 we may restrict $CSP_1(H)$ to three possible values in H per variable (using the unary relations) and still obtain an NP-complete problem. Again, for every collection of distinct variables $x_1, \ldots, x_w$ in D^G, where x_i has at most three possible values given by S^H_i, there exists a relation R^H and a permutation σ such that $S^H_{\sigma(1)} \times \cdots \times S^H_{\sigma(w)} \subseteq R^H$ (since H is W-set-full), so the instance G can be made W-full as before and thus becomes an instance of the W-set-full problem $CSP_W(H)$. Therefore $CSP_W(H)$ is also NP-complete.

It remains to enforce strictness to obtain an instance of $CSP^*_W(H)$. As in Theorem 3.1, we make s copies of each variable and use an auxiliary construction to define the relations on this enlarged set of variables. Specifically, we apply Lemma 4.2 below, with $t = w!r$ (corresponding to the $w!$ choices of permutations σ and r choices of relations R^H in the definition of strictly W-full structures). We then obtain an instance equivalent to G that is strictly W-full, whence $CSP^*_W(H)$ is also NP-complete. □

LEMMA 4.2. *Let K_{ws} be the structure with ws elements x_{ij}, $1 \leq i \leq w$, $1 \leq j \leq s$, and relation R consisting of all w^s tuples $(x_{1i_1}, \ldots, x_{wi_w})$. For every pair of integers $w \geq 2$ and $t \geq 1$, there exists an integer $s \geq 1$ and a coloring of the tuples in K_{ws} with t colors such that for every choice of w subsets $S_1, \ldots, S_w$ with $S_i \subseteq \{x_{i1}, \ldots, x_{is}\}$ and $|S_i| \geq s/3$, we have that $S_1 \times \cdots \times S_w$ contains tuples of all t colors.*

Proof. Assign a color out of t colors to each tuple uniformly at random. The probability that a choice of w sets S_i with $|S_i| \geq s/3$ will miss one of the t colors is at most $t(1 - 1/t)^{(s/3)^t}$ and the number of choices of such subsets is $\binom{s}{s/3}^t$, so it suffices to choose s large enough so that $t(1 - 1/t)^{(s/3)^w} \binom{s}{s/3}^w < 1/2$ to guarantee that with probability at least $1/2$ the structure K_{ws} will be assigned t tuple colors with the desired property. □

The generalization of Theorem 3.2 takes the following form.

THEOREM 4.3. *Let H be a three-inclusive structure, and let $\mathcal{H}$ be the set of all W-set-full structures H' that have the same nonunary relations as H and such that each $S \in \mathcal{U}(H')$ with $|S| \leq 3$ is also in $\mathcal{U}(H)$.*

If all $CSP_1(H')$ for $H' \in \mathcal{H}$ are polynomial, then both $CSP_W(H)$ and $CSP^*_W(H)$ are quasi-polynomial. If some $CSP_1(H')$ for $H' \in \mathcal{H}$ is NP-complete, then both $CSP_W(H)$ and $CSP^*_W(H)$ are NP-complete.

COROLLARY 4.4. *For each three-inclusive structure H, the problems $CSP_W(H)$ and $CSP^*_W(H)$ are both NP-complete or both quasi-polynomial.*

In the proof of Theorem 4.3 we shall again treat each $w \in W$ in turn; thus we shall assume that $W = \{1, w\}$.

Let G be a W-full structure, let $L_1, \ldots, L_w$ be (not necessarily disjoint) nonempty subsets of D^G, and let $\alpha = 1/(w!r)$, where r is the number of w-ary relations R^G_i. Say that a rooted tree T is a *select-tree* if the vertices of T at depth $1 \leq i \leq w$ correspond to the variables in L_i, with the root ρ at depth 0. For a w-ary relation R^G_i, a permutation σ of $\{1, \ldots, w\}$, and $0 < \beta < 1$, let an (R_i, σ, β)-*tree* be a select-tree T such that each path $(\rho, x_1, \ldots, x_w)$ in T has $(x_{\sigma(1)}, \ldots, x_{\sigma(w)}) \in R^G_i$ or has

$x_i = x_j$ for some $1 \le i < j \le w$, and each vertex of T at depth $0 \le i < w$ has at least $\beta|L_{i+1}|$ children.

LEMMA 4.5. *For every W-full structure G and sets $L_1, \ldots, L_w$ as above, there exist R_i and σ such that G has an (R_i, σ, β)-tree for $\beta = \alpha/(4w)$.*

Proof. Let $z = \ell_1 \cdots \ell_w$ for $\ell_i = |L_i|$. There exist a relation R_i^G and a permutation σ of $\{1, \ldots, w\}$ such that at least $\alpha z = z/(w!r)$ tuples $(x_1, \ldots, x_w)$ with $x_i \in L_i$ have $(x_{\sigma(1)}, \ldots, x_{\sigma(w)}) \in R_i^G$ or have $x_i = x_j$ for some $1 \le i < j \le w$. By Markov's inequality, there are at least $\beta\ell_1$ elements $x \in L_1$ such that the number of w-tuples $(x_1, \ldots, x_w)$ with $x = x_1$ and $x_i \in L_i$ having $(x_{\sigma(1)}, \ldots, x_{\sigma(w)}) \in R_i^G$, or having $x_i = x_j$ for some $1 \le i < j \le w$, is at least $\alpha'(z/\ell_1)$ for $\alpha' = (\alpha - \beta)/(1 - \beta)$, for any β with $0 < \beta < \alpha$. (Markov's inequality states that a nonnegative random variable X has value at most $tE(X)$ with probability at least $1 - \frac{1}{t}$ for each choice of $t > 1$.) By our special choice of β, we have $\alpha''(1 - 1/(2(w-1))) \ge (\alpha'' - \beta)/(1 - \beta)$ for each $\alpha'' \ge \alpha/2$. Thus repeatedly replacing α by α' a total of $w - 1$ times reduces α by a factor of at least $(1 - 1/(2(w-1)))^{w-1} \ge 1/2$, so that at any point in time the new value α'' satisfies $\alpha'' \ge \alpha/2$. We therefore obtain at least $\beta\ell_1$ children of the root, and for each chosen child x, we have z replaced with z/ℓ_1 and αz replaced with $\alpha'(z/\ell_1)$, giving again at least $\beta\ell_2$ children of x, and by induction, each vertex of T at depth $0 \le i < w$ has at least $\beta\ell_{i+1}$ children for $\beta = \alpha/(4w)$. □

Proof of Theorem 4.3. We again focus on proving the statement for $CSP_W(H)$, the proof for $CSP_W^*(H)$ being similar. We proceed as in Theorem 3.2. Assume H is a three-inclusive structure, and transform $CSP_W(H)$ to problems $CSP_1(H')$ for W-set-full structures $H' \in \mathcal{H}$ such that if any of these problems is NP-complete then $CSP_W(H)$ is also NP-complete. In the transformation, each instance G of $CSP_W(H)$ gives rise to $n^{O(\log n)}$ instances G' of the problems $CSP_1(H')$, so that if all these problems are polynomial, then $CSP_W(H)$ is quasi-polynomial. Recall that we assume $W = \{1, w\}$. The problems $CSP_1(H')$ will again have a W-set-full structure H' obtained from H by repeatedly replacing certain unary relations S with all proper subsets $T \subset S$.

To obtain these $n^{O(\log n)}$ instances G' (and the problems $CSP_1(H')$ to which they correspond), we shall ensure that if some elements $a_{ij\sigma} \in D^H$, one for each of the r w-ary relations R_i^H, $1 \le j \le w$, and each fixed permutation σ on $\{1, \ldots, w\}$, are such that $(a_{i\sigma(1)\sigma}, \ldots, a_{i\sigma(w)\sigma}) \notin R_i^H$, and we let $A_j = \{a_{ij\sigma} : 1 \le i \le r\}$, then for some $1 \le j \le w$ the unary constraints on the instances G' (and corresponding structures H', both described below) do not contain A_j. This will guarantee that we obtain a W-set-full problem.

Let L_j be the set of variables with unary constraints S^G such that $A_j \subseteq S^H$ for $1 \le j \le w$, and let $\ell_j = |L_j|$. By Lemma 4.5 G has an (R_i, σ, β)-tree T for $\beta = 1/(4w!wr)$. For every solution f to the instance G, there must exist a path $(\rho, x_1, \ldots, x_{t-1})$ in T with $1 \le t \le w$, with the tree vertices x_u corresponding to distinct elements of D^G, that gives to x_u value $a_{iu\sigma}$ for $1 \le u < t$, but does not give value $a_{it\sigma}$ to any child x_t of x_{t-1} corresponding to an element distinct from the elements for these x_u, $1 \le u < t$.

Since the number of such children of x_{t-1} is at least $\beta\ell_t$, and the solution f belongs to a family of solutions that assigns the values $a_{iu\sigma}$ for $1 \le u < t$ and removes the value $a_{it\sigma}$ from the lists of these children, we obtain a new instance with ℓ_t replaced by $(1-\beta)\ell_t$. This simplification can be performed at most $-\log(\ell_1 \cdots \ell_w)\log(1-\beta) \le -w \log n \log(1 - 1/(4w!wr)) = O(\log n)$ times, and the number of possible choices of paths $(\rho, x_1, \ldots, x_{t-1})$ for each iteration is at most $2\ell_1 \cdots \ell_w \le 2n^w = n^{O(1)}$, so we obtain $n^{O(\log n)}$ instances that have some $\ell_j = 0$, thus satisfying the condition for the

corresponding A_j.

We repeat this process for each tuple of sets $(A_1, \ldots, A_w)$ as above. In the end, we obtain $n^{O(\log n)}$ instances (and corresponding structures H') that satisfy the stated condition for each tuple $(A_1, \ldots, A_w)$. Thus for each resulting instance G' of $CSP_1(H')$, the structure H' is strongly W-set-full. If all such $CSP_1(H')$ are polynomial, then we can solve each of these $n^{O(\log n)}$ resulting problems. If one such $CSP_1(H')$ is NP-complete, then by Corollary 2.3 the W-set-full structure H'', obtained from H' by restricting all unary relations to subsets of size at most three, yields a problem $CSP_1(H'')$ that is also NP-complete, and all these subsets of size at most three are in the original structure H. By Theorem 4.1 the problem $CSP_W(H'')$ is NP-complete, and since the instances of $CSP_W(H'')$ are also instances of $CSP_W(H)$, the problem $CSP_W(H)$ is also NP-complete.

We also obtain the analogue of Corollary 3.6, which follows from Theorems 4.1 and 4.3 by an application of Corollary 2.3.

COROLLARY 4.6. *If H and K are W-set-full structures on the same domain $D^H = D^K$ with the same nonunary relations and the same unary relations S with $|S| \leq 3$, then $CSP_W(H)$ and $CSP_W(K)$ are both NP-complete or both polynomial.*

If H and K are three-inclusive structures on the same domain $D^H = D^K$ with the same nonunary relations and the same unary relations S with $|S| \leq 3$, then $CSP_W(H)$ and $CSP_W(K)$ are both NP-complete or both quasi-polynomial.

The same conclusions hold for the strict versions $CSP^*_W(H)$ and $CSP^*_W(K)$.

As in the proof of Theorem 4.1, every three-inclusive problem $CSP_W(H)$ can be transformed into a polynomially equivalent problem $CSP^*_W(H)$. However, it is not clear how to accomplish the converse transformation in general; so the latter family of problems may define a more general family, up to polynomial time equivalence. But the two families of problems are the same, up to quasi-polynomial time equivalence, since all problems considered are either quasi-polynomial (hence equivalent up to quasi-polynomial reductions) or NP-complete (hence equivalent up to polynomial reductions).

5. Bounded degree problems. Here we consider restricting the three-inclusive constraint satisfaction problems $CSP_1(H)$ to instances G with bounded degree. We are led to propose the following conjecture.

CONJECTURE 5.1. *Every three-inclusive list constraint satisfaction problem $CSP_1(H)$ that is NP-complete remains NP-complete when restricted to instances of degree three.*

We denote by $CSP^3_1(H)$ the restriction of $CSP_1(H)$ to instances of degree three. We now briefly explain four special cases in which the conjecture holds. All are based on existing results.

We shall say that a three-inclusive constraint satisfaction problem $CSP_1(H)$ *d-simulates* a three-inclusive constraint satisfaction problem $CSP_1(H')$ if for every relation $R_i^{H'}$, there is an instance G_i of $CSP_1(H)$ of degree at most d containing ℓ_i particular variables $x_1, \ldots, x_{\ell_i}$ of degree at most one, such that $(a_1, \ldots, a_{\ell_i}) \in R_i^{H'}$ if and only if there exists a homomorphism f of G_i to H such that each $f(x_i) = a_i$. If $CSP_1(H)$ d-simulates $CSP_1(H')$ with $d \leq 3$, then there is a polynomial reduction of the problem $CSP^3_1(H')$ to $CSP^3_1(H)$, obtained by substituting each occurrence of a relation $R_i^{H'}$ in an instance G' of $CSP^3_1(H')$ by a copy of the instance G_i corresponding to R_i, thus obtaining an instance G of $CSP^3_1(H)$. Therefore showing the conjecture for $CSP^3_1(H')$ implies the conjecture for $CSP^3_1(H)$.

In the first case, we focus on the case of *Boolean constraint satisfaction problems*,

i.e., $CSP_1(H)$, where D^H has two elements $(0,1)$. The theorem stated below was first proved by Dalmau and Ford [7] and by Feder and Ford [11], using a result of Feder [9]. We briefly explain the proof as it is the basis for the other cases.

THEOREM 5.2 (see [7, 11]). *Conjecture* 5.1 *holds for Boolean constraint satisfaction problems.*

Proof. In other words, we claim that if for a three-inclusive H with $D = D^H = \{0,1\}$ the problem $CSP_1(H)$ is NP-complete, then so is its restriction $CSP_1^3(H)$ to instances of degree three. It is shown in [9] that each NP-complete case of Boolean constraint satisfaction problems $CSP_1(H)$ 2-simulates the problem $CSP_1(H')$ which contains the ternary relation $D^3-\{(0,0,0)\}$ (corresponding to the disjunction $x\vee y\vee z$), the ternary relation $D^3-\{(1,1,1)\}$ (corresponding to the disjunction $\overline{x}\vee\overline{y}\vee\overline{z}$), and the binary relation $D^2-\{(1,0)\}$ (corresponding to the implication $x\to y$).

This simulation yields a reduction from the NP-complete problem of 3-satisfiability to the problem $CSP_1^3(H')$, as we may include a variable x in any number k of clauses by using k auxiliary variables $x_1, x_2, \ldots, x_k$ with implications $x_1 \to x_2, x_2 \to x_3, \ldots, x_{k-1} \to x_k, x_k \to x_1$. These implications involve two occurrences for each x_i, and the third occurrence of each of these k variables x_i that all take the same value can be used for the k occurrences of x in clauses $x \vee y \vee z$ or $\overline{x} \vee \overline{y} \vee \overline{z}$. □

In the second case, we focus on three-inclusive constraint satisfaction problems $CSP_1(H)$ in which all $S \in \mathcal{U}(H)$ have $|S| \leq 2$. These problems correspond to list constraint satisfaction problems with lists of size at most two, and we call these problems *small-list constraint satisfaction problems.*

THEOREM 5.3. *Conjecture* 5.1 *holds for small-list constraint satisfaction problems.*

Proof. It is shown in [10] that each NP-complete case of the small-list constraint satisfaction problems 2-simulates a problem $CSP_1(H')$ for a structure H' containing the unary relations $\{0_i, 1_i\}$, for $1 \leq i \leq r$, and the binary relations $(\{0_i,1_i\} \times \{0_j,1_j\}) - \{(1_i,0_j)\}$ (corresponding to the implications $x_i \to x_j$) or the binary relations $(\{0_i,1_i\} \times \{0_j,1_j\}) - \{(1_i,0_j),(0_i,1_j)\}$ (corresponding to the equalities $x_i = x_j$) for each $1 \leq i,j \leq r$, $i \neq j$. It is also shown in [10] that the problem $CSP_1(H)$ with $D^H = \{0,1\}$, obtained from $CSP_1(H')$ by identifying all sets $\{0_i,1_i\}$ with $\{0,1\}$, is NP-complete.

We may thus represent kr occurrences of a Boolean variable x, with k of these occurrences having unary relation $\{0_i,1_i\}$ for each $1 \leq i \leq r$, by a cycle of implications $x_i \to x_j$ or equalities $x_i = x_j$, as in the proof of Theorem 5.2, involving variables x_{ij} having unary relation $\{0_i,1_i\}$ for $1 \leq i \leq r$ and $1 \leq j \leq k$. This completes the reduction from the NP-complete problem $CSP_1(H)$ obtained by identifying all $\{0_i,1_i\}$ with $\{0,1\}$ to $CSP_1^3(H')$. □

The third case concerns list homomorphism problems for graphs (with loops allowed). The problem $CSP_1(H)$, where H is three-inclusive and consists of a single symmetric binary relation, will be called a *graph list homomorphism problem.* The following theorem has been proved in [15].

THEOREM 5.4 (see [15]). *If H is a bi-arc graph, then $CSP_1^3(H)$ is polynomial; otherwise, $CSP_1^3(H)$ is NP-complete.*

Since this classification agrees exactly with that for $CSP_1(H)$ from [14], we obtain the desired corollary.

COROLLARY 5.5. *Conjecture* 5.1 *holds for graph list homomorphism problems.*

The last case deals with N-free binary relations. A binary relation B is called *N-free* if $(x,z),(y,z),(y,t) \in B$ imply either $x = y$ or $z = t$. A *binary N-free constraint*

satisfaction problem is a three-inclusive problem $CSP_1(H)$ where $\mathcal{U}(H)$ contains all S with $|S| \leq 3$, all binary relations of H are N-free, and H has no relations of higher arity.

THEOREM 5.6. *Conjecture* 5.1 *holds for binary N-free constraint satisfaction problems.*

Proof. It is shown in [19] that each NP-complete binary N-free constraint satisfaction problem 3-simulates a problem $CSP_1(H)$ containing the relation $\{(1_A, 0_B, 0_C), (0_A, 1_B, 0_C), (0_A, 0_B, 1_C)\}$, with $|\{0_A, 1_A\}| = |\{0_B, 1_B\}| = |\{0_C, 1_C\}| = 2$.

This relation may further be used to 3-simulate a problem $CSP_1(H')$ that also contains the relation $\{(1_A, 0_B), (0_A, 1_B)\}$ (corresponding to the inequality $x_A \neq x_B$), by setting x_C to value 0_C, and similarly $\{(1_B, 0_C), (0_B, 1_C)\}$ (corresponding to the inequality $x_B \neq x_C$, and also contains the relation $\{(0_A, 0_C), (1_A, 1_C)\}$ (corresponding to the equality $x_A = x_C$), obtained by combining $x_A \neq x_B$ with $x_B \neq x_C$, and similarly the relation $\{(0_A, 0_B), (1_A, 1_B)\}$ (corresponding to the equality $x_A = x_B$). Thus we may set $x_{A1} = \cdots = x_{Ak} = x_{B1} \cdots x_{Bk} = x_{C1} \cdots x_{Ck}$ as a chain of equalities that uses only two of the three allowed occurrences of each variable. This allows us to identify the three unary relations $\{0_i, 1_i\}$ with a single unary relation $\{0, 1\}$, with k occurrences per variable, for the NP-complete Boolean constraint satisfaction problem one-in-three SAT, with relation $\{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$ (obtained from $\{(1_A, 0_B, 0_C), (0_A, 1_B, 0_C), (0_A, 0_B, 1_C)\}$ by the identification $x_A = x_B = x_C$). Thus $CSP_1^3(H')$ and $CSP_1^3(H)$, with degree at most three, are also NP-complete. ∎

Acknowledgment. We are grateful to the referees for guiding us to present these results more clearly.

REFERENCES

[1] J. BANG-JENSEN, P. HELL, AND G. MACGILLIVRAY, *On the complexity of coloring by superdigraphs of bipartite graphs*, Discrete Math., 109 (1992), pp. 27–44.

[2] A. A. BULATOV, *Tractable conservative constraint satisfaction problems*, in Proceedings of the 18th IEEE Annual Symposium on Logic in Computer Science (LICS 2003), 2003, pp. 321–330.

[3] A. A. BULATOV, *A dichotomy theorem for constraints on a three-element set*, in Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science 2002, pp. 649–658.

[4] A. A. BULATOV AND P. JEAVONS, *Algebraic Structures in Combinatorial Problems*, Technical report MATH-AL-4-2001m, Technische Universität Dresden, Dresden, Germany, 2001.

[5] A. BULATOV, P. JEAVONS, AND A. KROKHIN, *Classifying the complexity of constraints using finite algebras*, SIAM J. Comput., 34 (2005), pp. 720–742.

[6] K. CAMERON, E. E. ESCHEN, C. T. HOÁNG, AND R. SRITHARAN, *The list partition problem for graphs*, in Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), New Orleans, LA, 2004, pp. 384–392.

[7] V. DALMAU AND D. FORD, *Generalized satisfiability with limited occurrences per variable: A study through delta-matroid parity*, in Mathematical Foundations of Computer Science (MFCS 2003), Lecture Notes in Comput. Sci. 2747, Springer, Berlin, 2003, pp. 358–367.

[8] R. DECHTER, *Constraint networks*, in The Encyclopedia of Artificial Intelligence, Wiley, New York, 1992, pp. 276–285.

[9] T. FEDER, *Fanout limitations on constraint systems*, Theoret. Comput. Sci., 255 (2001), pp. 281–293.

[10] T. FEDER, *Classification of homomorphisms to oriented cycles and of k-partite satisfiability*, SIAM J. Discrete Math., 14 (2001), pp. 471–480.

[11] T. FEDER AND D. FORD, *Classification of bipartite Boolean constraint satisfaction through delta-matroid intersection*, SIAM J. Discrete Math., 20 (2006), pp. 372–394.

[12] T. FEDER AND P. HELL, *List homomorphisms to reflexive graphs*, J. Combin. Theory Ser. B, 72 (1998), pp. 236–250.

[13] T. FEDER, P. HELL, AND J. HUANG, *List homomorphisms and circular arc graphs*, Combinatorica, 19 (1999), pp. 487–505.

[14] T. Feder, P. Hell, and J. Huang, *Bi-arc graphs and the complexity of list homomorphisms*, J. Graph Theory, 42 (1999), pp. 61–80.
[15] T. Feder, P. Hell, and J. Huang, *List homomorphisms of graphs with bounded degrees*, Discrete Math., to appear.
[16] T. Feder, P. Hell, and J. Huang, *List Homomorphisms to Reflexive Digraphs*, manuscript, 2004.
[17] T. Feder, P. Hell, S. Klein, and R. Motwani, *List partitions*, SIAM J. Discrete Math., 16 (2003), pp. 449–478.
[18] T. Feder, P. Hell, D. Král, and J. Sgall, *Two algorithms for general list matrix partitions*, in Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), Vancouver, BC, 2005, pp. 870–876.
[19] T. Feder, F. Madelaine, and I. A. Stewart, *Dichotomies for classes of homomorphism problems involving unary functions*, Theoret. Comput. Sci., 314 (2004), pp. 1–43.
[20] T. Feder and M. Y. Vardi, *The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory*, SIAM J. Comput., 28 (1998), pp. 57–104.
[21] A. Galluccio, P. Hell, and J. Nešetřil, *The complexity of H-coloring of bounded degree graphs*, Discrete Math., 222 (2000), pp. 101–109.
[22] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.
[23] P. Hell, *From graph colouring to constraint satisfaction—there and back again*, in Topics in Discrete Mathematics, Dedicated to Jarik Nešetřil on the occasion of his 60th birthday, Algorithms Combin. 26, M. Klazar et al., eds., Springer, Berlin, 2006, pp. 407–432.
[24] P. Hell, S. Klein, L. T. Nogueira, and F. Protti, *Partitioning chordal graphs into independent sets and cliques*, Discrete Appl. Math., 141 (2004), pp. 185–194.
[25] P. Hell and J. Nešetřil, *Counting list homomorphisms and graphs with bounded degrees*, in Graphs, Morphisms, and Statistical Physics, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 63, J. Nešetřil and P. Winkler, eds., AMS, Providence, RI, 2004, pp. 105–112.
[26] P. Hell and J. Nešetřil, *On the complexity of H-coloring*, J. Combin. Theory Ser. B, 48 (1990), pp. 92–110.
[27] P. Hell and J. Nešetřil, *Graphs and Homomorphisms*, Oxford University Press, Oxford, UK, 2004.
[28] P. Jeavons, *On the algebraic structure of combinatorial problems*, Theoret. Comput. Sci., 200 (1998), pp. 185–204.
[29] P. Jeavons, D. Cohen, and M. Gyssens, *Closure properties of constraints*, J. ACM, 44 (1997), pp. 527–548.
[30] R. E. Ladner, *On the structure of polynomial time reducibility*, J. Assoc. Comput. Mach., 22 (1975), pp. 155–171.
[31] P. Lincoln and J. C. Mitchell, *Algorithmic aspects of type inference with subtypes*, in Proceedings of the 19th ACM Symposium on Principles of Programming Languages, 1992, pp. 293–304.
[32] V. Kumar, *Algorithms for constraint-satisfaction problems*, AI Magazine, 13 (1992), pp. 32–44.
[33] P. Meseguer, *Constraint satisfaction problems: An overview*, AICOM, 2 (1989), pp. 3–16.
[34] J. C. Mitchell, *Coercion and type inference (summary)*, in Proceedings of the 11th ACM Symposium on Principles of Programming Languages, 1984, pp. 175–185.
[35] N. Megiddo and U. Vishkin, *On finding a minimum dominating set in a tournament*, Theoret. Comput. Sci., 61 (1988), pp. 307–316.
[36] U. Montanari, *Networks of constraints: Fundamental properties and applications to picture processing*, Inform. Sci., 7 (1974), pp. 95–132.
[37] C. H. Papadimitriou and M. Yannakakis, *On limited nondeterminism and the complexity of the V-C dimension*, J. Comput. System Sci., 53 (1996), pp. 161–170.
[38] T. J. Schaefer, *The complexity of satisfiability problems*, in Proceedings of the 10th Annual ACM Symposium on Theory of Computing, 1978, pp. 216–226.
[39] M. Y. Vardi, *Constraint satisfaction and database theory: A tutorial*, in Proceedings of the 19th Annual ACM Symposium on Principles of Database Systems (PODS 2000), 2000, pp. 76–85.
[40] M. Wand and P. M. O'Keefe, *On the complexity of type inference with coercion*, in Conference on Functional Programming Languages and Computer Architecture, 1989, pp. 293–298.

SIAM J. COMPUT.
Vol. 36, No. 1, pp. 247–278

RAPIDLY MIXING MARKOV CHAINS FOR SAMPLING CONTINGENCY TABLES WITH A CONSTANT NUMBER OF ROWS*

MARY CRYAN†, MARTIN DYER‡, LESLIE ANN GOLDBERG§, MARK JERRUM†, AND RUSSELL MARTIN¶

Abstract. We consider the problem of sampling almost uniformly from the set of contingency tables with given row and column sums, when the number of rows is a constant. Cryan and Dyer [*J. Comput. System Sci.*, 67 (2003), pp. 291–310] have recently given a *fully polynomial randomized approximation scheme* (fpras) for the related counting problem, which employs Markov chain methods indirectly. They leave open the question as to whether a natural Markov chain on such tables mixes rapidly. Here we show that the "2×2 heat-bath" Markov chain is rapidly mixing. We prove this by considering first a heat-bath chain operating on a larger window. Using techniques developed by Morris [*Random Walks in Convex Sets*, Ph.D. thesis, Department of Statistics, University of California, Berkeley, CA, 2000] and Morris and Sinclair [*SIAM J. Comput.*, 34 (2004), pp. 195–226] for the multidimensional knapsack problem, we show that this chain mixes rapidly. We then apply the comparison method of Diaconis and Saloff-Coste [*Ann. Appl. Probab.*, 3 (1993), pp. 696–730] to show that the 2×2 chain is also rapidly mixing.

Key words. contingency table, balanced almost-uniform permutation, strongly balanced permutation

AMS subject classifications. 60J20, 05B30, 68W20, 68R05

DOI. 10.1137/S0097539703434243

1. Introduction. Given two vectors of positive integers, $r = (r_1, \ldots, r_m)$ and $c = (c_1, \ldots, c_n)$, an $m \times n$ matrix $[X[i,j]]$ of nonnegative integers is a *contingency table* with row sums r and column sums c if $\sum_{j=1}^{n} X[i,j] = r_i$ for every row i and $\sum_{i=1}^{m} X[i,j] = c_j$ for every column j. We write $\Sigma_{r,c}$ to denote the set of all contingency tables with row sums r and column sums c. We assume that $\sum_{i=1}^{m} r_i = \sum_{j=1}^{n} c_j$ (since otherwise $\Sigma_{r,c} = \emptyset$) and denote by N the common total, called the *table sum*.

In this paper, we consider the problem of sampling contingency tables almost uniformly at random. No technique currently exists for polynomial-time sampling when the row and column sums can be arbitrary. In this paper we consider a particular restriction, namely, the case in which the number of rows is a constant. We focus on the Markov chain Monte Carlo (MCMC) method for sampling, which has already been successfully used to sample contingency tables, for other restrictions of the problem (see [8, 15, 2, 11]). We prove that a natural Markov chain, which we refer to as $\mathcal{M}_{2\times 2}$, is rapidly mixing when the number of rows is constant.

Before we give details of previous work on the MCMC method for sampling contingency tables, we will first discuss recent work on approximate counting of con-

*Received by the editors September 10, 2003; accepted for publication (in revised form) October 7, 2005; published electronically June 19, 2006. This work was partially supported by the EPSRC grant "Sharper Analysis of Randomised Algorithms: a Computational Approach," the EPSRC grant GR/R44560/01 "Analysing Markov-chain based random sampling algorithms," and the IST Programme of the EU under contracts IST-1999-14186 (ALCOM-FT) and IST-1999-14036 (RAND-APX). A preliminary version of this paper appeared in *Proceedings of the* 43*rd Annual IEEE Symposium on Foundations of Computer Science*, 2002, pp. 711–720.

http://www.siam.org/journals/sicomp/36-1/43424.html

†School of Informatics, University of Edinburgh, Edinburgh EH9 3JZ, UK.

‡School of Computing, University of Leeds, Leeds LS2 9JT, UK.

§Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK.

¶Department of Computer Science, University of Liverpool, Liverpool L69 3BX, UK.

tingency tables, when the number of rows is constant. Cryan and Dyer [3] recently gave a *fully polynomial randomized approximation scheme* (fpras) for approximately counting contingency tables in this setting (i.e., for approximating $|\Sigma_{r,c}|$ with given r, c). It was previously shown by Dyer, Kannan, and Mount [12] that the problem of *exact* counting is $\sharp P$-complete, even when there are only two rows. (Barvinok [1] gave a polynomial-time algorithm to exactly count contingency tables when the number of rows and the number of columns is constant.) It is well known that for all *self-reducible* problems, finding an fpras for approximate counting is equivalent to finding a *fully polynomial almost uniform sampler* (fpaus) (see Jerrum, Valiant, and Vazirani [17]). Contingency tables are not known to be self-reducible—it is true that the existence of an fpaus for almost-uniform sampling of contingency tables does imply an fpras for approximately counting contingency tables (see, for example, [11]), but the other direction is not known to hold. It is shown in [3] that the fpras does imply a sampling algorithm, though this algorithm depends on ϵ^{-1} rather than on $\log \epsilon^{-1}$. Recently Dyer [10] developed an elegant dynamic programming technique for contingency tables with a constant number of rows. He applied this technique to design two algorithms: an fpaus for uniformly sampling contingency tables with a constant number of rows and an fpras for approximately counting the number of contingency tables when the number of rows is constant. The running times of his algorithms significantly improve on the results in [3].

The algorithm in [3] is a mixture of dynamic programming and volume estimation and uses Markov chain methods only indirectly. The sampling algorithm of [10] is based on dynamic programming, and samples are generated by a probabilistic traceback through the dynamic programming table. It does not use Markov chain methods at all. Therefore the question still remains as to whether the MCMC method can be applied *directly* to this problem. In addition to its intrinsic interest, this question is important for two reasons. First, previous research in this area has routinely adopted the MCMC approach. Second, the MCMC method is more convenient, and has been more widely applied, for practical applications of sampling.

We give here the first proof of rapid mixing for a natural Markov chain when the number of rows m is a constant. This Markov chain, which we refer to as $\mathcal{M}_{2\times 2}$, was introduced by Dyer and Greenhill [11]. During a step of the chain, a 2×2 subtable is selected uniformly at random and is updated randomly. The subtable is updated according to the "heat-bath" method. In particular, a new subtable is chosen from the conditional distribution (the uniform distribution, conditioned on the configuration outside of the subtable). In order to analyze $\mathcal{M}_{2\times 2}$, we first introduce an alternative heat-bath chain, $\mathcal{M}_{\mathrm{HB}}$, which randomly updates a larger subtable. In particular, for a constant d_m which will be defined later, it updates a subtable of size $m \times (2d_m + 1)$ (also following the "heat-bath" method of selecting a new subtable chosen uniformly at random, conditioned on the configuration outside of the subtable). We use the multicommodity flow technique of Sinclair [24] to analyze the mixing time of $\mathcal{M}_{\mathrm{HB}}$. Using techniques developed by Morris and Sinclair [20] (see also [19]), we show that this chain mixes in time polynomial in the number of columns and the logarithm of the table sum. In section 5 we compare $\mathcal{M}_{\mathrm{HB}}$ to $\mathcal{M}_{2\times 2}$ and hence show that $\mathcal{M}_{2\times 2}$ is also rapidly mixing. This is the first proof that any chain converges in polynomial time even when the number of columns, as well as the number of rows, is constant. Establishing mixing in this case is one step of our proof. (See Pak [22] for an approach to this problem not using MCMC.) We note further that our results provide an alternative (and very different) fpras for counting contingency tables to that of Cryan and Dyer [3].

We now review previous work on the MCMC method for sampling contingency tables.

Contingency tables are important in applied statistics, where they are used to summarize the results of tests and surveys. The conditional volume test of Diaconis and Efron [5] is perhaps the most soundly based method for performing tests of significance in such tables. The Diaconis–Efron test provides strong motivation for the problem of efficiently choosing a contingency table with given row and column sums uniformly at random. Other applications of counting and sampling contingency tables are discussed by Diaconis and Gangolli [6]. See also Mount [21] for additional information, and De Loera and Sturmfels [4] for the current limits of exact counting methods.

With the exception of [1, 3] (and a recent result of Dyer [10]), most previous work on sampling contingency tables applies the MCMC method, as described in the survey of Jerrum and Sinclair [16]. This method, which has been used to solve many different sampling problems, is based on a very simple idea. Suppose that we have a Markov chain on a finite set of discrete structures Ω, defined by the transition matrix P. If the Markov chain is *ergodic*, then it will converge to a unique stationary distribution ϖ on Ω, regardless of the initial state. This gives a nice method for sampling from the distribution ϖ: Starting in any state, we run the Markov chain for some "sufficiently long" number of steps. Then the final state is taken as a sample. The key issue with using the MCMC method is determining how long the chain takes to converge to its stationary distribution.

The first explicit definition of Markov chains for uniformly sampling contingency tables apparently occurs in the papers of Diaconis and Gangolli [6] and Diaconis and Saloff-Coste [8], although it is mentioned in [6] that this chain had already been used by practitioners. A single step of the chain is generated as follows: An ordered pair of rows i_1, i_2 is chosen uniformly at random from all rows of the table, and an ordered pair of columns j_1, j_2 is chosen uniformly at random from all columns, giving a 2×2 submatrix. The entries of the 2×2 submatrix are modified as follows:

$$X'[i_1, j_1] = X[i_1, j_1] + 1, \quad X'[i_1, j_2] = X[i_1, j_2] - 1,$$
$$X'[i_2, j_1] = X[i_2, j_1] - 1, \quad X'[i_2, j_2] = X[i_2, j_2] + 1.$$

If modifying the matrix results in a negative value for any $X'[i, j]$, the move is not carried out. Diaconis and Gangolli proved that this Markov chain is ergodic, and the stationary distribution of the chain is uniform on $\Sigma_{r,c}$. They did not attempt to bound the mixing time of the chain, but it is clear that the mixing time is not better than pseudopolynomial in the input. That is, the mixing time is at least a polynomial in N (rather than a polynomial in $\log N$). For a discussion of pseudopolynomial time and approximation algorithms, see Chapter 8 of [26].

Later Diaconis and Saloff-Coste [8] considered the case when the numbers of rows and columns are both constant and proved that, in this case, their chain converges in time quadratic in the table sum. Hernek [15] considered the case when the table has two rows and proved that the same chain mixes in time polynomial in the number of columns and the table sum. Chung, Graham, and Yau [2] showed that a slightly modified version of the Diaconis and Saloff-Coste chain converges in time polynomial in the table sum, the number of rows, and the number of columns, provided that all row and column sums are sufficiently large.

The first truly polynomial-time algorithm (polynomial in the number of rows, the number of columns, and the logarithm of the table sum) for sampling contingency

tables was given by Dyer, Kannan, and Mount [12]. They took a different approach to the sampling problem, considering $\Sigma_{r,c}$ as the set of integer points within a convex polytope. They used an existing algorithm for sampling continuously from a convex polytope, combined with a rounding procedure, to sample integer points from inside the polytope. For any input with row sums of size $\Omega(n^2m)$ and column sums of size $\Omega(nm^2)$, their algorithm converges to the uniform distribution on $\Sigma_{r,c}$ in time polynomial in the number of rows, the number of columns, and the logarithm of the table sum. Their result was later refined by Morris [18], who showed that the result also holds when the row sums are $\Omega(n^{3/2}m\log m)$ and the column sums are $\Omega(m^{3/2}n\log n)$.

Using different techniques, Dyer and Greenhill [11] considered the problem of sampling contingency tables when the table has only two rows. They considered the 2×2 heat-bath chain $\mathcal{M}_{2\times 2}$ and showed that for two-rowed tables, the chain converges to the uniform distribution on $\Sigma_{r,c}$ in time that is polynomial in the number of columns and the logarithm of the table sum.

Our paper can properly be viewed as extending Dyer and Greenhill's results to any constant number of rows. Thus, our main result is that $\mathcal{M}_{2\times 2}$ is rapidly mixing for any constant number of rows. First, however, in section 4, we examine $\mathcal{M}_{\mathrm{HB}}$. Theorem 7 shows that this chain is rapidly mixing. Theorem 8 of section 5 bounds the mixing time of the $\mathcal{M}_{2\times 2}$ in terms of the mixing time of $\mathcal{M}_{\mathrm{HB}}$. Combining the two theorems gives the main result.

2. Definitions. First, we define the Markov chain $\mathcal{M}_{2\times 2}$. The state space is $\Sigma_{r,c}$. Given a contingency table $X \in \Sigma_{r,c}$, a move is made as follows. With probability $1/2$, the chain stays at state X. With the remaining probability, a 2×2 submatrix is chosen as follows. A pair of rows i_1, i_2 is chosen uniformly at random, and a pair of columns j_1, j_2 is chosen uniformly at random. The submatrix $X[i_1, j_1], X[i_1, j_2], X[i_2, j_1], X[i_2, j_2]$ is then replaced with a submatrix chosen uniformly at random from the set of 2×2 matrices with row sums $X[i_1, j_1] + X[i_1, j_2]$ and $X[i_2, j_1] + X[i_2, j_2]$ and column sums $X[i_1, j_1] + X[i_2, j_1]$ and $X[i_1, j_2] + X[i_2, j_2]$.

The self-loop probability $1/2$ in the definition of $\mathcal{M}_{2\times 2}$ is introduced for a technical reason—it simplifies the comparison of $\mathcal{M}_{2\times 2}$ and $\mathcal{M}_{\mathrm{HB}}$ in section 5 by ensuring that the eigenvalues of the transition matrix of $\mathcal{M}_{2\times 2}$ are not negative.

Next we define the Markov chain $\mathcal{M}_{\mathrm{HB}}$. In section 4 we will define a constant d_m (which depends on m but not on n or on the input vectors r and c). The state space is $\Sigma_{r,c}$. Given a contingency table $X \in \Sigma_{r,c}$, a move is made as follows. With probability $3/4$, the chain stays at state X. With the remaining probability, an $m \times (2d_m + 1)$ submatrix is chosen as follows. A set of $2d_m + 1$ columns $j_1, \ldots, j_{2d_m+1}$ is chosen uniformly at random from all columns of the table. The submatrix involving these columns is then replaced with a submatrix chosen uniformly at random from the set of all $m \times (2d_m + 1)$ matrices with the same row and column sums as the chosen submatrix.

The self-loop probability $3/4$ in the definition of $\mathcal{M}_{\mathrm{HB}}$ is again introduced for a technical reason—it ensures that the eigenvalues of the transition matrix of $\mathcal{M}_{\mathrm{HB}}$ are all at least $1/2$, which is useful in the comparison of $\mathcal{M}_{2\times 2}$ and $\mathcal{M}_{\mathrm{HB}}$. It is not necessary to make the self-loop probability be $3/4$—anything greater than $1/2$ suffices.

3. Background. In this section we summarize the techniques that we will use to bound the mixing time of $\mathcal{M}_{\mathrm{HB}}$. Our analysis is carried out using the multicommodity flow approach of Sinclair [24] for bounding the mixing time of a Markov chain. Sinclair's result builds on some earlier work due to Diaconis and Stroock [9].

In this section, and throughout the rest of the paper, we will use $[n]$ to denote the set $\{1, \ldots, n\}$ when n is a positive integer. We will use w^i to denote the ith component of a multidimensional weight vector w.

The setting is familiar: We have a finite set Ω of discrete structures and a transition matrix P on the state space Ω. It is assumed that the Markov chain defined by P is *ergodic*; that is, it satisfies the properties of *irreducibility* and *aperiodicity* (see Grimmett and Stirzaker [13]). It is well known that any ergodic Markov chain has a unique *stationary distribution*; that is, there is a unique distribution ϖ on Ω such that $\varpi P = \varpi$. Furthermore, for any choice of initial state $x \in \Omega$ and any state $y \in \Omega$, $P^t(x, y) \to \varpi(y)$ as $t \to \infty$ (see Chapter 6 of Grimmett and Stirzaker [13] for details). Sinclair also assumes that the Markov chain is *reversible* with respect to its stationary distribution; that is, $\varpi(x)P(x, y) = \varpi(y)P(y, x)$ for all $x, y \in \Omega$.

For any start state x, we define the *variation distance* between the stationary distribution and a walk of length t by $\mathrm{V}(\varpi, P^t(x)) = (1/2)\sum_{y\in\Omega} |\varpi(y) - P^t(x, y)|$. For any $0 < \epsilon < 1$ and any start state x, let $\tau_x(\epsilon)$ be defined as $\tau_x(\epsilon) = \min\{t : \mathrm{V}(\varpi, P^t(x)) \le \epsilon\}$. The *mixing time* of the chain is given by the function $\tau(\epsilon)$, defined as $\tau(\epsilon) = \max\{\tau_x(\epsilon) : x \in \Omega\}$.

The multicommodity flow approach is defined in terms of a graph G_Ω defined by the Markov chain. The vertices of G_Ω are the elements of Ω, and the graph contains an edge $(u \to v)$ for every pair of states such that $P(u, v) > 0$. We call this graph the *Markov kernel*. For any $x, y \in \Omega$, a *unit flow* from x to y is a set $\mathcal{P}_{x,y}$ of simple directed paths of G_Ω from x to y, such that each path $p \in \mathcal{P}_{x,y}$ has a positive weight α_p, and the sum of the α_p over $p \in \mathcal{P}_{x,y}$ is 1. A *multicommodity flow* is a family of unit flows $\mathcal{F} = \{\mathcal{P}_{x,y} : x, y \in \Omega\}$ containing a unit flow for every pair of states from Ω. The important properties of a multicommodity flow are the maximum flow passing through any edge and the maximum length of a path in the flow. We define the length $\mathcal{L}(\mathcal{F})$ of the multicommodity flow $\mathcal{F}$ by $\mathcal{L}(\mathcal{F}) = \max_{x,y} \max\{|p| : p \in \mathcal{P}_{x,y}\}$, where $|p|$ denotes the length of p. For any edge e of G_Ω, we define $\mathcal{F}(e)$ to be the sum of the α_p weights over all p such that $e \in p$ and $p \in \mathcal{P}_{x,y}$ for some $x, y \in \Omega$.

The following theorem is an amalgamation of the results of Sinclair [24]. See also the closely connected work of Diaconis and Stroock [9]. Note that all logarithms in this paper are taken to be natural logarithms.

THEOREM 1 (Sinclair [24]). *Let P be the transition matrix of an ergodic, reversible Markov chain on Ω whose stationary distribution is the uniform distribution. Suppose that the eigenvalues of P are nonnegative. Let $\mathcal{F}$ be a multicommodity flow on the graph G_Ω. Then the mixing time of the chain is bounded above by*

$$\tau(\epsilon) \le |\Omega|^{-1}\mathcal{L}(\mathcal{F}) \max_e \frac{\mathcal{F}(e)}{P(e)}(\log|\Omega| + \log\epsilon^{-1}). \tag{1}$$

Two key ingredients of our analysis of $\mathcal{M}_{\mathrm{HB}}$ in section 4 are the "balanced almost-uniform permutations" and the "strongly balanced permutations" used by Morris and Sinclair [20] for their analysis of the multidimensional knapsack problem. We will use an interleaving of a balanced almost-uniform permutation and a strongly balanced permutation to spread flow between each pair of states $x, y \in \Sigma_{r,c}$. The main idea is this: Given x and y we will use a permutation π of the columns of x to define a path of contingency tables from x to y. We will route flow from x to y along this path. Actually, π will be chosen from a distribution, and the amount of flow routed along the path corresponding to π will be proportional to the probability with which π is generated.

We will use the following notation. If π is a permutation of the n columns of a contingency table, $\pi(i)$ will denote the original column (in $\{1,\ldots,n\}$) which gets put into position i by the permutation. Thus, $\pi\{1,\ldots,k\} = \{\pi(1),\ldots,\pi(k)\}$ denotes the set of original columns that get put into the first k positions by the permutation.

DEFINITION 2 (Morris and Sinclair [20, Definition 3.2]). *Let σ be a random variable taking values in S_n (i.e., σ is a permutation of $\{1,\ldots,n\}$) and let $\lambda \in \mathbb{R}$. Then σ is a λ-uniform permutation if*

$$\Pr[\sigma\{1,\ldots,k\} = U] \le \lambda \times \binom{n}{k}^{-1}$$

for every k with $k \in [n]$ and every $U \subseteq \{1,\ldots,n\}$ of cardinality k.

DEFINITION 3 (Morris and Sinclair [20, Definition 5.1]). *Let $w_1,\ldots,w_n \in \mathbb{R}^d$ be any d-dimensional* weights *satisfying $\sum_{j=1}^n w_j = 0$ (i.e., $\sum_{j=1}^n w_j^i = 0$ for every $i \in [d]$). A permutation σ of $1,\ldots,n$ is ℓ-balanced if*

$$\left|\sum_{j=1}^{k} w_{\sigma(j)}^i\right| \le \ell M_i$$

for all $i \in [d]$ and $k \in [n]$, where $M_i = \max_{1\le j\le n} |w_j^i|$.

Checking the definition above, we see that a balanced permutation is one in which the partial sums of the weights (in each dimension) do not vary too much, where the factor ℓ gives us a bound on this variation. Morris and Sinclair showed how to construct balanced almost-uniform permutations when d is constant. (See also Theorem 3.2 in [19].)

THEOREM 4 (Morris and Sinclair [20, Theorem 5.3]). *For every positive integer d, there exist a constant g_d and a polynomial function p_d such that for any set of weights $\{w_j\}_{j=1}^n$ in $\mathbb{R}^d$, there exists a g_d-balanced, $p_d(n)$-uniform permutation.*

The key points which we should keep in mind are (1) the distribution which Morris constructs is "nearly" uniform (and has a fair amount of entropy) and (2) the permutations satisfy some sort of balance property on multidimensional weights. Roughly, one should think of these weights as corresponding to the columns of our contingency tables—the multiple dimensions come from having multiple rows. Loosely speaking, the "balance" property of these permutations will be used to construct our multicommodity flow to generate a path (or a family of paths) of contingency tables to get from X to Y, each pair of tables along this path differing by a single move of $\mathcal{M}_{\mathrm{HB}}$.

Note that the construction of the g_d-balanced, $p_d(n)$-uniform permutation of Morris and Sinclair is carried out by induction on the number of dimensions d. It is clear from the construction that g_d will be no greater than $4^{d+1} - 1$, though it is not easy to see a way of obtaining a smaller constant. We mention this because g_m will appear in the exponent of n when we bound the mixing time of our Markov chains (where m is the number of rows).

The "almost uniform" property will help ensure that the flow $\mathcal{F}(e)$ through any edge in G_Ω will not be too large (cf. Theorem 1). As mentioned before, we actually use a combination of permutations, one of which is balanced and almost-uniform, and a second type called "strongly balanced."

DEFINITION 5 (Morris and Sinclair [20, Definition 5.4]). *Let $w_1,\ldots,w_n \in \mathbb{R}^d$ be any d-dimensional weights. Define $\mu = (\mu^1,\ldots,\mu^d)$ to be the vector of means of the w_j weights ($\mu^i = (\sum_{j=1}^n w_j^i)/n$ for all i). A permutation σ of $1,\ldots,n$ is*

strongly ℓ-balanced if for all $k \in [n]$ and all $i \in [d]$, there exists a set $S \subseteq [n]$ with $|S \oplus \{1, \ldots, k\}| < \ell$ such that $(\sum_{j=1}^{k} w^i_{\sigma(j)} - k\mu^i)$ and $(\sum_{j \in S} w^i_{\sigma(j)} - k\mu^i)$ have opposite signs (or either is 0).

The main difference between ordinary balance and "strong balance" is that the definition of ordinary balance requires that the prefix sum $\sum_{j=1}^{k} w^i_{\sigma(j)}$ should be "close" to $k\mu$ for every k. However, strong balance requires that for every prefix k and every row i, it should be possible to find a small number of columns so that removing those columns changes the sign of $(\sum_{j=1}^{k} w^i_{\sigma(j)} - k\mu^i)$.

Morris and Sinclair [20] adapted a result of Steinitz [25] (see also Grinberg and Sevast'yanov [14]) to show that the following theorem holds.

THEOREM 6 (Morris and Sinclair [20, Lemma 5.5]). *For any sequence $\{w_j\}_{j=1}^{n}$ in $\mathbb{R}^d$, there exists a strongly $16d^2$-balanced permutation.*

4. Analysis of the generalized chain. We fix $r = (r_1, \ldots, r_m)$, the list of row sums, and $c = (c_1, \ldots, c_n)$, the list of column sums, and let Ω be the state space $\Sigma_{r,c}$ of $m \times n$ contingency tables with these row and column sums. Recall that N denotes the table sum $\sum_{i=1}^{m} r_i$.

Recall that g_m is the constant of Theorem 4 for balanced almost-uniform permutations for columns of dimension m. Let $d_m = 2m(3g_m + 1) + 1 + 34m^3$. We use P_{HB} for the transition matrix of the Markov chain $\mathcal{M}_{\mathrm{HB}}$ which was defined in section 2.

In this section, our goal is to prove the following theorem.

THEOREM 7. *The mixing time τ_{HB} of $\mathcal{M}_{\mathrm{HB}}$ is bounded from above by a polynomial in n, $\log N$, and $\log \epsilon^{-1}$.*

In order to prove Theorem 7, we will show how to define a multicommodity flow $\mathcal{F}$ such that the total flow along any transition (ω, ω'') is at most $8fn^{2d_m+1}P_{\mathrm{HB}}(\omega, \omega'')$, where f is an expression that is at most $poly(n)\,|\Omega|$. We will also ensure that $\mathcal{L}(\mathcal{F})$ is bounded from above by a polynomial in n. Theorem 7 will then follow from (1) in Theorem 1. Constructing the flow $\mathcal{F}$ is done in a two-stage process. In subsection 4.1, we will define a multicommodity flow $\mathcal{F}^*$. Then in subsection 4.2, we will first prove that the total flow through any state ω is at most f. Finally, also in subsection 4.2, we will construct $\mathcal{F}$ by modifying $\mathcal{F}^*$.

In the first subsection we define the multicommodity flow we use in our application of Theorem 1. The construction uses the balanced, strongly balanced, and almost-uniform properties of permutations of (some of) the columns of the contingency tables.

The second subsection shows that, with the multicommodity flow we define, each edge of the graph G_Ω is not too congested, and each path length is small. Theorem 7 then follows from Theorem 1.

4.1. Defining the flow. The construction of $\mathcal{F}$ in this subsection uses the methods of Morris and Sinclair [20] introduced in section 3.

Let k be the index of the largest column sum c_k. Let X and Y be contingency tables in Ω. Let X_j denote the jth column of X. We show how to route a unit of flow from X to Y.

The rough idea is as follows. We first define the notion of a *column constrained table*, which is an $m \times n$ matrix that has the correct column sums for $\Sigma_{r,c}$ but may violate the row sum constraints. We will choose a permutation π from an appropriate distribution. The distribution from which π is chosen will be defined in terms of an interlacing of the random balanced permutation of Theorem 4 and the strongly balanced permutation of Theorem 6. π will be a permutation of most of the columns

of the table. The permutation π will define a path

$$Z_0 = X, \ldots, Z_{n'}$$

(for some $n' < n$) of column constrained tables, where each table Z_h contains the column Y_j for $j \in \pi\{1, \ldots, h\}$ and the column X_j for all other j (so at each point, we swap another column of X for the same column of Y). In subsection 4.1.1 we show that the balance properties of π ensure that for any Z_h, we can bring all the row sums below r_i by deleting a constant number of columns. Then in subsection 4.1.2 we will show how to use this fact to define a path

$$X = Z'_0, \ldots, Z'_{n'+1} = Y,$$

where each Z'_h is in $\Sigma_{r,c}$ and there is a transition in $\mathcal{M}_{\mathrm{HB}}$ from each Z'_h to Z'_{h+1}. The amount of flow that we will route along this path will be proportional to the probability with which π is chosen.

4.1.1. A first step toward building paths. We start building our path(s) from X to Y by first defining a path of column constrained tables $Z_0 = X, \ldots, Z_{n'}$ using an interlacing of the random balanced permutation of Theorem 4 and the strongly balanced permutation of Theorem 6. In subsection 4.1.2, we will show how to modify these columns, in a specific manner, to yield a new path of tables such that (i) the new tables are contingency tables (i.e., they satisfy the row sums as well as the column sums) and (ii) each successive pair along this path differs by a transition of $\mathcal{M}_{\mathrm{HB}}$.

Let R_i^X be the set of indices for the $3g_m + 1$ largest entries of row i of X.

Let R_i^Y be the set of indices for the $3g_m + 1$ largest entries of row i of Y.

Let $R = \left(\cup_i R_i^X\right) \cup \left(\cup_i R_i^Y\right) \cup \{k\}$ be the union of all the R_i^X and R_i^Y sets, together with the index k (which was defined earlier to be the index of the largest column sum).

The cardinality of R is at most $2m(3g_m + 1) + 1$.

The columns in R are "reserved" columns that we identify before permuting the columns. We do not permute these columns—we need them for something else. For every row i, define

$$M_i = \min\{\max\{X[i,j] : j \notin R\}, \max\{Y[i,j] : j \notin R\}\},$$
$$L_i = \{j : j \notin R, X[i,j] > M_i\} \cup \{j : j \notin R, Y[i,j] > M_i\}.$$

Note that by definition of M_i, we either have $\{j : j \notin R,\ X[i,j] > M_i\} = \emptyset$ or $\{j : j \notin R, Y[i,j] > M_i\} = \emptyset$, so each L_i corresponds to a set of columns of X *or* a set of columns of Y, but not both. Also from their definitions, we see that $M_i \le X[i,j]$ for all $j \in R_i^X$, and $M_i \le Y[i,j]$ for all $j \in R_i^Y$. Set $L = \cup_{i=1}^m L_i$ and $S = [n] - (L \cup R)$.

For every column $j \in [n] - R$, define the m-dimensional weight $w_j = Y_j - X_j$. Let μ be the m-dimensional vector representing the mean of the $w_{j \in [n]-R}$. Note that

$$\mu^i = \frac{\sum_{j \in [n]-R} Y[i,j] - X[i,j]}{n - |R|} = \frac{\sum_{j \in R} X[i,j] - Y[i,j]}{n - |R|}.$$

Let π_1 be a strongly $16m^2$-balanced permutation on the set of weights $\{w_j\}_{j \in L}$. This exists by Theorem 6. Let π_2 be a g_m-balanced $p_m(|S|)$-uniform permutation on $\{w_j\}_{j \in S}$. This exists by Theorem 4. π_2 is a random permutation. We interlace π_1 and π_2 in the same way as Morris and Sinclair [20] do to get a permutation π on $\{w_j\}_{j \in [n]-R}$. For the benefit of the reader, we restate the rule for performing

this interlacing: Suppose that $\pi(1), \pi(2), \ldots, \pi(h)$ have already been assigned and that $\pi\{1, 2, \ldots, h\} = \pi_1\{1, \ldots, h_1\} \cup \pi_2\{1, \ldots, h_2\}$. Then either $\frac{h_1}{h} \leq \frac{|L|}{|L|+|S|}$ or $\frac{h_2}{h} < \frac{|S|}{|L|+|S|}$. We define $\pi(h+1)$ by

$$\pi(h+1) = \begin{cases} \pi_1(h_1+1) & \text{if } \frac{h_1}{h} \leq \frac{|L|}{|L|+|S|}, \\ \pi_2(h_2+1) & \text{if } \frac{h_2}{h} < \frac{|S|}{|L|+|S|}. \end{cases}$$

This new permutation π satisfies inequalities (5.8) and (5.9) in Morris and Sinclair [20], reproduced as inequalities (2) and (3) below and proved in [20]. (See also inequalities (3.8) and (3.9) in [19].) These inequalities state that for every prefix h (h is the index of a column) and every dimension i (i is the index of a row), there exist sets of column indices $V_{i,h}$ and $W_{i,h}$ such that $V_{i,h}$ differs from $\{1, \ldots, h\}$ by at most $17m^2$ indices and $W_{i,h}$ differs from $\{1, \ldots, h\}$ by at most $17m^2$ indices, and

$$\sum_{j \in V_{i,h}} w^i_{\pi(j)} \leq (h-1)\mu^i + 3g_m M_i \quad \text{for every } i = 1, \ldots, m, \tag{2}$$

$$\sum_{j \in W_{i,h}} w^i_{\pi(j)} \geq (h-1)\mu^i - 3g_m M_i \quad \text{for every } i = 1, \ldots, m. \tag{3}$$

The sets $V_{i,h}$ and $W_{i,h}$ will play an important role for us later.

Now let $n' = n - |R|$. For the permutation π constructed above we define the path of tables $X = Z_0, Z_1, \ldots, Z_{n'}$ as follows: For every h, Z_h contains the columns X_j for $j \in R \cup \pi\{h+1, \ldots, n'\}$ and columns Y_j for $j \in \pi\{1, \ldots, h\}$. We see that $Z_{n'}$ differs from Y by at most $2m(3g_m + 1) + 1$ columns.

It is important to note that $Z_0, \ldots, Z_{n'}$ may not be contingency tables in $\Sigma_{r,c}$ since they need not satisfy the row constraints. Thus, we cannot use this path directly to define our flow from X to Y. Nevertheless, we may base our path on these tables. The important point is that π and, in particular, (2) and (3) will allow us to turn $Z_0, \ldots, Z_{n'}$ into a path of contingency tables. In the remainder of this subsection, we will show that we do not have to change too many columns to turn Z_h into a contingency table. Subsequently in subsection 4.1.2 we will show that if we are careful about how we map Z_h to a contingency table, the resulting collection of paths will have good congestion.

We introduce the notation $(J(X), J(Y))$ to denote a set containing columns from X and from Y: For sets of indices $J(X) \subseteq [n]$ and $J(Y) \subseteq [n]$, $(J(X), J(Y))$ contains the column X_j for each $j \in J(X)$ and the column Y_j for each $j \in J(Y)$. For any set of columns $(J(X), J(Y))$, we represent the "row sum" for row i by $row^i(J(X), J(Y))$, which has the value $\sum_{j \in J(X)} X[i,j] + \sum_{j \in J(Y)} Y[i,j]$.

Defining $J_h(X) = R \cup \pi\{h+1, \ldots, n'\}$ and $J_h(Y) = \pi\{1, \ldots, h\}$, we see that Z_h is the set of columns $(J_h(X), J_h(Y))$. Each of the Z_h tables is a column constrained table because, as previously noted, it is possible that some rows i may have $row^i(J_h(X), J_h(Y)) \neq r_i$. However, all the column sums are satisfied by Z_h.

Step 1. We show we can modify Z_h by "deleting" at most d_m columns (including all of the X_j columns for $j \in R$) to bring the row sum for every row i below $r_i(1-1/n)$. We also show a dual result—if we "add" at most d_m columns to Z_h this brings the row sum for every row i above $r_i(1+1/n)$. The "adding" causes some column indices to appear in both $J(X)$ and $J(Y)$; thus the resulting configuration isn't much like a contingency table, but the construction will be useful below. Let $V_{i,h}$ and $W_{i,h}$ be the sets of inequalities (2) and (3).

First, instead of considering Z_h, consider $(\pi([n'] - V_{i,h}), \pi(V_{i,h}))$. This set of columns is the result of starting with the contingency table X, removing X_j for every reserved column $j \in R$, and then adding the weights w_j for $j \in \pi(V_{i,h})$. By (2), we know that $|V_{i,h} \oplus \{1, \ldots, h\}| \leq 17m^2$ and

$$\begin{aligned} row^i(\pi([n'] - V_{i,h}), \pi(V_{i,h})) &\leq \left(r_i - \sum_{j \in R} X[i,j]\right) + (h-1)\mu^i + 3g_m M_i \\ &= \left(r_i - \sum_{j \in R} X[i,j]\right) + 3g_m M_i \\ &\quad + \frac{h-1}{n'}\left(\sum_{j \in R} X[i,j] - Y[i,j]\right) \\ &= r_i - \frac{n'-h+1}{n'}\left(\sum_{j \in R} X[i,j]\right) \\ &\quad - \frac{h-1}{n'}\left(\sum_{j \in R} Y[i,j]\right) + 3g_m M_i \\ &\leq r_i\Big(1 - \frac{1}{n}\Big), \qquad \text{(a)} \end{aligned}$$

where the last step follows because (i) we know that $M_i \leq X[i,j]$ for every $j \in R_i^X$, and $R_i^X \subseteq R$, $|R_i^X| = 3g_m + 1$. Also $\max_{j \in R_i^X} X[i,j] \geq r_i/n$. Thus $3g_m M_i + r_i/n \leq \sum_{j \in R} X[i,j]$; (ii) similarly, $3g_m M_i + r_i/n \leq \sum_{j \in R} Y[i,j]$; and (iii) the convex combination $((n'-h+1)\sum_{j \in R} X[i,j] + (h-1)\sum_{j \in R} Y[i,j])/n'$ is at least $3g_m M_i + r_i/n$.

Now suppose we also "delete" $\pi\{1, \ldots, h\} \oplus \pi\{V_{i,h}\}$ from $(\pi([n'] - V_{i,h}), \pi(V_{i,h}))$. Let $B_{i,h} = R \cup (\pi\{1, \ldots, h\} \oplus \pi\{V_{i,h}\})$.

Then

$$\begin{aligned} J_h(X) - B_{i,h} &= \pi\{h+1, \ldots, n'\} \cap \pi([n'] - V_{i,h}), \\ J_h(Y) - B_{i,h} &= \pi\{1, \ldots, h\} \cap \pi(V_{i,h}). \end{aligned}$$

By (a), we have $row^i(J_h(X) - B_{i,h}, J_h(Y) - B_{i,h}) \leq row^i(\pi([n'] - V_{i,h}), \pi(V_{i,h})) \leq r_i(1 - 1/n)$. Also, $|B_{i,h}| \leq 2m(3g_m + 1) + 1 + 17m^2$.

For the dual result, consider $(\pi([n'] - W_{i,h}) \cup R, \pi(W_{i,h}) \cup R)$. This set of columns is the result of starting with the contingency table X, adding the Y_j columns for $j \in R$, and then adding the weights w_j for $j \in \pi(W_{i,h})$. By (3), we know that $|W_{i,h} \oplus \{1, \ldots, h\}| \leq 17m^2$ and

$$\begin{aligned} row^i(\pi([n'] - W_{i,h}) \cup R, \pi(W_{i,h}) \cup R) &\geq \left(r_i + \sum_{j \in R} Y[i,j]\right) + (h-1)\mu^i - 3g_m M_i \\ &= \left(r_i + \sum_{j \in R} Y[i,j]\right) - 3g_m M_i \\ &\quad + \frac{h-1}{n'}\left(\sum_{j \in R} X[i,j] - Y[i,j]\right) \end{aligned}$$

$$= r_i + \frac{n' - h + 1}{n'}\left(\sum_{j \in R} Y[i,j]\right) + \frac{h-1}{n'}\left(\sum_{j \in R} X[i,j]\right) - 3g_m M_i$$
$$\geq r_i\Big(1 + \frac{1}{n}\Big), \qquad \text{(b)}$$

where the last step follows using (i) $3g_m M_i + r_i/n \leq \sum_{j \in R} X[i,j]$; (ii) $3g_m M_i + r_i/n \leq \sum_{j \in R} Y[i,j]$; and (iii) the convex combination property mentioned on the previous page.

Suppose we add $\pi\{1,\ldots,h\} \oplus \pi\{W_{i,h}\}$ to $(\pi([n'] - W_{i,h}) \cup R, \pi(W_{i,h}) \cup R)$.

Let $C_{i,h} = R \cup (\pi\{1,\ldots,h\} \oplus \pi\{W_{i,h}\})$.

Then

$$J_h(X) \cup C_{i,h} = \pi\{h+1,\ldots,n'\} \cup \pi([n'] - W_{i,h}) \cup R,$$
$$J_h(Y) \cup C_{i,h} = \pi\{1,\ldots,h\} \cup \pi(W_{i,h}) \cup R.$$

By (b), we have $row^i(J_h(X) \cup C_{i,h}, J_h(Y) \cup C_{i,h}) \geq row^i(\pi([n'] - W_{i,h}) \cup R, \pi(W_{i,h}) \cup R) \geq r_i(1 + 1/n)$. Also, $|C_{i,h}| \leq 2m(3g_m + 1) + 1 + 17m^2$.

Finally, define D_h to be a set of d_m column indices, including any column that is in $(\cup_i B_{i,h}) \cup (\cup_i C_{i,h})$. This is possible because $d_m = 2m(3g_m + 1) + 1 + 34m^3$. In addition to the set R (of size $2m(3g_m + 1) + 1$), each set $B_{i,h}$ contains up to $17m^2$ indices as does each set $C_{i,h}$.

Consider Z_h with all of the columns in D_h "deleted." This is the table

$$Z_h^* =_{def} (J_h(X) - D_h, J_h(Y) - D_h).$$

Since $B_{i,h} \subseteq D_h$ for every i, we have $row^i(J_h(X) - D_h, J_h(Y) - D_h) \leq r_i(1 - 1/n)$ for all i. Also define

$$\bar{Z}_h^* =_{def} (J_h(X) \cup D_h, J_h(Y) \cup D_h).$$

Since $C_{i,h} \subseteq D_h$ for every i, we have $row^i(J_h(X) \cup D_h, J_h(Y) \cup D_h) \geq r_i(1 + 1/n)$ for all i.

Note that D_h contains all of R, including the index k.

4.1.2. Going from Z_h to an element of Ω. Having defined the set D_h, we now show how to change Z_h into a contingency table. This is a crucial step, actually turning the tables $Z_0, \ldots, Z_{n'}$ into a path of elements of Ω that joins the two contingency tables X and Y. The most critical part in the construction of this multicommodity flow is to ensure that the flow on any edge $e \in G_\Omega$ is not too large. Therefore it will be important to prove that we do not map too many of the column constrained tables Z_h to any given element of Ω. We will see below that we will need to be careful in mapping column constrained tables which have large column sums for some of the "deleted" columns D_h.

Step 2. We now show how to convert Z_h into an element of Ω. We focus on the "deleted" columns D_h and show that by changing only the entries of the columns in D_h, we can obtain a contingency table $Z'_h \in \Sigma_{r,c}$. We also show a dual result: that if we define $\bar{Z}_h$ to be the set of columns which contains X_j for every Y_j column in Z_h

and contains Y_j for every X_j column in Z_h, we can show the same result for $\bar{Z}_h$ (we can construct a $\bar{Z}'_h$ in $\Sigma_{r,c}$ by changing at most d_m columns).

First let $\widehat{r}_i = row^i(J_h(X) - D_h, J_h(Y) - D_h)$, the partial row sum for row i of Z_h with the D_h columns removed. Define $s_i = r_i - \widehat{r}_i$ for all i, the sum for row i of the subtable that was removed from Z_h.

Note that $s_i \geq r_i/n$ for all i.

Let $N_h = \sum_{i=1}^{m} s_i = \sum_{j \in D_h} c_j$, by construction. We have two cases to consider.

Case 1. First suppose $N_h < 2(md_m)^2$. It is well known that whenever the total of the row sums equals the total of the column sums, there is at least one contingency table satisfying these row and column sums (see Diaconis and Gangolli [6]). For this case we choose *any* set of modified values $Z'_h[i,j]$ for $j \in D_h$ such that $\sum_{i=1}^{m} Z'_h[i,j] = c_j$ for all $j \in D_h$ and $\sum_{j \in D_h} Z'_h[i,j] = s_i$ for all $1 \leq i \leq m$. Note that because $N_h < 2(md_m)^2$ we have $s_i < 2(md_m)^2$ for all i and therefore $r_i < 2n(md_m)^2$ for all i.

Case 2. Alternatively, assume that $N_h \geq 2(md_m)^2$. As above, we are guaranteed that there is some set of $Z'_h[i,j]$ values for $j \in D_h$ that satisfy the row and column sums. But, for this case, we need something stronger—we show how we can modify the values of $Z_h[i,j]$ for the $j \in D_h$ columns in a *structured way* to obtain a subtable Z'_h satisfying the induced row sums s_i and the column sums c_j. Performing the modification in this carefully structured way allows us to ensure that the congestion on edges in G_Ω in our multicommodity flow is not too large, a point we return to following the definitions in the next paragraph.

By our previous definition of D_h, we already know that k is the index of the largest column sum c_j for $j \in D_h$. Let ℓ be the index of the biggest s_i value. For every $i \neq \ell$ and every $j \in D_h - \{k\}$, we define $a_{i,j}$ in terms of the overall row sums and the column sums:

$$a_{i,j} =_{def} \lfloor \min\{r_i, c_j\}/n(d_m)^2 \rfloor.$$

Since $Z_h[i,j]$ is either $X[i,j]$ or $Y[i,j]$, we know $Z_h[i,j] \leq \min\{r_i, c_j\}$. Therefore, for every $i \neq \ell$ and every $j \in D_h - \{k\}$, we can write

$$Z_h[i,j] = Q[i,j](a_{i,j} + 1) + R[i,j]$$

for nonnegative integers $Q[i,j], R[i,j]$, where $Q[i,j] < n(d_m)^2$ and $0 \leq R[i,j] \leq a_{i,j}$. $Q[i,j]$ and $R[i,j]$ are uniquely determined by $Z_h[i,j]$. We will show that by changing only the values of the $Q[i,j]$ to new values $Q'[i,j]$, we can obtain a subtable Z'_h satisfying the row sums s and the column sums. As promised, however, we first give the reason why we want to view the entries of Z_h in this way, focusing on the importance of changing only the $Q[i,j]$ terms to obtain the subtable Z'_h.

Our analysis of the flow we construct will rely on bounding the number of column constrained tables Z_h that can be transformed into the contingency table Z'_h. In particular, for each particular choice of D_h, we bound the number of tables Z_h that can be transformed into Z'_h. If the subtable sum N_h is less than $2(md_m)^2$, then $r_i < 2n(md_m)^2$ for every i and therefore, for every $j \in D_h$ and every row i, there are at most $2n(md_m)^2$ possible original values for $Z_h[i,j]$. On the other hand, when $N_h \geq 2(md_m)^2$, we do not have an upper bound on the original entries of the subtable. However, the method above for constructing the $Z'_h[i,j]$ values using $a_{i,j}$ gives an indirect upper bound for the number of Z_h that could be converted to Z'_h. For any true contingency table Z'_h and any selection of D_h columns which gives $N_h \geq 2(md_m)^2$, we can calculate $a_{i,j}$ for all $i \neq \ell$ and all $j \in D_h - \{k\}$. Then, for every row $i \neq \ell$

and every column j in $D_h - \{k\}$, we can find the original value of $R[i,j]$ using that $R[i,j] = Z'_h[i,j] \bmod (a_{i,j}+1)$. By the definition of $a_{i,j}$, there are at most $n(d_m)^2$ possible values for the original $Q[i,j]$, so there are only $(n(d_m)^2)^{(m-1)(d_m-1)}$ different ways of filling in the entire subtable on the $D_h - \{k\}$ columns. The fact that there are at most $poly(n)$ possible values for the original $Q[i,j]$ cells is the key that makes our analysis work in section 4.2.

We now return to our proof, since we still need to show that we can choose values $Q'[i,j]$ which ensure that the new $Z'_h[i,j]$ values will satisfy row sums s_i and column sums c_j. Recall our assumption that $N_h \geq 2(md_m)^2$. It is well known (see Dyer, Kanaan, and Mount [12]) that the row and column sums are satisfied by any integer matrix Z'_h which has $Z'_h[i,j] \geq 0$ for all i,j, where the $Z'[i,j]$ also satisfy the following inequalities:

$$\sum_{i \neq \ell} Z'_h[i,j] \leq c_j \quad \text{for every } j \in D_h - \{k\}, \tag{4}$$

$$\sum_{j \in D_h - \{k\}} Z'_h[i,j] \leq s_i \quad \text{for every } i \neq \ell, \tag{5}$$

$$\sum_{i \neq \ell} \sum_{j \in D_h - \{k\}} Z'_h[i,j] \geq N_h - s_\ell - c_k. \tag{6}$$

Now define $Q'[i,j]$ in terms of the induced row sums and the original column sums:

$$Q'[i,j] =_{def} \lfloor s_i c_j / N_h (a_{ij}+1) \rfloor$$

for all $i \neq \ell$ and all $j \in D_h - \{k\}$. For those values of i and j, we set $Z'_h[i,j] = Q'[i,j](a_{i,j}+1) + R[i,j]$. Note that $Q'[i,j] \geq 0$ for all i,j. We now prove that inequalities (4), (5), and (6) are satisfied for these $Z'[i,j]$.

Inequality (4):

$$\begin{aligned}
\sum_{i \neq \ell} Z'_h[i,j] &= \sum_{i \neq \ell} Q'[i,j](a_{i,j}+1) + \sum_{i \neq \ell} R[i,j] \\
&= \sum_{i \neq \ell} \lfloor s_i c_j / N_h (a_{ij}+1) \rfloor (a_{i,j}+1) + \sum_{i \neq \ell} R[i,j] \\
&\leq \sum_{i \neq \ell} s_i c_j / N_h + \sum_{i \neq \ell} R[i,j] \\
&\leq \sum_{i \neq \ell} s_i c_j / N_h + \sum_{i \neq \ell} a_{i,j} \\
&= c_j (1 - s_\ell / N_h) + \sum_{i \neq \ell} \lfloor \min\{r_i, c_j\} / n(d_m)^2 \rfloor \\
&\leq c_j (1 - s_\ell / N_h) + c_j (m-1) / n(d_m)^2 \\
&\leq c_j (1 - 1/m + 1/nd_m) \\
&\leq c_j.
\end{aligned}$$

The last two steps use the facts that (i) $s_\ell / N_h \geq 1/m$ (because s_ℓ is the largest row sum) and (ii) $d_m > m$.

Inequality (5): The early steps are similar to the proof for inequality (4). We get

$$\begin{aligned}\sum_{j\in D_h-\{k\}} Z'_h[i,j] &\le s_i(1-c_k/N_h) + \sum_{j\in D_h-\{k\}} \lfloor \min\{r_i,c_j\}/n(d_m)^2 \rfloor \\ &\le s_i(1-c_k/N_h) + (d_m-1)r_i/n(d_m)^2 \\ &\le s_i(1-1/d_m) + s_i/d_m \\ &\le s_i,\end{aligned}$$

where the second-to-last step uses the facts that (i) $c_k/N_h \ge 1/d_m$ (because c_k is the largest column sum in the subtable) and (ii) $s_i \ge r_i/n$.

Inequality (6):

$$\begin{aligned}\sum_{i\neq\ell}\sum_{j\in D_h-\{k\}} Z'_h[i,j] &= \sum_{i\neq\ell}\sum_{j\in D_h-\{k\}} Q'[i,j](a_{i,j}+1) + \sum_{i\neq\ell}\sum_{j\in D_h-\{k\}} R[i,j] \\ &= \sum_{i\neq\ell}\sum_{j\in D_h-\{k\}} \lfloor s_i c_j/N_h(a_{i,j}+1)\rfloor (a_{i,j}+1) + \sum_{i\neq\ell}\sum_{j\in D_h-\{k\}} R[i,j] \\ &\ge \sum_{i\neq\ell}\sum_{j\in D_h-\{k\}} s_i c_j/N_h - \sum_{i\neq\ell}\sum_{j\in D_h-\{k\}} (a_{i,j}+1) + \sum_{i\neq\ell}\sum_{j\in D_h-\{k\}} 0 \\ &= (N_h-s_\ell)(N_h-c_k)/N_h - \sum_{i\neq\ell}\sum_{j\in D_h-\{k\}} (a_{i,j}+1) \\ &= (N_h-s_\ell-c_k) + s_\ell c_k/N_h - \sum_{i\neq\ell}\sum_{j\in D_h-\{k\}} (a_{i,j}+1).\end{aligned}$$

Next, note that

$$\begin{aligned} s_\ell c_k/N_h &\ge N_h/(md_m), \\ \sum_{i\neq\ell}\sum_{j\in D_h-\{k\}} (a_{i,j}+1) &\le md_m + m(N_h-c_k)/n(d_m)^2 \le md_m + N_h/(nd_m m),\end{aligned}$$

by definition of the $a_{i,j}$ and because $d_m \ge m^2$. Therefore, to show that inequality (6) holds, it suffices to show

$$N_h/(md_m) \ge md_m + N_h/(nd_m m). \tag{7}$$

We note the following statements:

$$\begin{aligned} & N_h/(md_m) \ge md_m + N_h/(nd_m m) \\ \iff & N_h(1-1/n)/(md_m) \ge md_m \\ \Longleftarrow\ & N_h/2 \ge (md_m)^2 \quad \text{(using that } n\ge 2\text{)} \\ \iff & N_h \ge 2(md_m)^2.\end{aligned}$$

This last statement holds by our previous assumption on N_h; therefore inequality (7) holds, and hence (6) is true, as required.

Therefore, in parallel with our path of column constrained tables, we have a path $X = Z'_0, \ldots, Z'_h, \ldots, Z'_{n'}$ such that Z'_h differs from Z_h in only d_m columns and $Z'_0, Z'_1, \ldots$ are true contingency tables. We can add a final step to change $Z'_{n'}$ (using one step of the Markov chain) into Y. The amount of flow from X to Y that is routed

along this path is proportional to the probability that π is chosen. Since $n' \le n$, we see that the length of this path from X to Y is at most $n+1$.

Remember that the column constrained table Z_h contains exactly n columns and that $X_j \in Z_h \Leftrightarrow Y_j \notin Z_h$. Also, Z_h is the pair $(J_h(X), J_h(Y))$. Then if we define $\bar{Z}_h$ by the pair of sets $(J_h(Y), J_h(X))$, $\bar{Z}_h$ also contains exactly n columns and $X_j \in \bar{Z}_h \Leftrightarrow X_j \notin Z_h$. Now to calculate the row sums of $\bar{Z}_h$, consider $\bar{Z}_h$ with the columns of D_h removed. Then

$$\begin{aligned} row^i(J_h(Y) - D_h, J_h(X) - D_h) &= row^i(([n] - J_h(X)) - D_h, ([n] - J_h(Y)) - D_h) \\ &= row^i([n] - (J_h(X) \cup D_h), [n] - (J_h(Y) \cup D_h)) \\ &= 2r_i - row^i(J_h(X) \cup D_h, J_h(Y) \cup D_h) \\ &\le r_i(1 - 1/n) \end{aligned}$$

because we proved that $row^i(J_h(X) \cup D_h, J_h(Y) \cup D_h) \ge r_i(1 + 1/n)$ in Step 1.

Therefore, for $\bar{Z}_h$, there also exists a set of d_m columns (the same set of indices D_h) such that we can obtain a true contingency table $\bar{Z}'_h \in \Sigma_{r,c}$ by modifying these columns in the structured way described above.

4.2. Analysis of the multicommodity flow. Next we show that the flow through any state $Z' \in \Omega$ is at most $poly(n)\,|\Omega|$. (Remark: We actually bound all of the flow *except* that due to pairs (X, Y) where $Z' = Y$. The total flow for all such pairs is only $|\Omega|$. Hence showing a bound of the form $poly(n)\,|\Omega|$ for the flow between remaining pairs where $Z' \ne Y$ provides us a similar bound for the total flow through Z'.) We assume Z' occurs as Z'_h for some pairs of contingency tables (X, Y) and some values of h and $\pi\{1, \ldots, h\}$. Again, we write $\bar{Z}_h$ for the column constrained table with X_j for $j \in J_h(Y)$ and Y_j for $j \in J_h(X)$. We claim that if we are given Z'_h and

(1) $\bar{Z}'_h \in \Omega$, a true contingency table obtained by changing d_m of the columns of $\bar{Z}_h$;
(2) the value of h and the set $\pi\{1, \ldots, h\}$ of columns already changed from X to Y in Z_h;
(3) the set D_h of d_m columns of Z_h which were modified to obtain Z'_h (note that the columns of $\bar{Z}_h$ that were modified are also the columns in D_h);
(4) the index ℓ of the largest induced row sum s_ℓ for the subtable of Z_h on D_h and the index ℓ' of the largest induced row sum for the subtable of $\bar{Z}_h$ on D_h;
(5) two possibilities:
(i) If $N_h < 2(md_m)^2$, then we are given the original values of $Z_h[i, j]$ for all i, for all $j \in D_h$. Note we have $Z_h[i, j] \le 2(md_m)^2$;
(ii) Otherwise if $N_h \ge 2(md_m)^2$, we are given the original integers $Q[i, j]$ obtained from $Z_h[i, j]$ for every $i \ne \ell$ and every $j \in D_h - \{k\}$. Note we have $0 \le Q[i, j] < n(d_m)^2$;
(5′) two possibilities:
(i) If $N_h < 2(md_m)^2$, then we are given the original values of $\bar{Z}_h[i, j]$ for all i, for all $j \in D_h$. Note we have $\bar{Z}_h[i, j] \le 2(md_m)^2$;
(ii) Otherwise if $N_h \ge 2(md_m)^2$, we are given the original integers $\bar{Q}[i, j]$ obtained from $\bar{Z}_h[i, j]$ for every $i \ne \ell'$ and every $j \in D_h - \{k\}$. Note we have $0 \le \bar{Q}[i, j] < n(d_m)^2$;

then we can construct X and Y. We refer to the information in (1)–(5′) as an "encoding" (for X and Y at the element $Z' \in \Omega$).

First we concentrate on recovering Z_h. From (3) we know the submatrix D_h which has to be modified to obtain Z_h from Z'_h. Since we know the D_h columns, we can calculate N_h. If $N_h < 2(md_m)^2$, then the information in (5) tells us the original $Z_h[i,j]$ values for $j \in D_h$ and this gives us the entire column constrained table Z_h. If $N_h \geq 2(md_m)^2$, then we use (4) to identify ℓ. For every $i \neq \ell$ and $j \in D_h - \{k\}$ we calculate $a_{i,j}$ from r_i and c_j and we calculate $R[i,j] = Z'_h[i,j] \bmod (a_{i,j}+1)$. Finally, by (5) we have the original values of $Q[i,j]$ for all $i \neq \ell$ and all $j \in D_h - \{k\}$. Therefore we calculate $Z_h[i,j] = Q[i,j](a_{i,j}+1) + R[i,j]$. We can also calculate $Z_h[\ell,j]$ for $j \in D_h - \{k\}$ by subtracting the other values of $Z_h[i,j]$ (which we just calculated) from c_j. We still need to find the values $Z_h[i,k]$, but we defer this for the moment.

In a similar way, we use (3), (4), and (5′) to obtain all columns except column k of $\bar{Z}_h$ from $\bar{Z}'_h$ (given to us in (1)).

We now have all the columns except for column k of Z_h and $\bar{Z}_h$. Z_h contains column X_j for every $j \in J_h(X) = R \cup \pi\{h+1,\ldots,n'\}$ and contains column Y_j for every $j \in J_h(Y) = \pi\{1,\ldots,h\}$. $\bar{Z}_h$ contains column X_j for every $j \in J_h(Y) = [n] - J_h(X)$ and column Y_j for every $j \in J_h(X) = [n] - J_h(Y)$. By (2), we are given $\pi\{1,\ldots,h\}$. Then the contingency table X is the set of columns where

$$X_j = \begin{cases} \text{column } j \text{ of } Z_h \text{ for } j \in [n] - \pi\{1,\ldots,h\}, \\ \text{column } j \text{ of } \bar{Z}_h \text{ for } j \in \pi\{1,\ldots,h\}. \end{cases}$$

The contingency table Y is the set of columns where

$$Y_j = \begin{cases} \text{column } j \text{ of } Z_h \text{ for } j \in \pi\{1,\ldots,h\}, \\ \text{column } j \text{ of } \bar{Z}_h \text{ for } j \in [n] - \pi\{1,\ldots,h\}. \end{cases}$$

Thus for any $Z'_h \in \Omega$, we can construct all columns of X and Y except column k. Column k of each of these can then be recovered using the original r_i values. Thus, for any $Z'_h \in \Omega$, we can construct X and Y for any pair (X,Y) whose path passes through Z'_h, given the encoding (1)–(5′).

Now we bound the flow through our given Z'_h. Note that the flow through Z'_h is (bounded by) the number of possible choices for (1) and for (3)–(5′), times the amount of flow given by all possible choices for $\pi\{1,\ldots,h\}$ (given by (2)).

Since $\bar{Z}'_h \in \Omega$, there are $|\Omega|$ choices for (1).

There are $\binom{n}{d_m-1}$ possible D_h sets (choices for (3)), since D_h always contains k.

There are m^2 choices for (4).

Depending on the value of N_h, either there are at most $n(d_m)^2$ possible values for the original value of $Z_h[i,j]$ for $j \in D_h$ (obtained via $Q[i,j]$), or there are at most $2(md_m)^2$ possible values for the original value of $Z_h[i,j]$ for $j \in D_h$. Therefore there are at most $(2n(md_m)^2)^{md_m}$ possible sets of $Z_h[i,j]$ for (5). The same upper bound holds for (5′).

The total number of choices for (4)–(5′) is $m^2(2n(md_m)^2)^{2md_m}$. We will write this as $C_m n^{2md_m}$, where C_m is the constant $2^{2m(d_m)}m^2(md_m)^{4md_m}$.

Recall that we are shipping one unit of flow from X to Y for all ordered pairs $(X,Y) \in \Omega^2$, and we want to find an upper bound on the amount of flow that passes through a given element $Z' \in \Omega$. The portion of flow from X to Y passing through Z' is determined by the distribution on the choices of permutations in (2). The permutation π defines the sequence $Z_0,\ldots,Z_{n'}$ which, in turn, defines $Z'_0,\ldots,Z'_{n'}$. We want to know how much flow passes through Z'_h for the choices (1), (3)–(5′) of the

encoding. Therefore, we see the amount passing through Z'_h is bounded above by

$$|\Omega| C_m \binom{n}{d_m - 1} n^{2md_m} \sum_h \sum_{U:\ |U|=h} \Pr[\pi\{1,\dots,h\} = U],$$

where $\Pr[\pi\{1,\dots,h\} = U]$ is the probability that U is the set of the first h columns to be changed.

Therefore (as in [20]), we need to bound $\sum_h \sum_{U:\ |U|=h} \Pr[\pi\{1,\dots,h\} = U]$ for each particular possibility for parts 1, 3, 4, 5, and 5′ of the encoding. As explained before, these parts of the encoding determine Z_h and $\bar{Z}_h$ (apart from column k). Z_h and $\bar{Z}_h$ together give us the set $\{X_j, Y_j\}$ for any $j \neq k$, but they don't tell us which of the two columns is X_j and which is Y_j. The relevant parts of the encoding also determine D_h. For the given value of D_h, there are at most $2^{d_m - 1}$ possibilities for R (which must include column k), and we shall sum over all of them. We then upper bound the flow coming from all permutations $\pi\{1,\dots,h\}$ on the columns of $[n] - R$. There are at most $(2(n - |R|))^m \leq (2n)^m$ possibilities for the vector M giving the M_i values and we like likewise sum over all such choices. For each choice of (possible) values $\{M_i\}$ we can compute from Z_h and $\bar{Z}_h$ the set $L_i = \{j : j \notin R,\ X[i,j] > M_i\} \cup \{j : j \notin R,\ Y[i,j] > M_i\}$. Note that this gives us $S = [n] - (L \cup R)$ as well. By our remark on page 254, we know that each L_i consists solely of columns of X, or solely of columns of Y. Therefore there are only two possibilities for assigning all the L_i columns to X or Y (2^m choices for all of $L = \cup_i L_i$).

Let $h_2 = h - |\pi\{1,\dots,h\} \cap L|$. We bound the flow passing through Z'_h below. Note that the first summation is over all $(2(n - |R|))^m$ possibilities for M and all 2^m possible assignments of L.

$$\begin{aligned}
&|\Omega| C_m \binom{n}{d_m - 1} n^{2md_m} \sum_{R,M,L,h_2} \sum_{U \subseteq S:\ |U|=h_2} \Pr[\pi_2\{1,\dots,h_2\} = U] \\
&\leq |\Omega| C_m \binom{n}{d_m - 1} n^{2md_m} 2^{d_m - 1} (2n)^m 2^m n \sum_{U \subseteq S:\ |U|=h_2} \Pr[\pi_2\{1,\dots,h_2\} = U] \\
&\leq |\Omega| C_m \binom{n}{d_m - 1} n^{2md_m} 2^{d_m - 1} (2n)^m 2^m n \sum_{U \subseteq S:\ |U|=h_2} p_m(|S|) \Big/ \binom{|S|}{h_2} \\
&\leq |\Omega| C_m \binom{n}{d_m - 1} n^{2md_m} 2^{d_m - 1} (2n)^m 2^m n \binom{|S|}{h_2} p_m(|S|) \Big/ \binom{|S|}{h_2} \\
&\leq |\Omega| C_m \binom{n}{d_m - 1} n^{2md_m} 2^{d_m - 1} (2n)^m 2^m n p_m(|S|),
\end{aligned}$$

which is $poly(n)\,|\Omega|$.

Thus we know that the flow through any *state* is bounded by a quantity f which is at most $poly(n)\,|\Omega|$. In the application of Morris and Sinclair [20] this is already sufficient to prove polynomial time mixing, since the term $P(e)$ in the denominator of (1) is only polynomially small. However, for our heat-bath chain P_{HB}, this term may be exponentially small, and a further argument is required to establish rapid mixing.

To this end, let $e = (\omega, \omega')$ $(\omega, \omega' \in \Omega^2)$ be a (directed) transition of $\mathcal{M}_{\mathrm{HB}}$, with transition probability $P_{\mathrm{HB}}(e)$. Suppose that f_e units of flow are shipped along e in the multicommodity flow $\mathcal{F}^*$ defined above. We will construct a new flow $\mathcal{F}$ in which these f_e units are dispersed, travelling from ω to ω' via a "random destination" ω''.

Let B be the set of columns on which ω and ω' disagree and let W be the set of all size $m \times (2d_m + 1)$ heat-bath windows which include B. Let Ω'' be the set of all contingency tables ω'' such that

1. there is a $U \in W$ which contains all the columns on which ω and ω'' differ, and

2. there is a $U' \in W$ which contains all the columns on which ω' and ω'' differ.

For each $\omega'' \in \Omega''$, the flow $\mathcal{F}$ will route $f_e/|\Omega''|$ flow from ω to ω' via ω''. Note that this construction doubles the length of our flow paths, but no more. Hence, the length of the longest path in the new flow $\mathcal{F}$ is at most $2(n+1)$.

The quantity shipped through (ω, ω'') in $\mathcal{F}$ from the original transition e in the multicommodity flow $\mathcal{F}^*$ is $f_e/|\Omega''|$, which is at most $4f_e n^{2d_m+1} P_{\mathrm{HB}}(\omega, \omega'')$. To see this, let K be the (nonempty) set of columns on which ω and ω'' differ. For every heat-bath window U, let $\Omega_\omega(U)$ denote the set of contingency tables which agree with ω, except possibly on window U, and write

$$\begin{aligned} P_{\mathrm{HB}}(\omega, \omega'') &= \frac{1}{4} \sum_{U \supseteq K} \frac{1}{\binom{n}{2d_m+1}} \frac{1}{|\Omega_\omega(U)|} \\ &\geq \frac{1}{4} \sum_{U \supseteq K, U \in W} \frac{1}{\binom{n}{2d_m+1}} \frac{1}{|\Omega_\omega(U)|} \\ &\geq \frac{1}{4} \sum_{U \supseteq K, U \in W} \frac{1}{\binom{n}{2d_m+1}} \frac{1}{|\Omega''|} \\ &\geq \frac{1}{4} \frac{1}{\binom{n}{2d_m+1}} \frac{1}{|\Omega''|}, \end{aligned}$$

where the last inequality follows from the fact that $\omega'' \in \Omega''$, so there is at least one U in the summation.

We call the transition (ω, ω'') a *type* 1 transition and a transition (ω'', ω') a *type* 2 transition.

We can now give an upper bound for the total type 1 flow along the transition (ω, ω''). For each $e = (\omega, \omega')$, we ship at most $4f_e n^{2d_m+1} P_{\mathrm{HB}}(\omega, \omega'')$ flow. Let f be the bound from above on the *total* flow that leaves node ω in our original multicommodity flow $\mathcal{F}^*$ (so $f = \sum_e f_e$, where the sum is over transitions e which start at ω). Then the total amount routed via ω'' in $\mathcal{F}$ is at most $4fn^{2d_m+1} P_{\mathrm{HB}}(\omega, \omega'')$.

Using a symmetric argument, we can show that the total type 2 flow along the transition (ω'', ω') in $\mathcal{F}$ is at most $4fn^{2d_m+1} P_{\mathrm{HB}}(\omega'', \omega')$.

Thus, the total flow in $\mathcal{F}$ along transition (ω, ω'') is at most $8fn^{2d_m+1} P_{\mathrm{HB}}(\omega, \omega'')$. Using the fact that the longest flow-carrying path length is at most $2(n+1)$, this is now sufficient for the right-hand side of (1) to be polynomially bounded, since the (possibly small) $P_{\mathrm{HB}}(e)$ term cancels. This completes the proof of Theorem 7. □

5. Mixing of the 2×2 chain. Theorem 7 shows that the Markov chain $\mathcal{M}_{\mathrm{HB}}$ is rapidly mixing. In this section we use the comparison method of Diaconis and Saloff-Coste [7] to show that the 2×2 chain $\mathcal{M}_{2\times 2}$ is also rapidly mixing. We briefly describe the comparison method, in the context of contingency tables, adapted from [7]. For more details and other examples of applications of this method, see [7], Randall and Tetali [23], and Vigoda [27].

5.1. Setting up the comparison. Recall that P_{HB} denotes the transition matrix of the Markov chain $\mathcal{M}_{\mathrm{HB}}$ which was analyzed in section 4. Let $E(P_{\mathrm{HB}})$ be

the set of edges (excluding loops) in the Markov kernel of that chain. That is, $E(P_{\mathrm{HB}}) = \{(X,Y) : X \neq Y \text{ and } P_{\mathrm{HB}}(X,Y) > 0\}$. Let $P_{2\times 2}$ denote the transition matrix of $\mathcal{M}_{2\times 2}$ and let $E(P_{2\times 2})$ denote the edge-set of its Markov kernel.

For every $(X,Y) \in E(P_{\mathrm{HB}})$ we define a set of paths $\Gamma_{X,Y}$ where each $\gamma \in \Gamma_{X,Y}$ is a path $X = \eta_0, \eta_1, \ldots, \eta_k = Y$, such that $(\eta_i, \eta_{i+1}) \in E(P_{2\times 2})$ for all $i \in \{0, \ldots, k-1\}$. Let $|\gamma|$ denote the length (number of edges) of the path. We also define a *flow* $f_{X,Y}$, which is a function from $\Gamma_{X,Y}$ to the positive reals satisfying the condition

$$\sum_{\gamma \in \Gamma_{X,Y}} f_{X,Y}(\gamma) = 1. \tag{8}$$

It is important to note that this flow need be defined only for pairs $(X,Y) \in E(P_{\mathrm{HB}})$, not for all pairs (X,Y).

For each $(Z,Z') \in E(P_{2\times 2})$, define the quantity

$$A_{Z,Z'} = \frac{1}{P_{2\times 2}(Z,Z')} \sum_{(X,Y)\in E(P_{\mathrm{HB}})} \sum_{\substack{\gamma \in \Gamma_{X,Y} \text{ such} \\ \text{that } (Z,Z') \in \gamma}} |\gamma| f_{X,Y}(\gamma) P_{\mathrm{HB}}(X,Y).$$

We use the comparison theorem of Diaconis and Saloff-Coste, which says[1] that

$$\tau_{2\times 2}(\epsilon) \in O(\tau_{HB}(\epsilon) \log(|\Sigma_{r,c}|) \max_{(Z,Z')\in E(P_{2\times 2})} A_{Z,Z'}).$$

In our construction of the flow, we ensure that the length of each path $\gamma \in \Gamma_{X,Y}$ is bounded by a constant. Thus, the theorem of Diaconis and Saloff-Coste tells us that to establish rapid mixing, we need only define $f_{X,Y}$ for every $(X,Y) \in E(P_{\mathrm{HB}})$ such that (8) is satisfied and also, for all $(Z,Z') \in E(P_{2\times 2})$, the following is satisfied:

$$\frac{1}{P_{2\times 2}(Z,Z')} \sum_{(X,Y)\in E(P_{\mathrm{HB}})} \sum_{\substack{\gamma \in \Gamma_{X,Y} \text{ such} \\ \text{that } (Z,Z') \in \gamma}} f_{X,Y}(\gamma) P_{\mathrm{HB}}(X,Y) \leq poly(n). \tag{9}$$

It helps us to rework (9) before defining the flows. For $(X,Y) \in E(P_{\mathrm{HB}})$, let $\mathcal{W}(X,Y)$ be the set of all $m \times (2d_m + 1)$ "windows" such that X and Y agree outside of W, where a "window" is just a set of m rows and $2d_m + 1$ columns. Note that

$$P_{\mathrm{HB}}(X,Y) = \frac{1}{4} \sum_{W\in\mathcal{W}(X,Y)} \frac{1}{\binom{n}{2d_m+1}} \frac{1}{|\Omega_X(W)|},$$

where $\Omega_X(W)$ is the set of all contingency tables that agree with X outside of W. We may view $P_{\mathrm{HB}}(X,Y)$ as (a multiple of) the average of the quantities $1/|\Omega_X(W)|$ over all windows $W \in \mathcal{W}(X,Y)$. Therefore, there is some $W(X,Y) \in \mathcal{W}(X,Y)$ such that

$$P_{\mathrm{HB}}(X,Y) \leq \frac{1}{|\Omega_X(W(X,Y))|}. \tag{10}$$

[1]The statement of the theorem in this form is from Vigoda [27] and Randall and Tetali [23]. The derivation of Proposition 4 in [23] required the eigenvalues of P_{HB} to be at least $1/2$, which is why we added the self-loops to $\mathcal{M}_{\mathrm{HB}}$. (Actually, bounding the eigenvalues above zero by any amount suffices.) The comparison uses the fact that the eigenvalues of $P_{2\times 2}$ are positive since this method provides a lower bound for $1 - \lambda_1(P_{2\times 2})$ in terms of $1 - \lambda_1(P_{\mathrm{HB}})$, and in these situations those differences directly relate to mixing times.

The essential idea to keep in mind in what follows is that routing the unit flow $f_{X,Y}$ from X to Y is done using paths of contingency tables that differ from one another solely on (a 2×2 part of) this specially chosen window $W(X,Y)$ that satisfies (10).

For each $m \times (2d_m + 1)$ window W, let $E_W = \{(X,Y) : (X,Y) \in E(P_{HB}) \text{ and } W(X,Y) = W\}$. Later, when we define our flows, we do the following for every fixed window W: For every $(X,Y) \in E_W$, we define a flow $f_{X,Y}$ such that (8) is satisfied. We also ensure that for all $(Z,Z') \in E(P_{2\times 2})$, the following is satisfied:

$$\frac{1}{P_{2\times 2}(Z,Z')} \sum_{(X,Y)\in E_W} \sum_{\substack{\gamma\in\Gamma_{X,Y} \text{ such} \\ \text{that } (Z,Z')\in\gamma}} f_{X,Y}(\gamma) P_{\mathrm{HB}}(X,Y) \le poly(n). \tag{11}$$

Since there are only polynomially many windows W, by summing over all of them we see that (11) implies (9), and ensures rapid mixing of $\mathcal{M}_{2\times 2}$.

For each window W, section 5.2 shows how to define a flow $f^*_{X,Y}$ for every $(X,Y) \in E_W$ such that

$$\sum_{\gamma\in\Gamma_{X,Y}} f^*_{X,Y}(\gamma) = 1$$

and the total flow through any contingency table $Z \in \Sigma_{r,c}$ is in $O(|\Omega_X(W)|)$. At the end of section 5.1 we will define $f_{X,Y}$ by modifying $f^*_{X,Y}$. Let $f^*_{X,Y}(Z)$ denote the amount of flow passing through the contingency table Z in the flow $f^*_{X,Y}$. Let $f^*(Z) = \sum_{(X,Y)\in E_W} f^*_{X,Y}(Z)$. Similarly, let $f_{X,Y}(Z,Z')$ denote the amount of flow passing through the transition (Z,Z') in the flow $f_{X,Y}$. Let $f(Z,Z') = \sum_{(X,Y)\in E_W} f_{X,Y}(Z,Z')$. Our construction of $f_{X,Y}$ from $f^*_{X,Y}$ will ensure that for every $(Z,Z') \in E(P_{2\times 2})$, we have

$$f(Z,Z') \le 4 f^*(Z) P_{2\times 2}(Z,Z') \binom{n}{2}\binom{m}{2}. \tag{12}$$

Thus, the left-hand side of (11) is equal to

$$\begin{aligned}
&\frac{1}{P_{2\times 2}(Z,Z')} \sum_{(X,Y)\in E_W} \sum_{\substack{\gamma\in\Gamma_{X,Y} \text{ such} \\ \text{that } (Z,Z')\in\gamma}} f_{X,Y}(\gamma) P_{\mathrm{HB}}(X,Y) \\
&\le \frac{1}{P_{2\times 2}(Z,Z')} \sum_{(X,Y)\in E_W} \sum_{\substack{\gamma\in\Gamma_{X,Y} \text{ such} \\ \text{that } (Z,Z')\in\gamma}} f_{X,Y}(\gamma) \frac{1}{|\Omega_X(W)|} \qquad (13)\\
&= \frac{1}{P_{2\times 2}(Z,Z')} \cdot \frac{1}{|\Omega_X(W)|} \sum_{(X,Y)\in E_W} f_{X,Y}(Z,Z') \\
&= \frac{f(Z,Z')}{P_{2\times 2}(Z,Z')} \cdot \frac{1}{|\Omega_X(W)|} \\
&\le \frac{4 f^*(Z)\binom{n}{2}\binom{m}{2}}{|\Omega_X(W)|} \le poly(n). \qquad (14)
\end{aligned}$$

Inequality (13) comes from (10), where $\Omega_X(W) = \Omega_X(W(X,Y))$, and from our definition of E_W. The first inequality in (14) comes from (12) which we establish shortly, and the second inequality in (14) comes from the fact that $f^*(Z) \in$

$O(|\Omega_X(W)|)$, which is established in section 5.2. We will then have shown that (11) is satisfied, as required, and thus $\mathcal{M}_{2\times 2}$ is rapidly mixing on $\Sigma_{r,c}$.

We finish this section by showing how to construct $f_{X,Y}$ given the flow $f^*_{X,Y}$, thereby establishing (12). The method is similar to (but simpler than) the one that we used at the end of section 4.2.

Recall that $E(P_{2\times 2})$ excludes self-transitions of the form (Z, Z). Thus, for each $(Z, Z'') \in E(P_{2\times 2})$, there is a *unique* 2×2 window $U(Z, Z'')$ on which Z and Z'' disagree. Let $\Omega_Z(U(Z, Z''))$ denote the set of all contingency tables that agree with Z (and Z'') everywhere outside of the 2×2 window $U(Z, Z'')$. Let $f^*_{X,Y}(Z, Z'')$ be the flow that passes through (Z, Z'') in $f^*_{X,Y}$. For each $Z' \in \Omega_Z(U(Z, Z''))$, some of this flow is allocated to the path Z, Z', Z''. In particular, $f^*_{X,Y}(Z, Z'')/|\Omega_Z(U(Z, Z''))|$ flow is sent this way.

Now $f_{X,Y}(Z, Z') \le 2\sum_{Z''\in U(Z,Z')} \frac{f^*_{X,Y}(Z,Z'')}{|\Omega_Z(U(Z,Z''))|}$, where the first "2" comes from the fact that we must consider paths Z, Z', Z'' above, and also paths that end in edge Z, Z'. The right-hand side is at most

$$2\frac{f^*_{X,Y}(Z)}{|\Omega_Z(U(Z, Z'))|} \le 4f^*_{X,Y}(Z)P_{2\times 2}(Z, Z')\binom{n}{2}\binom{m}{2};$$

thus (12) holds.

By considering all $m \times (2d_m + 1)$-sized windows which contain the two columns on which Z and Z' differ, we can see that for each $(Z, Z') \in E(P_{2\times 2})$, we have

$$A_{Z,Z'} \le C\binom{n}{2}\binom{n-2}{2d_m - 1},$$

where the constant C accounts for the maximum length of any (X, Y) path for $(X, Y) \in E(P_{\mathrm{HB}})$, and the constant factors arising in the bound for the flow $f^*(Z)$ over a single $m \times (2d_m + 1)$ window W. Therefore, we have the following theorem.

THEOREM 8.

$$\tau_{\mathcal{M}_{2\times 2}}(\epsilon) \in O(\tau_{\mathcal{M}_{\mathrm{HB}}}(\epsilon)\log(|\Sigma_{r,c}|)n^{2d_m+1}).$$

5.2. Defining $f^*(X, Y)$. In this section we define a flow $f^*_{X,Y}$ for every $(X, Y) \in E_W$ such that $\sum_{\gamma\in\Gamma_{X,Y}} f^*_{X,Y}(\gamma) = 1$ and the total flow through any contingency table Z, due to pairs in E_W, is in $O(|\Omega_X(W)|)$.

Throughout this entire section, we therefore focus on some fixed $m \times (2d_m + 1)$-sized window W of the larger $m \times n$ table. Without loss of generality (and to make our notation simpler in what follows), we assume that W includes the first $2d_m + 1$ columns of the table. This window W has induced row sums ρ_i (for $i \in [m]$) and induced column sums ζ_j (for $j \in [2d_m + 1]$). For convenience we also set $\delta = 2d_m + 1$.

Let $\rho = (\rho_1, \ldots, \rho_m)$, $\zeta = (\zeta_1, \ldots, \zeta_\delta)$ be the lists of induced row and column sums. Let $\Sigma_{\rho,\zeta}$ denote the set of $m \times \delta$ contingency tables with rows sums ρ and column sums ζ, and let N_W denote the table sum. Let $\Upsilon, \Psi \in \Sigma_{\rho,\zeta}$. We show how to route a unit of flow between Υ and Ψ using a path of contingency tables that differ by moves of $\mathcal{M}_{2\times 2}$. This flow lifts in the obvious fashion to transitions $(X, Y) \in E(P_{\mathrm{HB}})$, giving us the flow $f^*_{X,Y}$ required in the previous section. In other words, we simply use the exact same sequence of 2×2 transitions on the window $W(X, Y)$, keeping everything outside this window fixed (where X and Y agree anyway).

If $N_W < (2m\delta)^2$, then $|\Sigma_{\rho,\zeta}| \in O(1)$; thus it does not really matter how we route flow between Υ and Ψ. For example, it suffices to fix each square in the contingency

table in lexicographic order. Each path in the resulting flow is of length $O(1)$, and there are $O(1)$ pairs (Υ, Ψ) of contingency tables, so the desired bound is easily established. This is similar to, but simpler than, what we do later in section 5.2.3. Thus, from now on we assume $N_W \geq (2m\delta)^2$ and show how to construct a flow between Υ and Ψ in $\Sigma_{\rho,\zeta}$.

Without loss of generality we may assume that the row totals are sorted into nondescending order and that the column totals are also sorted into nondescending order. Therefore ρ_m is the largest row sum and ζ_δ is the largest column sum.

As we did in section 4.1.2, we view the space $\Sigma_{\rho,\zeta}$ of contingency tables as the $(m-1)(\delta-1)$-dimensional space of integer matrices Φ that satisfy $\Phi[i,j] \geq 0$ for all $i \in [m-1]$ and all $j \in [\delta-1]$, and also satisfy the following inequalities (see Dyer, Kannan, and Mount [12]):

$$\sum_{i=1}^{m-1} \Phi[i,j] \leq \zeta_j \quad \text{for every } j \in [\delta-1], \tag{15}$$

$$\sum_{j=1}^{\delta-1} \Phi[i,j] \leq \rho_i \quad \text{for every } i \in [m-1], \tag{16}$$

$$\sum_{i=1}^{m-1} \sum_{j=1}^{\delta-1} \Phi[i,j] \geq N_W - \rho_m - \zeta_\delta. \tag{17}$$

Let

$$\alpha_{i,j} = \left\lfloor \frac{\min\{\rho_i, \zeta_j\}}{m^2\delta^2} \right\rfloor \tag{18}$$

for all $i \in [m-1]$, $j \in [\delta-1]$.

Note that

$$\begin{aligned} \max_{j \in [\delta-1]} \alpha_{i,j} &= \alpha_{i,\delta-1} \text{ for all } i; \\ \max_{i \in [m-1]} \alpha_{i,j} &= \alpha_{m-1,j} \text{ for all } j; \\ \max_{i \in [m-1], j \in [\delta-1]} \alpha_{i,j} &= \alpha_{m-1,\delta-1}. \end{aligned}$$

For any contingency table $\Phi \in \Sigma_{\rho,\zeta}$, and any $i \in [m-1]$, $j \in [\delta-1]$, we can write

$$\Phi[i,j] = Q[i,j](\alpha_{i,j}+1) + R[i,j]$$

for a unique integer $R[i,j]$ satisfying $0 \leq R[i,j] \leq \alpha_{i,j}$, and a unique integer $Q[i,j]$.

For all $i \in [m-1]$, $j \in [\delta-1]$ we define

$$Q^*[i,j] = \left\lfloor \frac{\rho_i \zeta_j}{N_W(\alpha_{i,j}+1)} \right\rfloor. \tag{19}$$

Let $\Upsilon, \Psi \in \Sigma_{\rho,\zeta}$. We are almost ready to start building a path from Υ to Ψ that uses transitions of $\mathcal{M}_{2\times 2}$. We use the following lemma.

LEMMA 9. *Let $\Phi^*[i,j]$ be defined by*

$$\Phi^*[i,j] = Q^*[i,j](\alpha_{i,j}+1) + R[i,j]$$

for any nonnegative integers $R[i,j] \le \alpha_{i,j}$, *for all* $i \in [m-1]$, $j \in [\delta-1]$. *Also, we set*

$$\Phi^*[i,\delta] = \rho_i - \sum_{j=1}^{\delta-1} \Phi^*[i,j] \qquad \text{for } i \in [m-1],$$

$$\Phi^*[m,j] = \zeta_j - \sum_{i=1}^{m-1} \Phi^*[i,j] \qquad \text{for } j \in [\delta-1],$$

$$\text{and} \quad \Phi^*[m,\delta] = \zeta_\delta - \sum_{i=1}^{m-1} \Phi^*[i,\delta].$$

Then the following hold.

(i) $\Phi^* \in \Sigma_{\rho,\zeta}$.

(ii) *Under the assumption that* $N_W \ge (2m\delta)^2$, *we have the following:*

$$\Phi^*[i,\delta] \ge (\alpha_{i,\delta-1}+1) \text{ for all } i \in [m-1],$$
$$\Phi^*[m,j] \ge (\alpha_{m-1,j}+1) \text{ for all } j \in [\delta-1],$$
$$\Phi^*[m,\delta] \ge 2(\alpha_{m-1,\delta-1}+1).$$

Proof. (i) It suffices to show that the inequalities (15), (16), and (17) hold for the defined values of $\Phi^*[i,j]$ with $i \in [m-1]$ and $j \in [\delta-1]$. (Then the definitions of $\Phi^*[i,\delta]$, $\Phi^*[m,j]$, and $\Phi^*[m,\delta]$ are such that the entire $m \times \delta$ table satisfies the row and column sums ρ and ζ, respectively.)

This proof is analogous to that given in Case 2 in section 4.1.2 when we showed that inequalities (4), (5), and (6) held there, so we do not repeat that proof here.

(ii) By definition,

$$\begin{aligned}
\Phi^*[i,\delta] &= \rho_i - \sum_{j=1}^{\delta-1} \Phi^*[i,j] \\
&= \rho_i - \sum_{j=1}^{\delta-1} \left(\left\lfloor \frac{\rho_i \zeta_j}{N_W(\alpha_{i,j}+1)} \right\rfloor (\alpha_{i,j}+1) + R[i,j] \right) \\
&\ge \rho_i - \sum_{j=1}^{\delta-1} \left(\frac{\rho_i \zeta_j}{N_W} + \alpha_{i,j} \right) \\
&= \frac{\rho_i \zeta_\delta}{N_W} - \sum_{j=1}^{\delta-1} \alpha_{i,j} \\
&\ge \frac{\rho_i \zeta_\delta}{N_W} - \frac{\delta \rho_i}{m^2\delta^2} \\
&\ge \frac{\rho_i}{\delta} - \frac{\rho_i}{m^2\delta} \\
&\ge \frac{\rho_i}{m\delta} \\
&\ge 2\alpha_{i,\delta-1}.
\end{aligned}$$

Then, if $\alpha_{i,\delta-1} \ge 1$, we automatically have $\Phi^*[i,\delta] \ge (\alpha_{i,\delta-1}+1)$. However, if $\alpha_{i,\delta-1} = 0$, then $\alpha_{i,j} = 0$ for all $j \in [\delta-1]$, and there are two subcases to consider.

The first subcase is when $\rho_i\zeta_\delta < N_W$. Then $\rho_i\zeta_j < N_W$ for all $j \in [\delta-1]$, and $\Phi^*[i,j] = 0$ for all $j \in [\delta-1]$. Therefore $\Phi^*[i,\delta] = \rho_i \geq 1$. If $\rho_i\zeta_\delta \geq N_W$, then the fourth line of the derivation above (with $\alpha_{i,j} = 0$) gives $\Phi^*[i,\delta] \geq \rho_i\zeta_\delta/N_W \geq 1$. So in either subcase we have $\Phi^*[i,\delta] \geq 1 = (\alpha_{i,\delta-1}+1)$.

Using a similar argument we conclude that

$$\Phi^*[m,j] \geq (\alpha_{m-1,j}+1).$$

By definition

$$\begin{aligned}
\Phi^*[m,\delta] &= \zeta_\delta - \sum_{i=1}^{m-1} \Phi^*[i,\delta] \\
&= \zeta_\delta - \sum_{i=1}^{m-1} \left(\rho_i - \sum_{j=1}^{\delta-1} \Phi^*[i,j] \right) \\
&= (\rho_m + \zeta_\delta - N_W) + \sum_{i=1}^{m-1}\sum_{j=1}^{\delta-1} \left(\left\lfloor \frac{\rho_i\zeta_j}{N_W(\alpha_{i,j}+1)} \right\rfloor (\alpha_{i,j}+1) + R[i,j] \right) \\
&\geq (\rho_m + \zeta_\delta - N_W) + \sum_{i=1}^{m-1}\sum_{j=1}^{\delta-1} \left(\left\lfloor \frac{\rho_i\zeta_j}{N_W(\alpha_{i,j}+1)} \right\rfloor (\alpha_{i,j}+1) \right) \\
&\geq (\rho_m + \zeta_\delta - N_W) + \sum_{i=1}^{m-1}\sum_{j=1}^{\delta-1} \left(\frac{\rho_i\zeta_j}{N_W} - (\alpha_{i,j}+1) \right) \\
&= \frac{\rho_m\zeta_\delta}{N_W} - \sum_{i=1}^{m-1}\sum_{j=1}^{\delta-1} (\alpha_{i,j}+1) \\
&\geq \frac{\rho_m\zeta_\delta}{N_W} - \frac{m(N_W-\zeta_\delta)}{m^2\delta^2} - (m-1)(\delta-1) \\
&\geq \frac{N_W}{m\delta} - \frac{N_W}{m\delta^2} - (m-1)(\delta-1) \\
&\geq \frac{N_W}{4m\delta} + \left(\frac{N_W}{2m\delta} - \frac{N_W}{m\delta^2} \right) + \left(\frac{N_W}{4m\delta} - (m-1)(\delta-1) \right) \\
&\geq \frac{2\zeta_{\delta-1}}{4m\delta} + 0 + (\delta+m-1) \\
&\geq 2\alpha_{m-1,\delta-1} + 2,
\end{aligned}$$

as required. □

DEFINITION 10. *If Φ is a contingency table such that $Q[i,j] = Q^*[i,j]$ for every $i \in [m-1]$, $j \in [\delta-1]$, then we say that Φ belongs to the* inner domain *of $\Sigma_{\rho,\zeta}$.*

Now we consider a pair $\Upsilon, \Psi \in \Sigma_{\rho,\delta}$. For every $i \in [m-1]$, $j \in [\delta-1]$, we write

$$\Upsilon[i,j] = Q_\Upsilon[i,j](\alpha_{i,j}+1) + R_\Upsilon[i,j]$$

for the unique integer $R_\Upsilon[i,j]$ that satisfies $0 \leq R_\Upsilon[i,j] \leq \alpha_{i,j}$.

Similarly, for every $i \in [m-1]$, $j \in [\delta-1]$, we write

$$\Psi[i,j] = Q_\Psi[i,j](\alpha_{i,j}+1) + R_\Psi[i,j]$$

for the unique $R_\Psi[i,j]$ satisfying $0 \leq R_\Psi[i,j] \leq \alpha_{i,j}$.

The routing from Υ to Ψ proceeds in two stages. In the first phase, we route flow from Υ to the table Υ^* in the inner domain of $\Sigma_{\rho,\zeta}$ such that $\Upsilon^*[i,j] = Q^*[i,j](\alpha_{i,j}+1) + R_\Upsilon[i,j]$ holds for every $i \in [m-1]$, $j \in [\delta-1]$. Note that $\Upsilon^* \in \Sigma_{\rho,\zeta}$ using the previous lemma.

By defining a similar path between Ψ and Ψ^* and then reversing all the edges, we can route flow from some Ψ^* in the inner domain of $\Sigma_{\rho,\zeta}$ to Ψ such that $\Psi^*[i,j] = Q^*[i,j](\alpha_{i,j}+1) + R_\Psi[i,j]$ holds for every $i \in [m-1]$, $j \in [\delta-1]$. Similarly, we also have $\Psi^* \in \Sigma_{\rho,\zeta}$.

In the second phase of the routing we show how to route flow from Υ^* to Ψ^* by changing the $R_\Upsilon[i,j]$ to the $R_\Psi[i,j]$ values.

5.2.1. Phase 1. We show how to route Υ to Υ^* in the inner domain of $\Sigma_{\rho,\zeta}$, by changing only $Q[i,j]$ values at any step. For our analysis, we define the following metric on pairs $\Phi, \Phi' \in \Sigma_{\rho,\zeta}$:

$$
\begin{aligned}
d(\Phi,\Phi') &= d_1(\Phi,\Phi') + d_2(\Phi,\Phi') + d_3(\Phi,\Phi') + d_4(\Phi,\Phi'), \qquad \text{where} \\
d_1(\Phi,\Phi') &= \sum_{i=1}^{m-1}\sum_{j=1}^{\delta-1} |\Phi[i,j] - \Phi'[i,j]| \\
d_2(\Phi,\Phi') &= \frac{N_W+1}{3N_W}\left(\sum_{i=1}^{m-1} |\Phi[i,\delta] - \Phi'[i,\delta]|\right) \\
d_3(\Phi,\Phi') &= \frac{N_W+1}{3N_W}\left(\sum_{j=1}^{\delta-1} |\Phi[m,j] - \Phi'[m,j]|\right) \\
d_4(\Phi,\Phi') &= \frac{N_W+1}{3N_W}|\Phi[m,\delta] - \Phi'[m,\delta]|.
\end{aligned}
$$

This metric is used to show that the path we construct is moving us closer to Υ^* and that the length of this path from Υ to Υ^* is bounded by a constant.

We route Υ to Υ^* as a series of moves. Let Υ' denote some interim contingency table on the path from Υ to Υ^*. We choose our next move on this path from amongst four cases.

Case (a). Suppose that $\Upsilon'[m,\delta] < (\alpha_{m-1,\delta-1}+1)$. Then we perform a move to make $\Upsilon'[m,\delta]$ bigger (because we need to "leave room" for our other cases).

We now show there is at least one $\ell \in [m-1]$ such that $\Upsilon'[\ell,\delta] \geq \Upsilon^*[\ell,\delta] + (\alpha_{\ell,\delta-1}+1)$.

Note that by Lemma 9, in this case $\Upsilon'[m,\delta] < \Upsilon^*[m,\delta] - (\alpha_{m-1,\delta-1}+1)$. Suppose (for contradiction) that there is no ℓ as described above, so that

$$\Upsilon'[i,\delta] < \Upsilon^*[i,\delta] + (\alpha_{i,\delta-1}+1) \quad \text{for } i \in [m-1].$$

By definition

$$
\begin{aligned}
\Upsilon^*[i,\delta] &= \rho_i - \left(\sum_{j=1}^{\delta-1} \Upsilon^*[i,j]\right) \\
&= \rho_i - \left(\sum_{j=1}^{\delta-1} \left(\left\lfloor \frac{\rho_i \zeta_j}{N_W(\alpha_{i,j}+1)} \right\rfloor (\alpha_{i,j}+1) + R_\Upsilon[i,j]\right)\right).
\end{aligned}
$$

Now

$$\begin{aligned}\Upsilon'[m,\delta] &= \zeta_\delta - \left(\sum_{i=1}^{m-1}\Upsilon'[i,\delta]\right)\\ &> \zeta_\delta - \sum_{i=1}^{m-1}\left(\rho_i - \left(\sum_{j=1}^{\delta-1}\Upsilon^*[i,j]\right) + (\alpha_{i,\delta-1}+1)\right)\\ &= (\zeta_\delta + \rho_m - N_W) + \sum_{i=1}^{m-1}\left(\left(\sum_{j=1}^{\delta-1}\Upsilon^*[i,j]\right) - (\alpha_{i,\delta-1}+1)\right). \qquad (20)\end{aligned}$$

We expand $\sum_{i=1}^{m-1}((\sum_{j=1}^{\delta-1}\Upsilon^*[i,j]) - (\alpha_{i,\delta-1}+1))$ as

$$\begin{aligned}&\sum_{i=1}^{m-1}\left(\sum_{j=1}^{\delta-1}\left(\left\lfloor\frac{\rho_i\zeta_j}{N_W(\alpha_{i,j}+1)}\right\rfloor(\alpha_{i,j}+1) + R_\Upsilon[i,j]\right) - (\alpha_{i,\delta-1}+1)\right)\\ &\geq \sum_{i=1}^{m-1}\left(\sum_{j=1}^{\delta-1}\left(\frac{\rho_i\zeta_j}{N_W} - (\alpha_{i,j}+1)\right) - (\alpha_{i,\delta-1}+1)\right)\\ &= (N_W - \rho_m - \zeta_\delta) + \frac{\rho_m\zeta_\delta}{N_W} - \sum_{i=1}^{m-1}\sum_{j=1}^{\delta-1}(\alpha_{i,j}+1) - \sum_{i=1}^{m-1}(\alpha_{i,\delta-1}+1)\\ &\geq (N_W - \rho_m - \zeta_\delta) + \frac{\rho_m\zeta_\delta}{N_W} - \frac{m(N_W-\zeta_\delta)}{m^2\delta^2} - (m-1)(\delta-1) - \frac{m\zeta_{\delta-1}}{m^2\delta^2} - (m-1)\\ &\geq (N_W - \rho_m - \zeta_\delta) + \frac{N_W}{m\delta} - \frac{N_W}{m\delta^2} - m\delta + \delta\\ &\geq (N_W - \rho_m - \zeta_\delta) + \frac{N_W}{2m\delta} - m\delta + \delta\\ &\geq (N_W - \rho_m - \zeta_\delta) + \frac{N_W}{4m\delta} + \delta\\ &\geq (N_W - \rho_m - \zeta_\delta) + (\alpha_{m-1,\delta-1}+1), \qquad (21)\end{aligned}$$

where the second-to-last line follows since $N_W \geq (2m\delta)^2$, and the last line follows from the definition of $\alpha_{i,j}$ and because $\delta \geq m \geq 2$.

Combining (20) and (21), we have a contradiction to our original assumption that $\Upsilon'[m,\delta] < (\alpha_{m-1,\delta-1}+1)$. Therefore it must be that $\Upsilon'[\ell,\delta] \geq \Upsilon^*[\ell,\delta] + (\alpha_{\ell,\delta-1}+1)$ for some $\ell \in [m-1]$.

Similarly, there must be some $k \in [\delta-1]$ such that $\Upsilon'[m,k] \geq \Upsilon^*[m,k] + (\alpha_{m-1,k}+1)$.

In this case we add the transition $(\Upsilon' \to \Upsilon'')$ to our path from Υ to Υ^*, where Υ'' is identical to Υ' except for the following entries:

$$\begin{aligned}\Upsilon''[\ell,k] &= \Upsilon'[\ell,k] + (\alpha_{\ell,k}+1), \quad \Upsilon''[\ell,\delta] = \Upsilon'[\ell,\delta] - (\alpha_{\ell,k}+1),\\ \Upsilon''[m,k] &= \Upsilon'[m,k] - (\alpha_{\ell,k}+1), \quad \Upsilon''[m,\delta] = \Upsilon'[m,\delta] + (\alpha_{\ell,k}+1).\end{aligned}$$

Now we show that $d(\Upsilon'', \Upsilon^*) < d(\Upsilon', \Upsilon^*)$:

$$d_1(\Upsilon'', \Upsilon^*) = \sum_{i=1}^{m-1}\sum_{j=1}^{\delta-1} |\Upsilon''[i,j] - \Upsilon^*[i,j]| \leq d_1(\Upsilon', \Upsilon^*) + (\alpha_{\ell,k}+1),$$

$$d_2(\Upsilon'', \Upsilon^*) = \frac{N_W+1}{3N_W}\left(\sum_{i=1}^{m-1} |\Upsilon''[i,\delta] - \Upsilon^*[i,\delta]|\right) = d_2(\Upsilon', \Upsilon^*) - \frac{N_W+1}{3N_W}(\alpha_{\ell,k}+1),$$

$$d_3(\Upsilon'', \Upsilon^*) = \frac{N_W+1}{3N_W}\left(\sum_{j=1}^{\delta-1} |\Upsilon''[m,j] - \Upsilon^*[m,j]|\right) = d_3(\Upsilon', \Upsilon^*) - \frac{N_W+1}{3N_W}(\alpha_{\ell,k}+1),$$

$$d_4(\Upsilon'', \Upsilon^*) = \frac{N_W+1}{3N_W}|\Upsilon''[m,\delta] - \Upsilon^*[m,\delta]| = d_4(\Upsilon', \Upsilon^*) - \frac{N_W+1}{3N_W}(\alpha_{\ell,k}+1).$$

The equation for d_4 follows from Lemma 9.

Thus $d(\Upsilon'', \Upsilon^*) \leq d(\Upsilon', \Upsilon^*) + (\alpha_{\ell,k}+1) - (\alpha_{\ell,k}+1)(N_W+1)/N_W \leq d(\Upsilon', \Upsilon^*) - (\alpha_{\ell,k}+1)/N_W < d(\Upsilon', \Upsilon^*)$.

Case (b). We execute (b) whenever Case (a) does not hold and when *either* $\Upsilon'[\ell,\delta] < \Upsilon^*[\ell,\delta]$ for some $\ell \in [m-1]$ *or* $\Upsilon'[m,j] < \Upsilon^*[m,j]$ for some $j \in [\delta-1]$. Without loss of generality, assume the former. If this is the case, then there must be at least one $k \in [\delta-1]$ such that $\Upsilon'[\ell,k] > \Upsilon^*[\ell,k]$. Since we change only $Q[i,j]$ values during our routing, we know that $\Upsilon'[\ell,k] \geq \Upsilon^*[\ell,k] + (\alpha_{\ell,k}+1)$. Also, since we are not in Case (a), we know that $\Upsilon'[m,\delta] \geq (\alpha_{m-1,\delta-1}+1) \geq (\alpha_{\ell,k}+1)$.

In this case we add the transition $(\Upsilon' \to \Upsilon'')$ in our path from Υ to Υ^*, where Υ'' is identical to Υ' except for the following entries:

$$\Upsilon''[\ell,k] = \Upsilon'[\ell,k] - (\alpha_{\ell,k}+1), \quad \Upsilon''[\ell,\delta] = \Upsilon'[\ell,\delta] + (\alpha_{\ell,k}+1),$$
$$\Upsilon''[m,k] = \Upsilon'[m,k] + (\alpha_{\ell,k}+1), \quad \Upsilon''[m,\delta] = \Upsilon'[m,\delta] - (\alpha_{\ell,k}+1).$$

Now we show that $d(\Upsilon'', \Upsilon^*) < d(\Upsilon', \Upsilon^*)$:

$$d_1(\Upsilon'', \Upsilon^*) = \sum_{i=1}^{m-1}\sum_{j=1}^{\delta-1} |\Upsilon''[i,j] - \Upsilon^*[i,j]| = d_1(\Upsilon', \Upsilon^*) - (\alpha_{\ell,k}+1),$$

$$d_2(\Upsilon'', \Upsilon^*) = \frac{N_W+1}{3N_W}\left(\sum_{i=1}^{m-1} |\Upsilon''[i,\delta] - \Upsilon^*[i,\delta]|\right) \leq d_2(\Upsilon', \Upsilon^*) + \frac{N_W+1}{3N_W}(\alpha_{\ell,k}-1),$$

$$d_3(\Upsilon'', \Upsilon^*) = \frac{N_W+1}{3N_W}\left(\sum_{j=1}^{\delta-1} |\Upsilon''[m,j] - \Upsilon^*[m,j]|\right) \leq d_3(\Upsilon', \Upsilon^*) + \frac{N_W+1}{3N_W}(\alpha_{\ell,k}+1),$$

$$d_4(\Upsilon'', \Upsilon^*) = \frac{N_W+1}{3N_W}|\Upsilon''[m,\delta] - \Upsilon^*[m,\delta]| \leq d_4(\Upsilon', \Upsilon^*) + \frac{N_W+1}{3N_W}(\alpha_{\ell,k}+1),$$

where the expression for $d_2(\Upsilon'', \Upsilon^*)$ follows because $\Upsilon'[\ell,\delta] \leq \Upsilon^*[\ell,\delta] - 1$.

Therefore $d(\Upsilon'', \Upsilon^*) \leq d(\Upsilon', \Upsilon^*) - (\alpha_{\ell,k}+1) + (\alpha_{\ell,k}+1)(N_W+1)/N_W - 2(N_W+1)/3N_W$. This is at most $d(\Upsilon', \Upsilon^*) + (\alpha_{\ell,k}+1)/N_W - 2(N_W+1)/3N_W$. Then because $\alpha_{\ell,k} \leq N_W/(\delta m)^2$, we have $d(\Upsilon'', \Upsilon^*) \leq d(\Upsilon', \Upsilon^*) + 1/(m\delta)^2 + 1/3N_W - 2/3$, and using $N_W \geq (2\delta m)^2$, we obtain $d(\Upsilon'', \Upsilon^*) < d(\Upsilon', \Upsilon^*)$.

Case (c). We execute (c) when Cases (a)–(b) do not hold and *either* $\Upsilon'[\ell,\delta] > \Upsilon^*[\ell,\delta]$ for some $\ell \in [m-1]$ *or* $\Upsilon'[m,j] > \Upsilon^*[m,j]$ for some $j \in [\delta-1]$. Assume the former without loss of generality. Then there must exist some $k \in [\delta-1]$ such

that $\Upsilon'[\ell,k] < \Upsilon^*[\ell,k]$, and since we can only change the $\Upsilon[\ell,k]$ values by factors of $(\alpha_{\ell,k}+1)$, we have $\Upsilon'[\ell,k] \le \Upsilon^*[\ell,k] + (\alpha_{\ell,k}+1)$. Note that since $\Upsilon'[\ell,\delta] > \Upsilon^*[\ell,\delta]$ and using part (ii) of Lemma 9, we know $\Upsilon'[\ell,\delta] \ge (\alpha_{\ell,\delta-1}+1) \ge (\alpha_{\ell,k}+1)$. Because Case (b) does not hold, we know $\Upsilon'[m,k] \ge \Upsilon^*[m,k]$, and, from part (ii) of Lemma 9, this is at least $(\alpha_{m-1,k}+1) \ge (\alpha_{\ell,k}+1)$.

Therefore we can perform the transition $(\Upsilon' \to \Upsilon'')$, where Υ'' is identical to Υ' except for the following entries:

$$\begin{aligned}\Upsilon''[\ell,k] &= \Upsilon'[\ell,k] + (\alpha_{\ell,k}+1), \quad & \Upsilon''[\ell,\delta] &= \Upsilon'[\ell,\delta] - (\alpha_{\ell,k}+1),\\ \Upsilon''[m,k] &= \Upsilon'[m,k] - (\alpha_{\ell,k}+1), \quad & \Upsilon''[m,\delta] &= \Upsilon'[m,\delta] + (\alpha_{\ell,k}+1).\end{aligned}$$

As before, we now show that $d(\Upsilon'',\Upsilon^*) < d(\Upsilon',\Upsilon^*)$:

$$\begin{aligned}
d_1(\Upsilon'',\Upsilon^*) &= \sum_{i=1}^{m-1}\sum_{j=1}^{\delta-1} |\Upsilon''[i,j] - \Upsilon^*[i,j]| = && d_1(\Upsilon',\Upsilon^*) - (\alpha_{\ell,k}+1),\\
d_2(\Upsilon'',\Upsilon^*) &= \frac{N_W+1}{3N_W}\sum_{i=1}^{m-1} |\Upsilon''[i,\delta] - \Upsilon^*[i,\delta]| \le && d_2(\Upsilon',\Upsilon^*) + \frac{N_W+1}{3N_W}(\alpha_{\ell,k}-1),\\
d_3(\Upsilon'',\Upsilon^*) &= \frac{N_W+1}{3N_W}\sum_{j=1}^{\delta-1} |\Upsilon''[m,j] - \Upsilon^*[m,j]| \le && d_3(\Upsilon',\Upsilon^*) + \frac{N_W+1}{3N_W}(\alpha_{\ell,k}+1),\\
d_4(\Upsilon'',\Upsilon^*) &= \frac{N_W+1}{3N_W}|\Upsilon''[m,\delta] - \Upsilon^*[m,\delta]| \le && d_4(\Upsilon',\Upsilon^*) + \frac{N_W+1}{3N_W}(\alpha_{\ell,k}+1),
\end{aligned}$$

where the bound on $d_2(\Upsilon'',\Upsilon^*)$ follows, using that $\Upsilon'[\ell,\delta] \ge \Upsilon^*[\ell,\delta]+1$.

Case (d). This is the case when $\Upsilon'[i,\delta] = \Upsilon^*[i,\delta]$ for all $i \in [m-1]$, and $\Upsilon'[m,j] = \Upsilon^*[m,j]$ for all $j \in [\delta-1]$ (so neither Case (b) nor (c) holds), but $\Upsilon'[\ell,k] \ne \Upsilon^*[\ell,k]$ for some $\ell \in [m-1], k \in [\delta-1]$. We also assume Case (a) does not hold; otherwise we would not consider Case (d). In this case we will specify *two* transitions of $\mathcal{M}_{2\times 2}$, $\Upsilon' \to \Upsilon''$ and $\Upsilon'' \to \Upsilon'''$, so that $d(\Upsilon''',\Upsilon^*) < d(\Upsilon',\Upsilon^*)$.

Assume without loss of generality that $\Upsilon'[\ell,k] > \Upsilon^*[\ell,k]$. Hence, there must be some $k' \in [\delta-1]$ such that $\Upsilon'[\ell,k'] < \Upsilon^*[\ell,k']$.

Now because we change only $Q[i,j]$ values on the path from Υ to Υ^*, $\Upsilon'[\ell,k] > \Upsilon^*[\ell,k]$ implies $\Upsilon'[\ell,k] \ge \Upsilon^*[\ell,k] + (\alpha_{\ell,k}+1)$, and $\Upsilon'[\ell,k'] \le \Upsilon^*[\ell,k'] + (\alpha_{\ell,k'}+1)$.

By Lemma 9 and $\Upsilon'[m,\delta] = \Upsilon^*[m,\delta]$ we know $\Upsilon'[m,\delta] \ge (\alpha_{\ell,k}+1)$. Therefore we can perform the transition $(\Upsilon' \to \Upsilon'')$, where Υ'' is identical to Υ' except for the following entries:

$$\begin{aligned}\Upsilon''[\ell,k] &= \Upsilon'[\ell,k] - (\alpha_{\ell,k}+1), \quad & \Upsilon''[\ell,\delta] &= \Upsilon'[\ell,\delta] + (\alpha_{\ell,k}+1),\\ \Upsilon''[m,k] &= \Upsilon'[m,k] + (\alpha_{\ell,k}+1), \quad & \Upsilon''[m,\delta] &= \Upsilon'[m,\delta] - (\alpha_{\ell,k}+1).\end{aligned}$$

By Lemma 9 we also know $\Upsilon'[\ell,\delta] \ge (\alpha_{\ell,k'}+1)$ and $\Upsilon'[m,k'] \ge (\alpha_{\ell,k'}+1)$. Then we can perform the transition $(\Upsilon'' \to \Upsilon''')$ by changing the following entries:

$$\begin{aligned}\Upsilon'''[\ell,k'] &= \Upsilon''[\ell,k'] + (\alpha_{\ell,k'}+1), \quad & \Upsilon'''[\ell,\delta] &= \Upsilon''[\ell,\delta] - (\alpha_{\ell,k'}+1),\\ \Upsilon'''[m,k'] &= \Upsilon''[m,k'] - (\alpha_{\ell,k'}+1), \quad & \Upsilon'''[m,\delta] &= \Upsilon''[m,\delta] + (\alpha_{\ell,k'}+1).\end{aligned}$$

Finally we show that $d(\Upsilon''', \Upsilon^*) < d(\Upsilon', \Upsilon^*)$.

$$\begin{aligned}
d_1(\Upsilon''', \Upsilon^*) &= d_1(\Upsilon', \Upsilon^*) - (\alpha_{\ell,k} + \alpha_{\ell,k'} + 2),\\
d_2(\Upsilon''', \Upsilon^*) &= d_2(\Upsilon', \Upsilon^*) + \tfrac{N_W+1}{3N_W}|\alpha_{\ell,k} - \alpha_{\ell,k'}|,\\
d_3(\Upsilon''', \Upsilon^*) &= d_3(\Upsilon', \Upsilon^*) + \tfrac{N_W+1}{3N_W}(\alpha_{\ell,k} + \alpha_{\ell,k'} + 2),\\
d_4(\Upsilon''', \Upsilon^*) &= d_4(\Upsilon', \Upsilon^*) + \tfrac{N_W+1}{3N_W}|\alpha_{\ell,k} - \alpha_{\ell,k'}|.
\end{aligned}$$

Therefore

$$\begin{aligned}
d(\Upsilon''', \Upsilon^*) &= d(\Upsilon', \Upsilon^*) - (2/3 - 1/3N_W)(\alpha_{\ell,k} + \alpha_{\ell,k'} + 2)\\
&\quad + (2/3 + 2/3N_W)|\alpha_{\ell,k} - \alpha_{\ell,k'}|\\
&\le d(\Upsilon', \Upsilon^*) - (2/3 - 1/3N_W)(\alpha_{\ell,k} + \alpha_{\ell,k'} + 2)\\
&\quad + (2/3 + 2/3N_W)(\alpha_{\ell,k} + \alpha_{\ell,k'})\\
&\le d(\Upsilon', \Upsilon^*) + (\alpha_{\ell,k} + \alpha_{\ell,k'})/N_W - (2 - 1/N_W)(2/3)\\
&\le d(\Upsilon', \Upsilon^*) + 2/m^2\delta^2 - (2/3)\\
&< d(\Upsilon', \Upsilon^*).
\end{aligned}$$

By a repeated application of these cases, we construct a path joining Υ to some Υ^* that is in the inner domain of $\Sigma_{\rho,\zeta}$. As mentioned, we can also construct such a path joining Ψ to some Ψ^* in the inner domain, and then reverse all of the edges. Following a brief analysis of the flow for this first phase in the next section, we show how to join pairs of elements in the inner domain.

5.2.2. Analysis of flow for Phase 1. The definition of $\alpha_{i,j}$ defines an equivalence class on the set $\Sigma_{\rho,\zeta}$, where $\Phi \equiv \Phi'$ if and only if $\Phi[i,j] = \Phi'[i,j] \bmod (\alpha_{i,j}+1)$ for every $i \in [m-1]$, $j \in [\delta - 1]$ (i.e., all the remainders $R[i,j]$ and $R'[i,j]$ are the same).

Note that by definition of the $\alpha_{i,j}$ values, and since $\Phi[i,j] \le \{\rho_i, \zeta_j\}$ for all $i \in [m-1]$, $j \in [\delta-1]$ for every $\Phi \in \Sigma_{\rho,\zeta}$, any equivalence class contains at most $(m^2\delta^2)^{m\delta}$ contingency tables (there are at most $m^2\delta^2$ choices for each $Q[i,j]$ with $i \in [m-1]$, $j \in [\delta-1]$). Therefore each equivalence class contains a constant number of contingency tables.

The routing scheme given in Cases (a)–(d) defines a path $\Upsilon = \Phi_0, \Phi_1, \ldots, \Phi_t = \Upsilon^*$ from Υ to Υ^* for every $\Upsilon \in \Sigma_{\rho,\zeta}$. We know Φ_h lies in the same equivalence class as Υ and Υ^* for every h. By our analysis of Cases (a)–(d), we know that for every $h \ge 0$, either $d(\Phi_{h+1}, \Upsilon^*) < d(\Phi_h, \Upsilon^*)$ or $d(\Phi_{h+2}, \Upsilon^*) < d(\Phi_h, \Upsilon^*)$. This means we can define a subsequence of the path $\{\Phi_h\}$ such that (i) the subsequence contains at least every second element of $\{\Phi_h\}$ and (ii) no contingency table ever appears twice in the subsequence. Thus the length of the path from Υ to Υ^* is at most $2(m^2\delta^2)^{m\delta}$.

To analyze the amount of flow from Phase 1 that may pass through $\Phi \in \Sigma_{\rho,\zeta}$, we rely on the fact that for any $\Upsilon \in \Sigma_{\rho,\zeta}$ the path from Υ to Υ^* lies in the equivalence class of Υ. Therefore, for any fixed Ψ, there are at most $(m^2\delta^2)^{m\delta}$ different contingency tables Υ that may pass through Φ on the way to Υ^*. Also, by our bound on the length of the path, we know that for any fixed Ψ and Υ, Φ may occur at most $(m^2\delta^2)^{m\delta}$ times on the path from Υ to Υ^*.

Putting all of this information together, we see that for any fixed Ψ, the flow through any Φ during Phase 1 is at most $(m^2\delta^2)^{2m\delta}$. This implies that the total flow through Φ during Phase 1 is at most $|\Sigma_{\rho,\zeta}|\,(m^2\delta^2)^{2m\delta}$.

5.2.3. Phase 2. In this phase we describe how to route Υ^* to Ψ^*, i.e., route flow between any pair of elements in the inner domain of $\Sigma_{\rho,\zeta}$. We know that

$$\Upsilon^*[i,j] = Q^*[i,j](\alpha_{i,j}+1) + R_\Upsilon[i,j],$$
$$\Psi^*[i,j] = Q^*[i,j](\alpha_{i,j}+1) + R_\Psi[i,j]$$

for all $i \in [m-1]$, $j \in [m-1]$, where $Q^*[i,j]$ was defined in (19).

In this phase we route Υ^* to Ψ^* in $(m-1)(\delta-1)$ steps, by "fixing" one remainder at a time. The key to this approach is part (i) of Lemma 9, which shows that for *any* remainders $R[i,j]$ satisfying $R[i,j] \le \alpha_{i,j}$ for $i \in [m-1]$, $j \in [\delta-1]$, if Φ is defined by

$$\Phi[i,j] = Q^*[i,j](\alpha_{i,j}+1) + R[i,j]$$

for $i \in [m-1]$, $j \in [\delta-1]$, then $\Phi \in \Sigma_{\rho,\zeta}$ (where $\Phi[m,j]$ and $\Phi[i,\delta]$ are defined as in Lemma 9 to satisfy the row and column sums).

Suppose we order the "boxes" of the $m \times \delta$ contingency tables in lexicographic order: $(1,1),(1,2),\ldots,(1,\delta-1),(2,1),\ldots,(m-1,1),\ldots,(m-1,\delta-1)$. Letting $h = (h_1,h_2)$ denote any point in this lexicographic order, we use h^+ to denote the successor to h in this ordering.

Then we can define a series of tables

$$\Upsilon^* = \Phi_{(1,1)}, \Phi_{(1,2)}, \ldots, \Phi_h, \ldots, \Phi_{(m-1,\delta-1)} = \Psi^*$$

by

$$\Phi_h[i,j] = \begin{cases} \Psi^*[i,j] & \text{if } (i,j) \le h \text{ (and } i \ne m \text{ and } j \ne \delta), \\ \Upsilon^*[i,j] & \text{if } (i,j) \ge h^+ \text{ (and } i \ne m \text{ and } j \ne \delta), \\ \rho_i - \sum_{j=1}^{\delta-1} \Phi_h[i,j] & \text{if } j = \delta, \\ \zeta_j - \sum_{i=1}^{m-1} \Phi_h[i,j] & \text{if } i = m. \end{cases}$$

By part (i) of Lemma 9 we know that $\Phi_h \in \Sigma_{\rho,\zeta}$ for all h, and therefore $\Phi_h \to \Phi_{h^+}$ is a transition of $\mathcal{M}_{2\times 2}$. ("Fixing" the (i,j) remainder, i.e., changing $R_\Upsilon[i,j]$ into $R_\Psi[i,j]$, uses a transition of $\mathcal{M}_{2\times 2}$ that involves the four "boxes" (i,j), (i,δ), (m,j), and (m,δ).)

Note that if we define a dual table $\bar{\Phi}_h$ by

$$\bar{\Phi}_h[i,j] = \begin{cases} \Upsilon^*[i,j] & \text{if } (i,j) \le h \text{ (and } i \ne m \text{ and } j \ne \delta), \\ \Psi^*[i,j] & \text{if } (i,j) \ge h^+ \text{ (and } i \ne m \text{ and } j \ne \delta), \\ \rho_i - \sum_{j=1}^{\delta-1} \bar{\Phi}_h[i,j] & \text{if } j = \delta, \\ \zeta_j - \sum_{i=1}^{m-1} \bar{\Phi}_h[i,j] & \text{if } i = m, \end{cases}$$

then Lemma 9(i) also tells us that $\bar{\Phi}_h \in \Sigma_{\rho,\zeta}$.

Therefore we have a path of length $(m-1)(\delta-1)$ connecting Υ^* to Ψ^* in $\Sigma_{\rho,\zeta}$.

5.2.4. Analysis of flow for Phase 2. We bound the amount of flow from Phase 2 that can pass through any $\Phi \in \Sigma_{\rho,\zeta}$. Similar to section 4.2, we do this using an encoding. Suppose that we are given

(1) a pair of indices $h = (h_1, h_2)$ specifying that Φ occurs as Φ_h on the path from Υ^* to Ψ^* during phase 2;
(2) the dual contingency table $\bar{\Phi}_h$;
(3) the integers $Q_\Upsilon[i,j]$ for all $i \in [m]$, $j \in [\delta]$;
(4) the integers $Q_\Psi[i,j]$ for all $i \in [m]$, $j \in [\delta]$.

Then we can construct the original pair of tables Υ and Ψ exactly. The information in (1) and (2) first allows us to reconstruct Υ^* and Ψ^*. Then using Υ^*, we may reconstruct $R_\Upsilon[i,j]$ for all i,j since we know (or may compute) $Q^*[i,j]$. Knowing $R_\Upsilon[i,j]$, together with the information in (3), we can find Υ exactly, and in a like manner we can reconstruct Ψ. There are at most $(m^2\delta^2)^{m\delta}$ possible values for the $Q_\Upsilon[i,j]$, and the same number for the $Q_\Psi[i,j]$.

Therefore, for any $\Phi \in \Sigma_{\rho,\zeta}$, the total amount of flow that may pass through Φ during Phase 2 is at most $(m\delta)(m^2\delta^2)^{2m\delta}|\Sigma_{\rho,\zeta}|$.

5.2.5. Finishing up. Combining Phases 1 and 2, the length of any (Υ, Ψ)-path is at most $4(m\delta)^{2m\delta} + (m-1)(\delta-1)$. The total amount of flow that can pass through any $\Phi \in \Sigma_{\rho,\zeta}$ is at most $|\Sigma_{\rho,\zeta}|(2(m\delta)^{4m\delta} + m\delta(m^2\delta^2)^{2m\delta})$.

This establishes the condition on the flow f^* that was cited in section 5.1. As explained in that section we can alter this flow to get the new flow f, letting us establish Theorem 8, proving rapid mixing of $\mathcal{M}_{2\times 2}$ on the set of $m \times n$ contingency tables $\Sigma_{r,c}$.

REFERENCES

[1] A. I. Barvinok, *A polynomial-time algorithm for counting integral points in polyhedra when the dimension is fixed*, Math. Oper. Res., 19 (1994), pp. 769–779.

[2] F. K. R. Chung, R. L. Graham, and S.-T. Yau, *On sampling with Markov chains*, Random Structures Algorithms, 9 (1996), pp. 55–77.

[3] M. Cryan and M. Dyer, *A polynomial time algorithm to approximately count contingency tables when the number of rows is constant*, J. Comput. System Sci., 67 (2003), pp. 291–310.

[4] J. A. De Loera and B. Sturmfels, *Algebraic unimodular counting*, Math. Program., 96 (2003), pp. 183–203.

[5] P. Diaconis and B. Efron, *Testing for independence in a two-way table: New interpretations of the chi-square statistic (with discussion)*, Ann. Statist., 13 (1995), pp. 845–913.

[6] P. Diaconis and A. Gangolli, *Rectangular arrays with fixed margins*, in Discrete Probability and Algorithms, IMA Vol. Math. Appl. 72, D. Aldous, P. P. Varaiya, J. Spencer, and J. M. Steele, eds., Springer-Verlag, New York, 1995, pp. 15–41.

[7] P. Diaconis and L. Saloff-Coste, *Comparison theorems for reversible Markov chains*, Ann. Appl. Probab., 3 (1993), pp. 696–730.

[8] P. Diaconis and L. Saloff-Coste, *Random Walk on Contingency Tables with Fixed Row and Column Sums*, Technical report, Department of Mathematics, Harvard University, Cambridge, MA, 1995.

[9] P. Diaconis and D. Stroock, *Geometric bounds for eigenvalues of Markov chains*, Ann. Appl. Probab., 1 (1991), pp. 36–61.

[10] M. Dyer, *Approximate counting by dynamic programming*, in Proceedings of the 35th ACM Symposium on Theory of Computing, San Diego, 2003, pp. 693–699.

[11] M. Dyer and C. Greenhill, *Polynomial-time counting and sampling of two-rowed contingency tables*, Theoret. Comput. Sci., 246 (2000), pp. 265–278.

[12] M. Dyer, R. Kannan, and J. Mount, *Sampling contingency tables*, Random Structures Algorithms, 10 (1997), pp. 487–506.

[13] G. R. Grimmett and D. R. Stirzaker, *Probability and Random Processes*, Oxford University Press, Oxford, UK, 1992.

[14] V. S. Grinberg and S. V. Sevast'yanov, *Value of the Steinitz constant*, Funktsional. Anal. i Prilozhen., 14 (1980), pp. 56–57.
[15] D. Hernek, *Random generation of* $2 \times n$ *contingency tables*, Random Structures Algorithms, 13 (1998), pp. 71–79.
[16] M. Jerrum and A. Sinclair, *The Markov chain Monte Carlo method: An approach to approximate counting and integration*, in Approximation Algorithms for NP-Hard Problems, D. S. Hochbaum, ed., PWS, Boston, 1996, pp. 482–520.
[17] M. R. Jerrum, L. G. Valiant, and V. V. Vazirani, *Random generation of combinatorial structures from a uniform distribution*, Theoret. Comput. Sci., 43 (1986), pp. 169–188.
[18] B. Morris, *Improved bounds for sampling contingency tables*, in 3rd International Workshop on Randomization and Approximation Techniques in Computer Science, Lecture Notes in Comput. Sci. 1671, Springer-Verlag, New York, 1999, pp. 121–129.
[19] B. Morris, *Random Walks in Convex Sets*, Ph.D. thesis, Department of Statistics, University of California, Berkeley, CA, 2000.
[20] B. Morris and A. Sinclair, *Random walks on truncated cubes and sampling* 0-1 *knapsack solutions*, SIAM J. Comput., 34 (2004), pp. 195–226.
[21] J. Mount, *Application of Convex Sampling to Optimization and Contingency Table Generation*, Ph.D. thesis, Technical report CMU-CS-95-152, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1995.
[22] I. Pak, *On sampling integer points in polyhedra*, in Foundations of Computational Mathematics, World Scientific, River Edge, NJ, 2002, pp. 319–324.
[23] D. Randall and P. Tetali, *Analyzing Glauber dynamics by comparison of Markov chains*, J. Math. Phys., 41 (2000), pp. 1598–1615.
[24] A. J. Sinclair, *Improved bounds for mixing rates of Markov chains and multicommodity flow*, Combin. Probab. Comput., 1 (1992), pp. 351–370.
[25] E. Steinitz, *Bedingt konvergente Reihen und konvexe Systeme*, J. Reine Angew. Math., 143 (1913), pp. 128–175.
[26] V. V. Vazirani, *Approximation Algorithms*, Springer-Verlag, New York, 1999.
[27] E. Vigoda, *Improved bounds for sampling colorings*, J. Math. Phys., 41 (2000), pp. 1555–1569.

SIAM J. COMPUT.
Vol. 36, No. 1, pp. 279–280

ERRATUM: DISTRIBUTED ANONYMOUS MOBILE ROBOTS: FORMATION OF GEOMETRIC PATTERNS*

ICHIRO SUZUKI† AND MASAFUMI YAMASHITA‡

Abstract. In this note we make a minor correction to a scheme for robots to broadcast their private information. All major results of the paper [I. Suzuki and M. Yamashita, *SIAM J. Comput.*, 28 (1999), pp. 1347–1363] hold with this correction.

Key words. anonymous robots, broadcast

AMS subject classification. 68Q99

DOI. 10.1137/050631562

1. Correction. Algorithms $\psi_{f-point(2)}$ in section 3 and $\psi_{getview}$ in section 4 of [1] use the following technique for all robots r_i to simultaneously "broadcast" to the other robots a privately chosen directed line ℓ_i (e.g., the positive x-axis of its local coordinate system Z_i). As outlined in the paragraph that follows the proof of Theorem 3.4, the basic idea is that each r_i moves repeatedly along ℓ_i in a fixed direction each time it becomes active until, for each $j \neq i$, it has seen r_j at two or more distinct locations. Robot r_i can then figure out ℓ_j based on two distinct locations that r_j has occupied. In an effort to ensure at the same time that r_j has also seen r_i at two or more distinct locations (so that it can figure out ℓ_i), we made an incorrect claim that if r_i has observed r_j at three or more distinct locations, then r_j has observed r_i at two or more distinct locations. This claim must be replaced by the following.

PROPOSITION 1. *For any integer $m \geq 1$, if r_i has seen r_j at $2m$ distinct locations in the time interval $[0, t]$, then r_j has seen r_i at m or more distinct locations in $[0, t)$.*

Proof. Suppose r_i becomes active at times t_1 and t_2, $t_1 < t_2$, and observes r_j at two distinct locations. Since r_j occupies distinct locations at t_1 and t_2, r_j must become active in $[t_1, t_2)$ and observe r_i at a location on the line segment $\overline{p_1 p_2}$, where p_1 and p_2 are the locations of r_i on line ℓ_i at t_1 and t_2, respectively. This means that r_j observes r_i at a distinct location each time r_i observes r_j at two distinct locations, since r_i moves along ℓ_i in a fixed direction each time it becomes active. □

Therefore, to ensure that r_j has seen r_i at two distinct locations, r_i must continue to move along ℓ_i until it has seen r_j at four or more distinct locations. However, if we allow r_i to simply stop moving as soon as it has observed r_j at four or more distinct locations, then r_j may not be able to observe r_i at four or more distinct locations. This means that r_j may never finish the broadcast.

Fortunately, broadcasting a line is usually a preliminary step that precedes a main task. In the case of $\psi_{f-point(2)}$, the ultimate goal of r_i and r_j is to move to the midpoint p of their initial positions. (Because of the way ℓ_i and ℓ_j are chosen in $\psi_{f-point(2)}$, both r_i and r_j can compute p from ℓ_i and ℓ_j.) Note that when r_i has seen r_j at four or more distinct locations, r_i knows not only ℓ_j, but also that r_j knows ℓ_i (because by Proposition 1 r_j has seen r_i at two or more distinct locations). Thus r_i

*Received by the editors May 13, 2005; accepted for publication (in revised form) February 22, 2006; published electronically June 19, 2006.
http://www.siam.org/journals/sicomp/36-1/63156.html
†EECS, University of Wisconsin-Milwaukee, Milwaukee, WI 53201 (suzuki@cs.uwm.edu).
‡Department of Computer Science and Communications Engineering, Kyushu University, Fukuoka 812-8581, Japan (mak@csce.kyushu-u.ac.jp).

can now safely quit broadcasting ℓ_i and move to (or toward) p, regardless of whether r_j has seen r_i at four or more distinct locations. By moving to a point not on ℓ_i, r_i effectively "announces" to r_j that it now knows ℓ_j. Robot r_j eventually observes r_i at a location not on ℓ_i and learns that it can quit broadcasting ℓ_j as well and proceed to p.

Based on the above discussion, r_i can use the following scheme to broadcast ℓ_i to all other robots:

1. r_i moves along ℓ_i in a fixed direction each time it becomes active until, for each $j \neq i$, either
   (a) it has seen r_j at four or more distinct locations, or
   (b) it observes that r_j is at a location not on ℓ_j. (Note that r_i knows ℓ_j by the time this case occurs.)
2. Then it moves to a point not on ℓ_i.

Algorithm $\psi_{f-point(2)}$, the proof of Theorem 3.3, and the paragraph that follows the proof of Theorem 3.4 should be revised accordingly.

Algorithm $\psi_{getview}$ must be modified using a similar idea. In $\psi_{getview}$, each robot r_i, initially located at the origin o_i of its local coordinate system Z_i, broadcasts three lines: its x-axis, y-axis, and line L_i through o_i in direction $f(d_i)$, where d_i is the minimum distance between any two initial positions of the robots, and for $x > 0$, $f(x) = (1 - 1/2^x) \times 90^\circ$ is a monotonically increasing function with range $(0^\circ, 90^\circ)$. (This function replaces $f(x) = (1 - 1/2^x) \times 360^\circ$ used in the paper. Both $f(d_i)$ and d_i are measured in terms of Z_i.) Note that the orientations of the three lines are all distinct.

Each robot r_i first broadcasts its x-axis by moving along it in the positive direction. When r_i knows that all other robots know r_i's x-axis and its orientation, i.e., for each $j \neq i$, either

(a) r_i has seen r_j at four or more distinct locations, or
(b) r_i observes that r_j has changed the direction of its motion (r_i knows the x-axis of r_j by the time this occurs),

it returns straight to o_i and starts broadcasting its y-axis by moving along it in the positive direction (thereby changing its direction of motion and announcing to others that it now knows the x-axes of all other robots). Eventually all robots finish broadcasting their x-axes and start broadcasting their y-axes. r_i ends the broadcast of its y-axis when it knows that all other robots know its y-axis and its orientation, returns straight to o_i, and starts broadcasting line L_i by moving in direction $f(d_i)$ (thereby changing its direction of motion again, announcing to others that it knows the y-axes of all other robots). Eventually all robots r_j start broadcasting their lines L_j. Once again, r_i terminates this broadcast when it knows that all other robots know L_i, and returns to its initial position o_i. By returning to o_i, r_i announces to others that it has finished the broadcast of L_i.

REFERENCE

[1] I. Suzuki and M. Yamashita, *Distributed anonymous mobile robots: Formation of geometric patterns*, SIAM J. Comput., 28 (1999), pp. 1347–1363.

SIAM J. COMPUT.
Vol. 36, No. 2, pp. 281–309

DOMINATING SETS IN PLANAR GRAPHS: BRANCH-WIDTH AND EXPONENTIAL SPEED-UP*

FEDOR V. FOMIN† AND DIMITRIOS M. THILIKOS‡

Abstract. We introduce a new approach to design parameterized algorithms on planar graphs which builds on the seminal results of Robertson and Seymour on graph minors. Graph minors provide a list of powerful theoretical results and tools. However, the widespread opinion in the graph algorithms community about this theory is that it is of mainly theoretical importance. In this paper we show how deep min-max and duality theorems from graph minors can be used to obtain exponential speed-up to many known practical algorithms for different domination problems. Our use of branch-width instead of the usual tree-width allows us to obtain much faster algorithms. By using this approach, we show that the k-dominating set problem on planar graphs can be solved in time $O(2^{15.13\sqrt{k}} + n^3)$.

Key words. branch-width, tree-width, dominating set, planar graph, fixed-parameter algorithm

AMS subject classifications. 05C35, 05C69, 05C83, 05C85, 68R10

DOI. 10.1137/S0097539702419649

1. Introduction.

DOMINATING SET is a classic NP-complete graph problem which fits into the broader class of *domination* and *covering* problems on which hundreds of papers have been written. (The book of Haynes, Hedetniemi, and Slater [32] is a nice source for further references on the dominating set problem.) The problem PLANAR DOMINATING SET asks, given a planar graph G and a positive k, whether G has a dominating set of size at most k. It is well known that the PLANAR DOMINATING SET (as well as several variants of it) is NP-complete. In this paper we design exact *fixed-parameter* algorithms (which run fast provided that the parameter k is small). The theory of fixed-parameter algorithms and parameterized complexity has been thoroughly developed over the past few years; see, e.g., [1, 3, 4, 8, 12, 13, 21, 23, 24].

The last six years have seen dramatic developments and improvements to the design of subexponential algorithms with running times of $2^{O(\sqrt{k})}n^{O(1)}$ for different planar graph problems; see, e.g., [1, 4, 8, 9, 13, 14, 22, 31, 35]. For example, the first algorithm for the PLANAR DOMINATING SET appeared in [2], with running time $O(8^k n)$. The first algorithm with a *sublinear* exponent was given by Alber et al. in [1] and its running time was $O(2^{69.98\sqrt{k}}n)$. A time $O(2^{49.88\sqrt{k}}n)$ algorithm was obtained in [4], and Kanj and Perković [35] announced an algorithm of running time $O(2^{27\sqrt{k}}n)$.

A common method for solving PLANAR DOMINATING SET is to prove that every planar graph with a dominating set of size at most k has tree-width at most $c\sqrt{k}$, where c is a constant. With some work (sometimes very technical) a tree decomposition of width at most $c\sqrt{k} + O(1)$ is constructed, and standard dynamic programming techniques on graphs of bounded tree-width are implemented. Currently, the fastest

*Received by the editors December 17, 2002; accepted for publication (in revised form) December 6, 2005; published electronically June 19, 2006. An extended abstract of the results of this paper appeared in [25].

http://www.siam.org/journals/sicomp/36-2/41964.html

†Department of Informatics, University of Bergen, N-5020 Bergen, Norway (fomin@ii.uib.no). The work of this author was supported by the Norwegian Research Council.

‡Department of Mathematics, National and Capodistrian University of Athens Panepistimioupolis, GR-15784, Athens, Greece (sedthilk@lsi.upc.edu). The work of this author was supported by the Spanish CICYT project TIN-2004-07925 (GRAMMARS).

dynamic programming algorithm for a dominating set on graphs of tree-width at most t runs in $O(2^{2t}n)$ steps and was given by Alber et al. in [1]. This implies an $O(2^{2c\sqrt{k}}n)$ step algorithm for the Planar Dominating Set. Let

$$c_{\rm tw} = \min\{c \mid \text{if } G \text{ is planar and dominated by } k \text{ vertices, then } \mathbf{tw}(G) \leq c\sqrt{k} + O(1)\}.$$

The challenge in this approach is that a small bound for $c_{\rm tw}$ is required for most practical applications. The first bound for $c_{\rm tw}$ appeared in [1] and was $c_{\rm tw} < 6\sqrt{34} = 34.98$, while the next improvement was given by Kanj and Perković in [35], who proved that $c_{\rm tw} < 16.5$.

The main tool of this paper is the branch-width of a graph. Branch-width was introduced by Robertson and Seymour in their graph minors series of papers several years after tree-width. These parameters are rather close, but surprisingly many theorems of the graph minors series are easier to prove when one uses branch-width instead of tree-width. Nice examples of the use of branch-width in proof techniques can be found in [38] and [39]. Another powerful property of branch-width is that it can be naturally generalized for hypergraphs and matroids. A good example of generalization of Robertson and Seymour theory for matroids by using branch-width is the paper by Geelen, Gerards, and Whittle [29]. Algorithms for problems expressible in monadic second-order logic on matroids of bounded branch-width are discussed by Hliněný [34]. Alekhnovich and Razborov [5] use branch-width of hypergraphs to design algorithms for SAT.

From a practical point of view, branch-width is also promising. For some problems, branch-width is more suitable for actual implementations. Cook and Seymour [10, 11] used branch decompositions to solve the ring routing problem, related to the design of reliable cost-effective SONET networks and to solving TSP (see also [7, 19]). In theory, there is not a big difference between tree-width and branch-width based algorithms. However in practice, branch-width is sometimes easier to use. The question due to Bodlaender (private communication) is the following: Are there examples where the constant factors for branch-width algorithms are significantly smaller than for their tree-width counterparts? This paper is partially motivated[1] by this question.

Our results. In this paper we introduce a new approach for solving the Planar Dominating Set problem. Our approach is based on branch-width and provides an algorithm of running time $O(2^{15.13\sqrt{k}} + n^3)$, which is a significant step toward a practical algorithm. Instead of constructing a tree decomposition and proving that the width of the obtained tree decomposition is upper bounded by $c\sqrt{k}$, we prove a combinatorial result relating the branch-width with the domination number of a planar graph. The proof of the combinatorial bounds is complicated and is based on nice properties of branch-width, which follow from deep results of the graph minors series.

Our proof is not constructive, in the sense that it cannot be turned into a polynomial algorithm that *constructs* the corresponding branch decomposition. Fortunately, there is a well-known algorithm due to Seymour and Thomas for computing an optimal branch decomposition of a planar graph in $O(n^4)$ steps. We stress that this algorithm does not have the so-called enormous hidden constants and is really practical.

[1] One of the challenges that appeared during the workshop "Optimization Problems, Graph Classes and Width Parameters" (Centre de Recerca Matemàtica, Bellaterra, Spain, November 15–17, 2001), was the following question: *Is it possible, using bounded branch-width instead of bounded tree-width, to obtain more efficient solutions for* Planar Dominating Set *and other parameterized problems?*

(We refer to the work of Hicks [33] on implementations of the Seymour and Thomas algorithm; see also [30] for a recent algorithm that runs in $O(n^3)$ steps.)

Our main combinatorial result is that for every planar graph G with a dominating set of size $\leq k$, the branch-width of G is at most $3\sqrt{4.5}\sqrt{k} < 6.364\sqrt{k}$. We combine this bound with the following algorithmic results: (i) the algorithm of Seymour and Thomas for planar branch-width, (ii) the results of Alber, Fellows, and Niedermeier [3] on a linear problem kernel for Planar Dominating Set, and (iii) a new dynamic programming algorithm for solving the dominating set problem on graphs of bounded branch-width (see subsection 4.2). As a result, we obtain an algorithm of running time $O(2^{15.13\sqrt{k}} + n^3)$.

According to Robertson and Seymour [36], for any graph G with at least three edges, the tree-width of G is always bounded by $\frac{3}{2}$ times its branch-width. This result, in combination with our bound, implies that $c_{\mathrm{tw}} < 9.546$. To our knowledge, this gives an improvement on any other bound for the tree-width of planar graphs dominated by k vertices.

Organization of the paper. In section 2, we give basic definitions and state some known theorems. We also present how a theorem of Robertson, Seymour, and Thomas can be directly used to prove that every planar graph with a dominating set of size $\leq k$ has branch-width at most $\leq 12\sqrt{k} + 9$. This observation (combined with the results discussed in section 4) already implies an algorithm for the Planar Dominating Set problem with running time $O(2^{28.56\sqrt{k}} + n^3)$, where n is the number of vertices of G. This is already a strong improvement (for large k) on the result of Alber et al. in [1] and is close to the running time $O(2^{27\sqrt{k}}n)$ of the algorithm of Kanj and Perković in [35].

Section 3 is devoted to the proof of Theorem 3.22, the main combinatorial result of the paper. The proof of this result is complicated, and we split it into several parts. In subsection 3.1, we give technical results about branch decompositions. These results are based on the powerful theorem of Robertson and Seymour on the branch-width of dual graphs. We emphasize that these results are crucial for our proof. In subsection 3.2, we define the notion of the *extension* of a graph and prove that the branch-width of an extension is at most three times the branch-width of the original graph. In section 3.3 we introduce the notion of *nicely dominated graphs*, which is a suitable "normalization" of the structure of the dominated planar graphs. In subsection 3.4, we explain how nicely dominated graphs can be gradually decomposed into simpler ones so that the branch-width of the original graph is bounded by the branch-width of some "enhanced version" of the simpler ones. In subsection 3.5 we introduce the *prime nicely dominated graphs* as those that are "the simplest possible" with respect to the decomposition of subsection 3.4. In subsection 3.6, we prove that any prime nicely dominated graph G is "contained" in the extension of a simpler planar graph denoted as $\mathbf{red}(G)$. In subsection 3.7 we use this fact along with the results of subsections 3.2, 3.4, and 3.6 to prove that $\mathbf{bw}(G) \leq 3 \cdot \mathbf{bw}(\mathbf{red}(G))$. By its construction, all the vertices of $\mathbf{red}(G)$ are vertices of the dominating set D. The result follows because, according to [28], $\mathbf{bw}(\mathbf{red}(G)) \leq \sqrt{4.5 \cdot |D|}$.

Section 4 contains discussions on algorithmic consequences of the combinatorial result. Subsection 4.1 describes the general algorithmic scheme that we follow. Subsection 4.2 contains a dynamic programming algorithm for the solving dominating set problem on graphs of branch-width $\leq \ell$ and m edges, in time $O(3^{1.5\cdot\ell}m)$.

In section 5 we discuss the optimality of our results (subsection 5.1) and provide some concluding remarks and open problems (subsection 5.2).

2. Definitions and preliminary results. Let G be a graph with vertex set $V(G)$ and edge set $E(G)$. For every nonempty $W \subseteq V(G)$, the subgraph of G induced by W is denoted by $G[W]$. A vertex $v \in V(G)$ of a connected graph G is called a *cut vertex* if the graph $G - \{v\}$ is not connected. A connected graph on at least three vertices without a cut vertex is called 2-*connected*. Maximal 2-connected subgraphs of a graph G or induced edges whose two endpoints are cut vertices are called 2-*connected components*.

Let Σ be a sphere. By Σ-*plane* graph G we mean a planar graph G with the vertex set $V(G)$ and the edge set $E(G)$ drawn in Σ. To simplify notations, we usually do not distinguish between a vertex of the graph and the point of Σ used in the drawing to represent the vertex, or between an edge and the open line segment representing it. If $\Delta \subseteq \Sigma$, then $\overline{\Delta}$ denotes the *closure* of Δ, and the boundary of Δ is $\widehat{\Delta} = \overline{\Delta} \cap \overline{\Sigma - \Delta}$. We denote the set of the faces of the drawing by $R(G)$. (Every face is an open set.) An edge e (a vertex v) is incident to a face r if $e \subseteq \bar{r}$ ($v \subseteq \bar{r}$). We do not distinguish between a boundary of a face and the subgraph of the drawing induced by edges incident to the face. The *length* of a face r is the number of edges incident to r. $\Delta \subseteq \Sigma$ is an open disc if it is homeomorphic to $\{(x, y) : x^2 + y^2 < 1\}$. Let C be a cycle in a Σ-plane graph G. By the Jordan curve theorem, C bounds exactly two discs. For a vertex $x \in V(G)$, we call a disc Δ bounded by C x-*avoiding* if $x \notin \overline{\Delta}$. We call a face $r \in R(G)$ *square face* if $\widehat{r}$ is a cycle of length four.

A set $D \subseteq V(G)$ is a *dominating set* in a graph G if every vertex in $V(G) - D$ is adjacent to a vertex in D. Graph G is D-*dominated* if D is a dominating set in G.

For a hypergraph $\mathcal{G}$ we denote by $V(\mathcal{G})$ its vertex (ground) set and by $E(\mathcal{G})$ the set of its hyperedges. A *branch decomposition* of a hypergraph $\mathcal{G}$ is a pair (T, τ), where T is a tree with vertices of degree one or three and τ is a bijection from $E(\mathcal{G})$ to the set of leaves of T. The *order function* $\omega : E(T) \to 2^{V(\mathcal{G})}$ of a branch decomposition maps every edge e of T to a subset of vertices $\omega(e) \subseteq V(\mathcal{G})$ as follows. The set $\omega(e)$ consists of all vertices $v \in V(\mathcal{G})$ such that there exist edges $f_1, f_2 \in E(\mathcal{G})$ with $v \in f_1 \cap f_2$, and such that the leaves $\tau(f_1)$, $\tau(f_2)$ are in different components of $T - \{e\}$.

The *width* of (T, τ) is equal to $\max_{e \in E(T)} |\omega(e)|$, and the *branch-width* of $\mathcal{G}$, $\mathbf{bw}(\mathcal{G})$, is the minimum width over all branch decompositions of $\mathcal{G}$.

Given an edge $e = \{x, y\}$ of a graph G, the graph G/e is obtained from G by contracting the edge e; that is, to get G/e we identify the vertices x and y and remove all loops and duplicate edges. A graph H obtained by a sequence of edge contractions is said to be a *contraction* of G. H is a *minor* of G if H is a subgraph of a contraction of G. We use the notation $H \preceq G$ (resp., $H \preceq_c G$) when H is a minor (a contraction) of G. It is well known that $H \preceq G$ or $H \preceq_c G$ implies $\mathbf{bw}(H) \leq \mathbf{bw}(G)$. Moreover, the fact that G has a dominating set of size k and $H \preceq_c G$ imply that H has a dominating set of size $\leq k$ (which is not true for $H \preceq G$).

For planar graphs the branch-width can be bounded in terms of the domination number by making use of the following result of Robertson, Seymour, and Thomas (Theorems 4.3 in [36] and 6.3 in [38]).

THEOREM 2.1 (see [38]). *Let $k \geq 1$ be an integer. Every planar graph with no (k, k)-grid as a minor has branch-width $\leq 4k - 3$.*

To give an idea on how results from graph minors can be used on the study of dominating sets in planar graphs, we present the following simple consequence of Theorem 2.1.

LEMMA 2.2. *Let G be a planar graph with a dominating set of size $\leq k$. Then $\mathbf{bw}(G) \leq 12\sqrt{k} + 9$.*

Proof. Suppose that $\mathbf{bw}(G) > 12\sqrt{k}+9$. By Theorem 2.1, there exists a sequence of edge contractions or edge/vertex removals reducing G to a (ρ,ρ)-grid where $\rho = 3\sqrt{k}+3$. We apply to G only the contractions from this sequence and call the resulting graph J. J contains a (ρ,ρ)-grid as a subgraph. As $J \preceq_c G$, J also has a dominating set D of size $\leq k$. A vertex in D cannot dominate more than nine internal vertices of the (ρ,ρ)-grid. Therefore, $k \geq (\rho-2)^2/9$, which implies $\rho \leq 3\sqrt{k}+2 = \rho-1$, a contradiction. □

In the remaining part of the paper we show how the above upper bound for the branch-width of a planar graph in terms of its dominating set number can be strongly improved. Our results will use as a basic ingredient the following theorem, which is a direct consequence of the Robertson and Seymour min-max Theorem 4.3 in [36] relating tangles and branch-width and Theorem 6.6 in [37] establishing relations between tangles of dual graphs. Since the result is not mentioned explicitly in the articles of Robertson and Seymour, we provide here a short explanation of how it can be derived.

We denote as K_2^2 the graph consisting of two vertices connected by a double edge. Notice that K_2^2 is a dual of itself; therefore, if G contains K_2^2 as a minor, then its dual G^* also contains K_2^2 as a minor.

THEOREM 2.3. *Let G be a Σ-plane graph that contains K_2^2 as a minor and let G^d be its dual. Then $\mathbf{bw}(G) = \mathbf{bw}(G^d)$.*

Proof. A *separation* of a graph G is a pair (A,B) of subgraphs with $A \cup B = G$ and $E(A \cap B) = \emptyset$, and its *order* is $|V(A \cap B)|$. A *tangle of order* $\theta \geq 1$ is a set $\mathcal{T}$ of separations of G, each of order less than θ, such that

1. for every separation (A,B) of G of order less than θ, $\mathcal{T}$ contains one of (A,B) and (B,A);
2. if $(A_1,B_1),(A_2,B_2),(A_3,B_3) \in \mathcal{T}$, then $A_1 \cup A_2 \cup A_3 \neq G$; and
3. if $(A,B) \in \mathcal{T}$, then $V(A) \neq V(G)$.

The *tangle number* $\theta(G)$ of G is the maximum order of tangles in G. By the result of Robertson and Seymour [36, Theorem 4.3], for any graph G of branch-width at least two, $\theta(G) = \mathbf{bw}(G)$. Since $\mathbf{bw}(K_2^2) = 2$ and $K_2^2 \preceq G$, we have that $\theta(G) = \mathbf{bw}(G)$. By similar arguments, $\theta(G^d) = \mathbf{bw}(G^d)$.

Let G be a graph 2-cell embedded in a connected surface Σ. A subset of Σ meeting the drawing only at vertices of G is called *G-normal*. The *length* of a G-normal arc is the number of vertices it meets. A tangle $\mathcal{T}$ of order θ is *respectful* if, for every homeomorphic to a circle G-normal arc N in Σ of length less than θ, there is a closed disk $\Delta \subseteq \Sigma$ with $\widehat{\Delta} = N$ such that the separation $(G \cap \Delta, G \cap \overline{\Sigma - \Delta}) \in \mathcal{T}$. By the first tangle property, every tangle $\mathcal{T}$ of a graph embedded in a *sphere* is respectful.

By [37, Theorem 6.6], for every 2-cell embedded graph G on a connected surface Σ, G has a respectful tangle of order θ if and only if its dual G^d does. This implies that $\theta(G) = \theta(G^d)$ and the theorem follows. □

For our bounds, we need an upper bound on the size of branch-width of a planar graph in terms of its size. The best published bound for the branch-width that we were able to find in the literature is $\mathbf{bw}(G) \leq 4\sqrt{|V(G)|} - 3$ which follows directly from Theorem 2.1. An improvement of this inequality can be found in [28]. This proof is based on a relation between slopes and majorities, the two notions introduced by Robertson and Seymour in [36] and Alon, Seymour, and Thomas in [6], respectively.

THEOREM 2.4 (see [28]). *For any planar graph G, $\mathbf{bw}(G) \leq \sqrt{4.5 \cdot |V(G)|}$.*

3. Bounding branch-width of *D*-dominated planar graphs. This section is devoted to the proof of the main combinatorial result of this paper: The branch-

width of any planar graph with a dominating set of size k is at most $3\sqrt{4.5}\sqrt{k}$. The idea of the proof is to show that for every planar graph G with a dominating set of size k there is a graph H on at most k vertices such that $\mathbf{bw}(G) \leq 3 \cdot \mathbf{bw}(H)$. Then Theorem 2.4 will do the rest of the job.

The construction of the graph H and the proof of $\mathbf{bw}(G) \leq 3 \cdot \mathbf{bw}(H)$ is not direct. First we prove that every planar graph with a dominating set D is a minor of some graph with a nice structure. We call these "structured" graphs nicely D-dominated. For a nicely D-dominated planar graph G we show how to define a graph $\mathbf{red}(G)$ on $|D|$ vertices. The most complicated part of the proof is the proof that $\mathbf{bw}(G) \leq 3 \cdot \mathbf{bw}(\mathbf{red}(G))$ (clearly this implies the main combinatorial result). The proof of this inequality is based on a more general result about isomorphism of special hypergraphs obtained from G and $\mathbf{red}(G)$ (Lemma 3.16) and the structural properties of nicely D-dominated graphs.

3.1. Auxiliary results. In this section we obtain some useful technical results about branch-width.

LEMMA 3.1. *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be hypergraphs with one hyperedge in common, i.e., $V(\mathcal{G}_1) \cap V(\mathcal{G}_2) = f$ and $\{f\} = E(\mathcal{G}_1) \cap E(\mathcal{G}_2)$. Then $\mathbf{bw}(\mathcal{G}_1 \cup \mathcal{G}_2) \leq \max\{\mathbf{bw}(\mathcal{G}_1), \mathbf{bw}(\mathcal{G}_2), |f|\}$. Moreover, if every vertex $v \in f$ has degree ≥ 2 in at least one of the hypergraphs (i.e., v is contained in at least two edges in $\mathcal{G}_1$ or in at least two edges in $\mathcal{G}_2$), then $\mathbf{bw}(\mathcal{G}_1 \cup \mathcal{G}_2) = \max\{\mathbf{bw}(\mathcal{G}_1), \mathbf{bw}(\mathcal{G}_2)\}$.*

Proof. Clearly, $\mathbf{bw}(\mathcal{G}_1 \cup \mathcal{G}_2) \geq \max\{\mathbf{bw}(\mathcal{G}_1), \mathbf{bw}(\mathcal{G}_2)\}$.

For $i = 1, 2$, let (T_i, τ_i) be a branch decomposition of $\mathcal{G}_i$ of width $\leq k$ and let $e_i = \{x_i, y_i\}$ be the edge of T_i having as endpoint the leaf $\tau_i(f) = x_i$. We construct tree T as follows. First we remove the vertices x_i and add edge $\{y_1, y_2\}$. Then we subdivide $\{y_1, y_2\}$ by introducing a new vertex y. Finally we add vertex x and make it adjacent to y.

We set $\tau(f) = x$. For any other edge $g \in E(\mathcal{G}_1) \cup E(\mathcal{G}_2)$ we set $\tau(g) = \tau_1(g)$ if $g \in E(\mathcal{G}_1)$ and $\tau(g) = \tau_2(g)$ otherwise.

Because $|\omega(\{y_1, y\})| = |\omega(\{y_2, y\})| = |\omega(\{x, y\})| \leq |f|$ and for any other edge of T, its order is equal to the order of the corresponding edge in one of the T_i's, we have that (T, τ) is a branch decomposition of width $\leq \max\{k, |f|\}$.

If every vertex v of f has degree ≥ 2 in one of the hypergraphs, then $|f| \leq \max\{|\omega(e_1)|, |\omega(e_2)|\} \leq k$. Thus in this case, (T, τ) is a branch decomposition of width $\leq k$. □

Let G be a connected Σ-plane graph with all vertices of degree at least two. For a vertex x of G and a pair (z, y) of two of its neighbors, we call (z, y) a *pair of consecutive neighbors of x* if edges $\{x, z\}$, $\{x, y\}$ appear consecutively in the cyclic ordering of the edges incident to x. (Notice that if x has only two neighbors y and z, then both (y, z) and (z, y) are pairs of consecutive neighbors of x.)

LEMMA 3.2. *Let G be a planar graph. Then G is the minor of a planar 2-connected graph H such that $\mathbf{bw}(H) = \max\{\mathbf{bw}(G), 2\}$.*

Proof. We use induction on the number of vertices in G. Every graph on at most three vertices is the minor of a complete graph on three vertices, which is 2-connected and has branch-width two. Suppose that the lemma is correct for every planar graph on at most n vertices.

Let G be a graph on $n + 1$ vertices.

Case A. *G is 2-connected.* In this case the lemma trivially holds.

Case B. *G is connected (but not 2-connected).* Then G has a cut vertex x. Let $V_1, V_2, \ldots, V_k$ be the vertex sets of the connected components of $G - \{x\}$. Let G_1 be

the subgraph of G induced by $V_1 \cup \{x\}$ and let G_2 be the subgraph of G induced by $V_2 \cup V_3 \cup \cdots \cup V_k \cup \{x\}$.

By the induction assumption, there are 2-connected planar graphs H_i, $i = 1, 2$, such that $\mathbf{bw}(H_i) = \max\{\mathbf{bw}(G_i), 2\}$, and $G_i \prec H_i$.

Planar graphs H_1 and H_2 have only one common vertex x, and thus the graph $H_1 \cup H_2$ is also planar. Let H be a Σ-plane graph which is a drawing of $H_1 \cup H_2$. Let a and b be two consecutive neighbors of x in H (i.e., vertices such that the edges $\{a, x\}$, $\{b, x\}$ are incident to the same face), where $a \in V(H_1)$ and $b \in V(H_2)$. We denote by H' the graph obtained from H by drawing the edge $\{a, b\}$ so that it does not intersect other edges of H (this is possible because $\{a, x\}$, $\{b, x\}$ are incident to the same face). Let us remark that H' is 2-connected and contains H (and therefore G) as a minor.

The complete graph K on three vertices $\{a, b, x\}$ has one common edge $\{a, b\}$ with H_1. The degrees of a and x in K are two, and at least two in H_1 (H_1 is 2-connected). By Lemma 3.1, we have that

$$\mathbf{bw}(H_1 \cup K) = \max\{\mathbf{bw}(H_1), 2\} = \max\{\mathbf{bw}(G_1), 2\}.$$

By applying Lemma 3.1, for $H_1 \cup K$ and H_2, we arrive at

$$\mathbf{bw}(H') = \mathbf{bw}(H_1 \cup H_2 \cup K) = \max\{\mathbf{bw}(G_1), \mathbf{bw}(G_2), 2\} \leq \max\{\mathbf{bw}(G), 2\}.$$

Since G is the minor of H', we have that $\mathbf{bw}(H') = \max\{\mathbf{bw}(G), 2\}$.

Case C. *G is not connected.* Let F be the graph obtained from G by adding an edge connecting two connected components. By making use of Lemma 3.1, it is easy to show that $\mathbf{bw}(F) \leq \max\{\mathbf{bw}(G), 2\}$, and this case can be reduced to Case B. □

A graph G is *weakly triangulated* if all its faces are of length two or three. A graph is $(2, 3)$-regular if all its vertices have degree two or three. Notice that the dual of a weakly triangulated graph is $(2, 3)$-regular and vice versa.

LEMMA 3.3. *Every 2-connected Σ-plane graph G has a weak triangulation H such that $\mathbf{bw}(H) = \mathbf{bw}(G)$.*

Proof. Because G is 2-connected every face of G is bounded by a cycle. Suppose that there is a face r of G bounded by a cycle $C = (x_0, \ldots, x_{s-1}), s \geq 4$. We show that there are vertices x_i and x_j that are not adjacent in C such that the graph G' obtained from G by adding the edge $\{x_i, x_j\}$ has $\mathbf{bw}(G') = \mathbf{bw}(G)$. By applying this argument recursively, one obtains a weak triangulation of G of the same branch-width.

If there are vertices x_i and x_j that are adjacent in G and are not adjacent in C, then we can draw a chord joining x_i and x_j in r. Because G is 2-connected it holds that $\mathbf{bw}(G) \geq 2$ and, therefore, the addition of multiple edges does not increase the branch-width. Suppose now that the cycle C is chordless. Let (T, τ) be a branch decomposition of G and let ω be its order function. Every edge f of T corresponds to the partition of $E(G)$ into two sets. One of these sets contains at least $\lceil |C|/2 \rceil \geq 2$ edges of C. By induction on the number of edges in G, it is easy to show that there is always an edge f of T such that for the corresponding partition (E_1, E_2) of $E(G)$, the set E_1 contains exactly two edges of C. Let e_1, e_2 be such edges. Because C is chordless and its length is at least four, we have that $\omega(f)$ contains at least two vertices, say x_i and x_j, of C that are not adjacent. Then adding edge $\{x_i, x_j\}$ does not increase the branch-width. (The decomposition can be obtained from T by subdividing f and adding the leaf corresponding to $\{x_i, x_j\}$ to the vertex subdividing f.) □

In the next lemma we use powerful duality results of Robertson and Seymour. Moreover, the implications of these results play the crucial role in our proof.

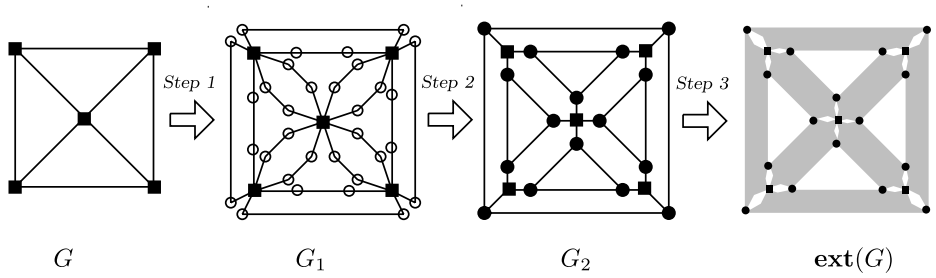


FIG. 1. *The steps* 1, 2, *and* 3 *of the definition of the function* **ext**.

LEMMA 3.4. *Every* 2*-connected* Σ*-plane graph* G *is the contraction of a* $(2,3)$*-regular* Σ*-plane graph* H *such that* $\mathbf{bw}(H) = \mathbf{bw}(G)$.

Proof. Let G^d be the dual graph of G. By Theorem 2.3, $\mathbf{bw}(G^d) = \mathbf{bw}(G)$ (the dual of a 2-connected graph is 2-connected, and any 2-connected graph contains K_2^2 as a minor). By Lemma 3.3, there is a weak triangulation H^d of G^d such that $\mathbf{bw}(H^d) = \mathbf{bw}(G^d)$. The dual of H^d, which we denote by H, contains G as a contraction (each edge removal in a planar graph corresponds to an edge contraction in its dual and vice versa). Applying Theorem 2.3 the second time, we obtain that $\mathbf{bw}(H) = \mathbf{bw}(H^d)$. Hence, $\mathbf{bw}(H) = \mathbf{bw}(G)$. Since H^d is weakly triangulated, we have that H is $(2,3)$-regular. □

3.2. Extensions of Σ-plane graphs. Let G be a connected Σ-plane graph where all the vertices have degree at least two. We define the *exension* of G, $\mathbf{ext}(G)$, as the hypergraph obtained from G by making use of the following three steps (see Figure 1 for an example).

Step 1. For each edge $e \in E(G)$, duplicate e and then subdivide each of its two copies twice. That way, each edge $e = \{x, y\}$ of G is replaced by a cycle denoted as $C_{x,y} = (x, x^+_{x,y}, y^-_{x,y}, y, y^+_{x,y}, x^-_{x,y}, x)$ (indexed in clockwise order). Let G_1 be the resulting graph.

Step 2. For each vertex $x \in V(G)$ and each pair (y, z) of consecutive neighbors of x (in G), identify the edges $\{x, x^-_{x,y}\}$ and $\{x, x^+_{x,z}\}$ in G_1. Let G_2 be the resulting graph.

Step 3. The hypergraph $\mathbf{ext}(G)$ is defined by setting $\mathbf{ext}(G) = (V(G_2), \{C_{x,y} \mid \{x, y\} \in E(G)\})$.

From the above construction, if $\mathcal{H} = \mathbf{ext}(G)$, then there exists a bijection $\theta : E(G) \to E(\mathcal{H})$ mapping each edge $e = \{x, y\}$ to the hyperedge formed by the vertices of $C_{x,y}$. See Figure 1 for an example of the definition of **ext**.

LEMMA 3.5. *For any* $(2,3)$*-regular* Σ*-plane graph* G, $\mathbf{bw}(\mathbf{ext}(G)) \leq 3 \cdot \mathbf{bw}(G)$.

Proof. Let (T, τ) be a branch decomposition of G of width $\leq k$. By the definition of $\mathbf{ext}(G)$, there is a bijection $\theta : E(G) \to E(\mathbf{ext}(G))$ defining which edge of G is replaced by which hyperedge of $\mathbf{ext}(G)$. Let L be the set of leaves in T. For $\mathbf{ext}(G)$ we define a branch decomposition (T, τ') with a bijection $\tau' : E(\mathbf{ext}(G)) \to L$ such that $\tau'(t) = \theta(\tau(t))$. We use the notations ω and ω' for the order functions of (T, τ) and (T, τ'), respectively.

We claim that (T, τ') is a branch decomposition of $\mathbf{ext}(G)$ of width $\leq 3k$. To prove the claim we show that for any $f \in E(T), |\omega'(f)| \leq 3 \cdot |\omega(f)|$. In other words, we need to show that it is possible to define a function σ_f mapping each vertex $v \in \omega(f)$ to a set of three vertices of $\omega'(f)$ such that every vertex $y \in \omega'(f)$ is contained in $\sigma_f(x)$ for some $x \in \omega(f)$.

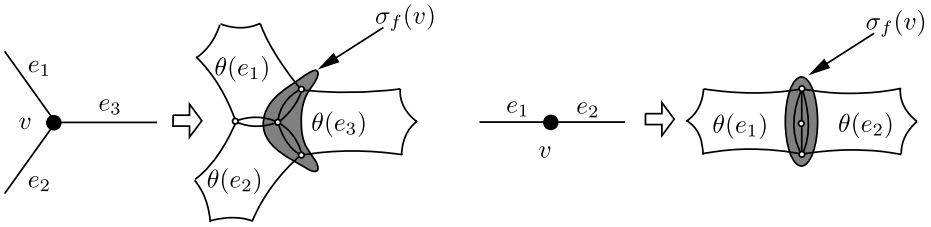


FIG. 2. *The construction of the value of $\sigma_f(v)$ in the proof of Lemma* 3.5.

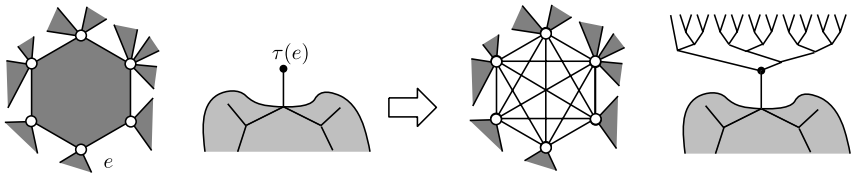


FIG. 3. *The construction of the branch decomposition of $\mathbf{cl}_E(H)$ in the proof of Lemma* 3.6.

Let T_1 and T_2 be the components of $T - \{f\}$. We construct σ_f by distinguishing two cases.

• *The degree of v is three in G.* We can assume that two of its incident edges, say e_1, e_2, are images of leaves of T_1 and one, say e_3, is an image of a leaf in T_2. We define $\sigma_f(v) = (\theta(e_1) \cap \theta(e_3)) \cup (\theta(e_2) \cap \theta(e_3))$. (This process is illustrated in the left half of Figure 2.)

• *The degree of v is two in G.* We can assume that one of its incident edges, say e_1, is an image of some leaf of T_1 and the other, say e_2, is an image of a leaf in T_2. We define $\sigma_f(v) = \theta(e_1) \cap \theta(e_2)$ (this is illustrated in the right half of Figure 2).

Note that in both cases $|\sigma_f(v)| = 3$. Suppose now that y is a vertex in $\omega'(f)$. Then y should be an endpoint of at least two hyperedges α and β of $\mathbf{ext}(G)$ and without loss of generality we assume that $\tau'(\alpha)$ is a leaf of T_1 and $\tau'(\beta)$ is a leaf of T_2. By the definition of τ', this means that $\tau(\theta^{-1}(\alpha))$ is a leaf of T_1 and $\tau(\theta^{-1}(\beta))$ is a leaf of T_2. By the construction of $\mathbf{ext}(G)$, $\theta^{-1}(\alpha)$ and $\theta^{-1}(\beta)$ have a vertex x in common; therefore $x \in \omega(f)$. From the definition of σ_f, we get that $y \in \sigma_f(x)$. This proves the relation $|\omega'(f)| \leq 3 \cdot |\omega(f)|$, and the lemma follows. □

Let $\mathcal{H}$ be a planar hypergraph and let $E \subseteq E(\mathcal{H})$. We set $\mathbf{cl}_E(\mathcal{H}) = (V(\mathcal{H}), E_{\mathcal{H}})$, where $E_{\mathcal{H}} = E(\mathcal{H}) - E \cup \{\{x, y\} \subseteq V(\mathcal{H}) \mid \exists_{e \in E(\mathcal{H})} : \{x, y\} \in e\}$ (in other words, we replace each hyperedge $e \in E$ by a clique formed by connecting each pair of endpoints of e).

LEMMA 3.6. *Let $\mathcal{H}$ be a hypergraph with every vertex of degree at least two. Then for any $E \subseteq E(\mathcal{H})$, $\mathbf{bw}(\mathbf{cl}_E(\mathcal{H})) \leq \mathbf{bw}(\mathcal{H})$.*

Proof. If (T, τ) is a branch decomposition of $\mathcal{H}$, then we construct a branch decomposition of $\mathbf{cl}_E(\mathcal{H})$ by identifying each leaf t where $\tau(t) \in E$ with the root of a binary tree T_t that has $\binom{|\tau(t)|}{2}$ leaves. The leaves of T_t are mapped to the edges of the clique made up by pairs of endpoints in $\tau(t)$ (see also Figure 3). □

LEMMA 3.7. *Let G and H be connected Σ-plane graphs with all vertices of minimum degree at least two and such that $G \preceq H$. Then $\mathbf{bw}(\mathbf{ext}(G)) \leq \mathbf{bw}(\mathbf{ext}(H))$.*

Proof. Let E' (resp., E'') be the set of edges that one should contract (resp., remove) in H in order to obtain G (clearly, we can assume that $E' \cap E'' = \emptyset$). Let

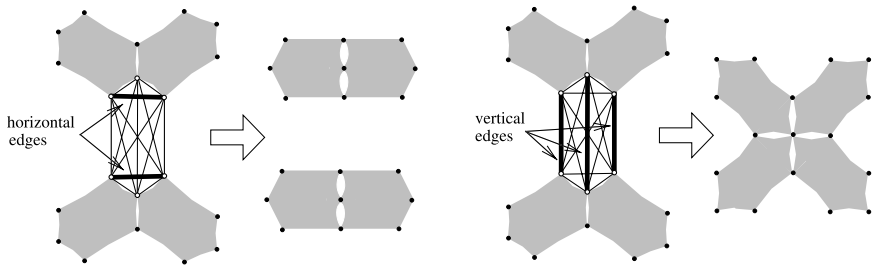


FIG. 4. *The construction of the branch decomposition of* $\mathbf{cl}_E(H)$ *in the proof of Lemma* 3.7.

θ be the bijection mapping edges of G to hyperedges of $\mathbf{ext}(G)$. If we prove that $\mathbf{ext}(G)$ is a minor of $\mathbf{cl}_{\theta(E'\cup E'')}(\mathbf{ext}(H))$, then the result will follow from Lemma 3.6. To see this, for each $e = \{x, y\} \in E'$, we separate the edges of the clique replacing $\theta(e) = (x, x^+_{x,y}, y^-_{x,y}, y, y^+_{x,y}, x^-_{x,y}, x)$ into two categories: We call $\{x^+_{x,y}, y^-_{x,y}\}$, $\{x, y\}$, and $\{y^+_{x,y}, x^-_{x,y}\}$ *horizontal* and we call the rest *unimportant*. Moreover, for any edge $e = \{x, y\} \in E''$, we separate the edges of the clique replacing $\theta(e) = (x, x^+_{x,y}, y^-_{x,y}, y, y^+_{x,y}, x^-_{x,y}, x)$ into two categories: We call $\{x^+_{x,y}, x^-_{x,y}\}$ and $\{y^+_{x,y}, y^-_{x,y}\}$ *vertical* and the rest *useless*. To obtain $\mathbf{ext}(G)$ from $\mathbf{cl}_{E'}(\mathbf{ext}(H))$ we first remove the useless and the unimportant edges and then contract all the horizontal and vertical ones (see Figure 4). □

We are ready to state the main property of **ext**.

LEMMA 3.8. *Let* G *be a connected* Σ*-plane graph with all vertices of degree at least two. Then* $\mathbf{bw}(\mathbf{ext}(G)) \leq 3 \cdot \mathbf{bw}(G)$.

Proof. The branch-width of G is at least two, and by Lemma 3.2, G is the minor of a 2-connected Σ-plane graph G' such that $\mathbf{bw}(G') = \mathbf{bw}(G)$. By Lemma 3.4, G' is the contraction of a $(2, 3)$-regular Σ-plane graph H where $\mathbf{bw}(H) \leq \mathbf{bw}(G')$. Notice that G is a minor of H and both G and H are Σ-plane and connected and have all vertices of degree at least two. By Lemma 3.7, $\mathbf{bw}(\mathbf{ext}(G)) \leq \mathbf{bw}(\mathbf{ext}(H))$. Note also that H is $(2, 3)$-regular. By Lemma 3.5, $\mathbf{bw}(\mathbf{ext}(H)) \leq 3 \cdot \mathbf{bw}(H)$, and the result follows. □

3.3. Nicely *D*-dominated Σ-plane graphs. An important tool spanning all of our proofs is the concept of unique D-domination. We call a D-dominated graph G *uniquely dominated* if there is no path of length < 3 connecting two vertices of D. Let us remark that this implies that each vertex $x \in V(G) - D$ has exactly one neighbor in D (i.e., is uniquely dominated).

We call a multiple edge $\{a, b\}$ represented by lines $l_1, l_2, \ldots, l_r$ of a D-dominated Σ-plane graph G *exceptional* if

- $a, b \notin D$;
- a and b are both adjacent to the same vertex in D;
- for any i, j, $i \neq j$, each of the open discs bounded by $l_i \cup l_j$ contains at least one vertex of D.

For example, all the multiple edges in the graphs in Figure 5 are exceptional.

LEMMA 3.9. *For every 2-connected* D*-dominated* Σ*-plane graph* G *without multiple edges, there exists a* Σ*-plane graph* H *such that the following hold:*

(a) G *is a minor of* H.
(b) H *is uniquely* D*-dominated.*
(c) *All multiple edges of* H *are exceptional.*

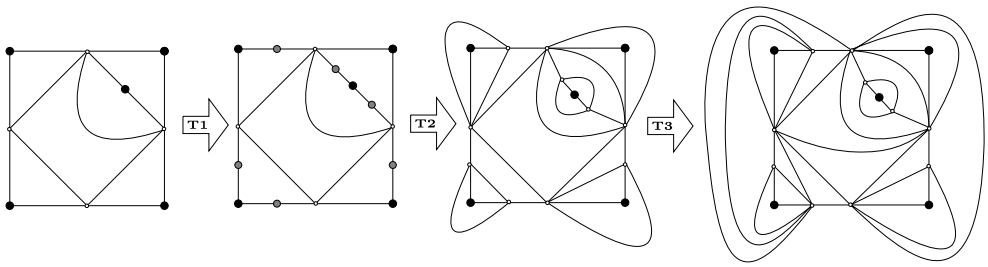


FIG. 5. *Example of the transformations* **T1**, **T2**, *and* **T3** *in the proof of Lemma* 3.9.

(d) *For any face r of H, $\widehat{r}$ is either a triangle or a square.*
(e) *If the distance between vertices $x, y \in D$ in H is three, then there exist at least two distinct (x, y)-paths in H of length three.*
(f) *If a (closed) face r of H contains a vertex of D, then $\widehat{r}$ is a triangle.*
(g) *Every square face of H contains two edges $e_i, i = 1, 2$, without common vertices such that for each $i = 1, 2$, there exists a vertex $x_i \in D$ adjacent to both endpoints of e_i.*
(h) *If $x, y \in D$, then every two distinct (x, y)-paths of H of length three are internally disjoint.*

Proof. We construct a graph H, satisfying properties (a)–(f), by applying, one after the other, on G the following transformations:

• **T1**. As long as there exists in G a vertex x with more than one neighbor y in D, subdivide the edge $\{x, y\}$.

We call the resulting graph G_1.

As G_1 does not have multiple edges, properties (a), (c) are trivially satisfied. Moreover, notice that, if G_1 is not uniquely dominated, then **T1** can be further applied. Therefore, (b) holds for G_1. For an example of the application of **T1**, see the first step of Figure 5.

• **T2**. As long as G_1 has a face r bounded by a cycle $\widehat{r} = (x_0, \ldots, x_{q-1}), q \geq 4$, and such that $x_i \in D$ for some $i, 0 \leq i \leq q-1$, add in G_1 the edge $\{x_{i-1}, x_{i+1}\}$ (indices are taken modulo q).

We call the resulting graph G_2.

Notice that the vertices of $\widehat{r}$ are distinct because G_2 is 2-connected. Clearly, G_2 satisfies property (a). Recall now that G_1 satisfies property (b). Therefore, if some vertex $x_i \in \widehat{r}$ is in D, then its neighbors x_{i-1} and x_{i+1} (the indices are taken modulo q) are not in D. Therefore, property (b) holds also for G_2. Notice that, if **T2** creates a multiple edge, then this can be only an exceptional multiple edge. Therefore, (c) holds for G_2. For an example of the application of **T2**, see the second step of Figure 5.

Finally note that none of the vertices of D is in a face of G_2 of length ≥ 4.

We call a square face that satisfies property (g) *solid*.

• **T3**. As long as G_2 has a face r that is not a solid square and such that $\widehat{r} = (x_0, \ldots, x_{q-1}), r \geq 4$, choose an edge in $\{\{x_1, x_3\}, \{x_0, x_2\}\}$ that is not already present in G_2 and add it to G_2.

We call the resulting graph G_3.

The above transformation can always be applied because it is impossible that both $\{x_1, x_3\}$ and $\{x_0, x_2\}$ are in the planar graph G_3. Therefore, property (c) is an invariant of **T3**. Clearly, G_3 satisfies property (a). Property (b) is an invariant of **T3** as the added edge has no endpoints in D. We have that all the faces of G_3 are either

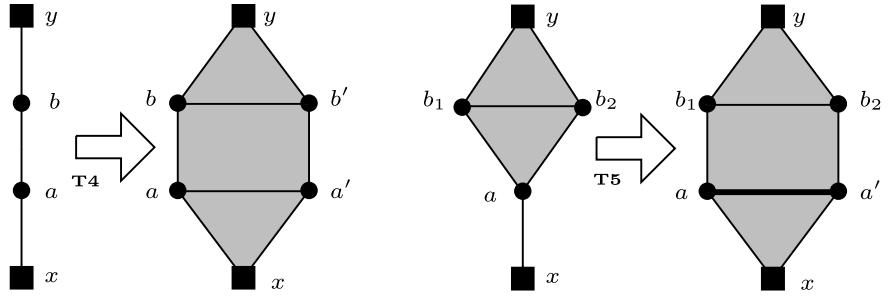


FIG. 6. *The transformations* **T4** *and* **T5** *in the proof of Lemma* 3.6.

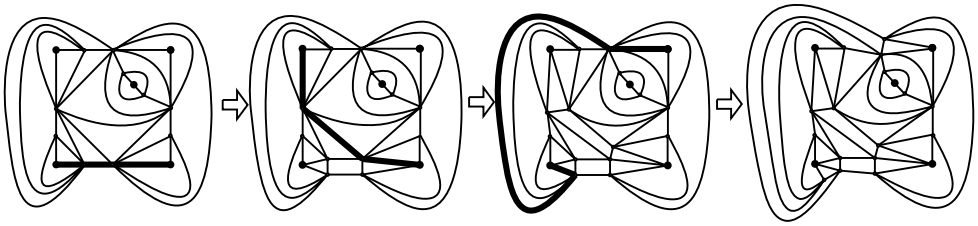


FIG. 7. *Example of the transformation* **T4** *in the proof of Lemma* 3.6.

triangles or solid squares and therefore G_3 also satisfies (d) and (g). For an example of the application of **T3**, see the third step of Figure 5.

• **T4**. As long as G_3 has a *unique* (x, y)-path $P = (x, a, b, y)$, where $x, y \in D$, apply the first transformation of Figure 6 on P.

We call the resulting graph G_4.

It is easy to verify that properties (a)–(d) are invariants of **T4**. Also, it is easy to see that the transformation of Figure 6 creates square faces with property (g) and does not alter property (g) for square faces that already have been created. Moreover, G_4 satisfies (e) because each time we apply the transformation of Figure 6 the number of pairs in D connected by unique paths decreases. Finally, none of the square faces appearing (because of **T4**) contains a vertex in D. Thus (f) holds. For an example of the application of **T4**, see Figure 7.

In order to give the transformation that enforces property (h) we need some definitions. Observe that if property (h) does not hold for G_4, this implies the existence of some pair of paths $P_i = (x, a, b_i, y), i = 1, 2$. We call the graph O defined by this pair an (h)-obstacle and we define its (h)-disc as the x-avoiding closed disc Δ_O bounded by the cycle (a, b_1, y, b_2, a). An (h)-obstacle is *minimal* if no (x, y)-path has vertices contained in its (h)-disc. Notice that if G_4 has an (h)-obstacle it also has a minimal (h)-obstacle and vice versa. We call an (h)-obstacle *hollow* if its (h)-disc contains no neighbor of a except b_1 and b_2. Notice that a hollow (h)-obstacle is always minimal. We claim that in any hollow (h)-disc, vertices b_1 and b_2 are adjacent. Indeed, by property (b), a is not adjacent to y in G_4. Therefore b_1, a, b_2 are in a face of G_4 that, from property (g), cannot be a square face (otherwise, property (b) would be violated). Therefore, (b_1, a, b_2) is a triangle and the claim follows.

• **T5**. As long as G_4 has a hollow (h)-obstacle O, apply the second transformation of Figure 6 on edge $\{a, x\}$ and the face bounded by (b_1, b_2, a).

We call the resulting graph G_5.

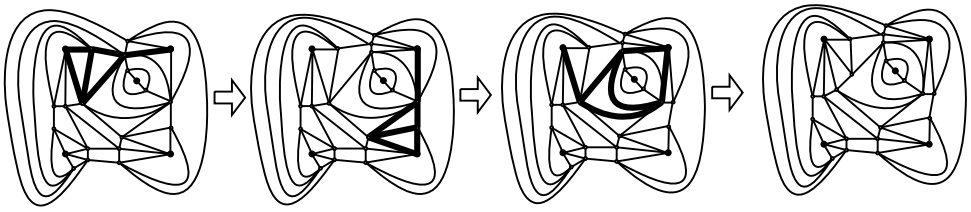


FIG. 8. *Example of the transformation* **T5** *in the proof of Lemma* 3.6.

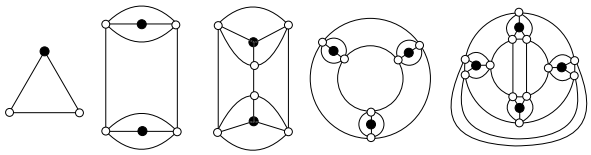


FIG. 9. *Simple examples of nicely* D*-dominated* Σ*-plane graphs.*

Notice that after **T5** none of the properties (a)–(g) is altered by the application of **T5** (the arguments are the same as those used for the previous transformations). Moreover, each time the second transformation of Figure 6 is applied, the number of hollow (h)-obstacles decreases and no new nonhollow (h)-obstacles appear. For an example of the application of **T5**, see Figure 8. To finish the proof, we show that **T5** is able to eliminate all the (h)-obstacles. It remains to prove the following claim.

Claim. If a 2-connected D-dominated Σ-plane graph satisfies properties (b)–(g) and contains a minimal (h)-obstacle, then it also contains a hollow (h)-obstacle.

Proof of claim. Let $O = (P_1, P_2)$ be a minimal nonhollow (h)-obstacle with (h)-disc Δ_O and let $\mathcal{O}$ be the set containing O along with of all the minimal (h)-obstacles that contain the edge $\{a, x\}$ and whose (h)-disc is a subset of Δ_O. If $O_1, O_2 \in \mathcal{O}$ and $\Delta_{O_1} \subset \Delta_{O_2}$, then we say that $O_1 < O_2$ (clearly, for any $O' \in \mathcal{O} - \{O\}$, $O' < O$). Let us remark that relation "$<$" is a partial order on $\mathcal{O}$ and that all its minimal elements are hollow (h)-obstacles. The claim follows and thus **T5** is able to enforce property (h). □

Let G be a connected D-dominated Σ-plane graph satisfying properties (b)–(h) of Lemma 3.9. We call such graphs *nicely* D-dominated Σ-plane graphs. For example, the graphs of Figure 9 and the last graph in Figure 8 are nicely D-dominated Σ-plane graphs (see also Figure 10 and all the graphs of Figure 11).

Given a nicely D-dominated Σ-plane graph G, we define $\mathcal{T}(G)$ as the set of all the triangles (cycles of length three) containing a vertex of D. By property (f), for every face r with $\widehat{r} \cap D \neq \emptyset$, $\widehat{r} \in \mathcal{T}(G)$. (The inverse is not always correct; i.e., not every triangle in $\mathcal{T}(G)$ bounds a face.) We call the triangles in $\mathcal{T}(G)$ D-triangles.

We also define $\mathcal{C}(G)$ as the set of all cycles consisting of two distinct paths of length three connecting two vertices of D (these are indeed cycles because of property (h) of nicely dominated graphs). Thus each cycle C in $\mathcal{C}(G)$ is of length six and is the union of two length-three paths connecting its two dominating vertices.

We call the cycles in $\mathcal{C}(G)$ *D-hexagons*. The *poles* of a cycle $C \in \mathcal{C}(G)$ are the vertices in $D \cap C$. We call a D-triangle T (D-hexagon C) *empty* if one of the open discs bounded in Σ by T (C) does not contain vertices of G. Notice that all empty

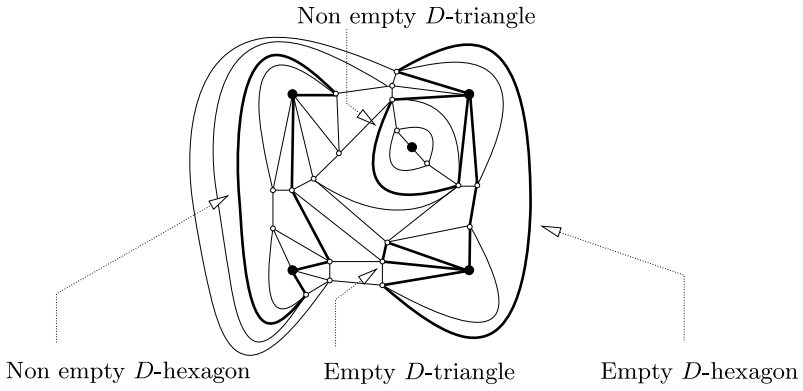


FIG. 10. *D-triangles and D-hexagons of the last graph of Figure* 8.

D-triangles are boundaries of faces of G. For some examples of the above definitions see Figure 10.

3.4. Decomposing nicely D-dominated Σ-planar graphs. In this subsection we show how nicely D-dominated planar graphs can be simplified. The idea is based on the structure imposed by properties (b)–(h): Any nicely D-dominated planar graph can be seen as the result of gluing together two simpler structures of the same type. This is described by the following two lemmata.

LEMMA 3.10. *Let G be a nicely D-dominated Σ-plane graph G and let $T \in \mathcal{T}(G)$ be a nonempty D-triangle bounding the closed discs Δ_1, Δ_2. Let also G_i, $i = 1, 2$, be the subgraph of G containing all vertices and edges included in Δ_i. Then $G_i, i = 1, 2$, is a nicely D_i-dominated graph for some $D_i \subseteq D$ and G_i has fewer vertices than G.*

Proof. Let $D_i = D \cap \Delta_i, i = 1, 2$. Clearly, $D_i \subseteq D$. Moreover, as T is nonempty, we have that $|V(G_i)| < |V(G)|$. Let us verify that properties (b)–(h) hold for $G_i, i = 1, 2$. First of all we observe that, by the construction of G_i, two vertices in G_i are adjacent if and only if they are adjacent in G. We will refer to this fact saying that G_i *preserves the adjacency of* G. (Note that since G can have multiple edges, G_i is not necessary an induced subgraph of G.)

To prove property (b), we show first that G_i is D_i-dominated. For the sake of contradiction, suppose that there exists a vertex $a \in V(G_i)$ that is not dominated by D_i. As property (b) holds for G, there exists a vertex $w \in D - D_i$ so that a is uniquely dominated by w in G. This means that $w \in \Sigma - \Delta_i$ and $a \in \Delta_i$. Therefore, a is a vertex of T. Because T is a D-triangle, there is some $x \in D \cap T$. Since a is adjacent in G_i to x and $x \neq w$, we have a contradiction to the property (b) on G. Now it remains to prove that G_i is uniquely D'-dominated and that this is a direct consequence of the fact that G_i preserves the adjacency of G.

For property (c), let $e = \{v, u\}$ be some multiple edge in G_i represented by edges $l_1, \ldots, l_r$, and suppose that x is the dominating vertex of T. As e is an exceptional multiple edge in G and because of property (b), none of its endpoints is in D and also $x \notin e$. Let Δ_l, Δ_l^* be the two closed discs defined by some pair l_h, l_j of edges representing e. By the definition of G_i, $l_h \cup l_j \subseteq \Delta_i$, therefore one, say Δ_l, of Δ_l, Δ_l^* includes T. As $x \notin e$, we have that $x \notin \hat{\Delta}_l$ and $\Delta_l - \hat{\Delta}_l$ contains some vertex of D. Observe now that $\Delta_l^* \subseteq \Delta_i$. Therefore, if $\Delta_l^* - \hat{\Delta}_l^*$ does not contain vertices of D in G_i, then the same holds also for G, which is a contradiction as e is exceptional in G. It remains now to prove that v and u are adjacent to the same vertex of D in G_i.

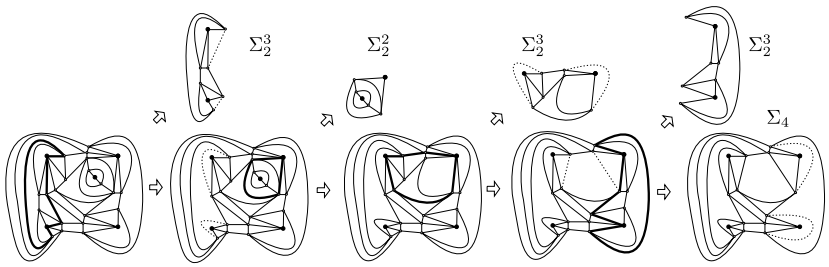


FIG. 11. *Examples of the application of Lemmata* 3.10 *and* 3.11.

Indeed, this is the case for G, and we let w be this vertex. If $w \notin \Delta_i$, then both v, u should be vertices of T, which contradicts property (b). Therefore, $w \in V(G_i)$ and property (c) holds for G_i.

For (d), we stress that all the faces of G_i that are in Δ_i are also the faces of G. Therefore, property (d) holds for all these faces. Also, it holds for the unique new face $r = \Sigma - \Delta_i$ of G_i because $\widehat{r}$ is a triangle.

For property (e), let x, y be two vertices in D_i of distance three in G_i. Let P_i^1 and P_i^2 be two internally disjoint paths connecting x and y in G (these paths exist because of properties (e) and (h) in G). Notice that (e) holds if we prove that both P_i^j, $j = 1, 2$, are paths of G_i, $i = 1, 2$, as well. Suppose to the contrary that one, say $P_i^1 = (x, a, b, y)$, of P_i^j, $j = 1, 2$, is not a path in G_i. This means that at least one of a, b is in $(\Sigma - \Delta_i) \cap V(G)$. It follows that two nonconsecutive vertices of P_i^1 are vertices of T. Therefore, the distance between x and y in G is at most two, a contradiction to property (b) for G.

Suppose now that (f) does not hold for G_i. As (d) holds for G_i we have that there exists a square in G_i containing a vertex of D. As G_i preserves the adjacency of G, this square also should exist in G, a contradiction to (f) for G.

To prove (g), suppose that (a, b, c, d) is a square of G_i. As G_i preserves the adjacency of G, (a, b, c, d) is also a square of G; therefore we may assume that there are vertices $z, w \in D$ where (z, a, b) and (w, c, d) are triangles of G. It is enough to prove that $\{z, a\}, \{z, b\}, \{w, c\}$, and $\{w, d\}$ are edges of G_i. Suppose to the contrary that one of them, say $\{a, z\}$, is not an edge of G_i. As G_i preserves the adjacency of G, this means that $z \notin V(G_i)$. In other words, we have that (z, a, b) is a triangle of G where $z \in (\Sigma - \Delta_i) \cap V(G)$ and $\{a, b\} \in \Delta_i \cap V(G)$. If this is true, then a, b should be vertices of T; therefore the distance in G between z and the dominating vertex belonging in T is at most two, a contradiction to property (b).

Finally, if there exist two paths violating (h) in G_i the same also should happen in G as G_i preserves the adjacency of G. □

For an example of the application of Lemma 3.10, see the second step of Figure 11.

LEMMA 3.11. *Let G be a nicely D-dominated Σ-plane graph G and let $C = (x, a, b, y, c, d, x)$ be a nonempty D-hexagon with poles x, y bounding the closed discs Δ_1, Δ_2. Let also $G_i, i = 1, 2$, be the graph containing all the edges and vertices included in Δ_i and extended by adding the edges $\{b, c\}$ and $\{a, d\}$ (edges $\{b, c\}$ and $\{a, d\}$ are placed outside D_i to ensure planarity of G_i). Then $G_i, i = 1, 2$, is a nicely D_i-dominated graph for some $D_i \subseteq D$ and $G_i, i = 1, 2$, has fewer vertices than G.*

Proof. Let G_i^- be a graph where $V(G_i^-) = \Delta_i \cap V(G)$ and $E(G_i^-) = \{e \in E(G) \mid e$ is included in $\Delta_i\}$; i.e. G_i^-, contains all edges and vertices included in Δ_i. Set $D_i = D \cap \Delta_i, i = 1, 2$. Therefore, G_i can be seen as the graph with $V(G_i) = V(G_i^-)$

and $E(G_i) = E(G_i^-) \cup \{\{b,c\} \cup \{a,d\}\}$. As in the proof of Lemma 3.10, we will say that G_i^- preserves the adjacency of G in the sense that two vertices in G_i^- are adjacent if and only if they are adjacent in G. We also have that $D_i \subseteq D$ and $|V(G_i)| < |V(G)|$.

Let us verify properties (b)–(h) for G_i, $i = 1, 2$.

To prove (b) we first claim that G_i is D_i-dominated. If some vertex $\alpha \in V(G_i) - D_i$ is not dominated by D_i, then it is dominated by some vertex $w \in D - D_i$ (property (b) for G). This means that $w \in \Sigma - \Delta_i$ implying $\alpha \in C$. Thus $\alpha \in \{a,b,c,d\}$. But this means that the distance between $w, x \in D$ or the distance between $w, y \in D$ in G is ≤ 2, which also violates (b) for G. Therefore G_i is D_i-dominated. Clearly, as G_i preserves the adjacency of G, G_i should be uniquely dominated and (b) holds for G_i.

For property (c), we will first prove that it holds for G_i^-. Let $e = \{v, u\}$ be some multiple edge in G_i^- represented by edges $l_1, \ldots, l_r$. As e is an exceptional multiple edge in G and because of property (b), none of its endpoints is in D and also $x, y \notin e$. Let Δ_l, Δ_l^* be the two closed discs defined by some pair l_h, l_j of edges representing e. By the definition of G_i^-, $l_h \cup l_j \subseteq \Delta_i$, therefore one of Δ_l, Δ_l^*, say Δ_l, includes C. As $x, y \notin e$, we have that $x, y \notin \hat{\Delta}_l$ and $\Delta_l - \hat{\Delta}_l$ contains some vertex of D. Observe now that $\Delta_l^* \subseteq \Delta_i$. Therefore, if $\Delta_l^* - \hat{\Delta}_l^*$ does not contain vertices of D, then the same holds also for G, which is a contradiction, as e is exceptional in G. It remains now to prove that v and u are adjacent to the same vertex of D in G_i^-. Since this holds for G, we have that there exists a vertex $w \in D$ such that $\{u, w\}, \{v, w\} \in E(G)$. If $w \notin \Delta_i$, then both v, u should be vertices of C, which contradicts property (b). Therefore, $w \in V(G_i^-)$ and property (c) holds for G_i^-. If now the addition of any, say $\{b, c\}$, of $\{b, c\}$, $\{a, d\}$ creates a multiple edge, then $\{b, c\}$ should already be an edge in G_i^-. Suppose then that $l_{\text{old}}, l_{\text{new}}$ are two lines in G_i, representing $\{b, c\}$, and l_{new} is the newly added one. As $l_{\text{new}} \not\subseteq D_i$ and $l_{\text{old}} \subseteq D_i$, it follows that the one of the open discs defined by $l_{\text{old}} \cup l_{\text{new}}$ contains y and the other contains x. Therefore, (c) holds also for G_i.

Notice that all the faces of G_i that are included in Δ_i are also faces of G_i. The boundaries of the new faces are the cycles (y, a, b), (a, b, c, d), and (x, c, b) that are all either triangles or squares. Therefore, (d) holds for G_i.

If property (e) holds for G_i^-, then it also holds for G_i. Let P be a (w, v)-path in G_i^- of length three. Property (e) holds trivially for G_i^- if $\{w, v\} = \{x, y\}$. So suppose that it is violated for some pair $\{w, v\} \neq \{x, y\}$. Because (e) holds for G, we can find a $\{w, v\}$-path $P' = (w, \alpha, \beta, v)$ of length three in G that is not a path in G_i^-. As $\{w, v\} \neq \{x, y\}$, only one, say α, of α, β can be outside Δ_i. This means that w and β are vertices of C. Since $\beta \in \{a, b, c, d\}$, we have that v is adjacent in G to a vertex in $\{a, b, c, d\}$. This contradicts property (b) for G, as it implies the existence of a path of length ≤ 2 connecting $v \in D$ and one of the vertices $x, y \in D$.

It is easy to verify (f) for the new faces (x, a, d), (a, b, c, d), and (y, c, d) of G_i. Suppose now that (f) is violated for some face of G_i that is also a face of G. As (d) holds for G_i, we have that there exists a square in G_i containing a vertex of G_i. As G_i preserves the adjacency of G, this square should exist also in G, a contradiction to (f) for G.

Property (g) is trivial for the new square face of G_i bounded by (a, b, c, d). Let us prove that (g) also holds for all the square faces of G_i^-. Let $\widehat{r} = (\alpha, \beta, \gamma, \delta)$ be the boundary of some square face r of G_i^-. As G_i^- preserves the adjacency of G, $(\alpha, \beta, \gamma, \delta)$ is also the boundary of some square face of G. Therefore, we may assume that there are vertices $z, w \in D$ where (z, α, β) and (w, γ, δ) are triangles of G. It is

enough to prove that $\{z,\alpha\},\{z,\beta\},\{w,\gamma\}$, and $\{w,\delta\}$ are all edges of G_i^-. Suppose, to the contrary, that one of them, say $\{a,z\}$, is not an edge of G_i^-. As G_i^- preserves the adjacency of G, this means that $z \notin V(G_i^-)$. In other words, we have that (z,α,β) is a triangle of G, where $z \in (\Sigma - \Delta_i) \cap V(G)$ and $\{\alpha,\beta\} \in \Delta_i \cap V(G)$. Then α,β should be vertices of C different from x and y. Therefore, either z,x or z,y are at distance at most two in G, contradicting property (b).

For (h), we observe that no path of length three in G_i connecting two vertices of D can use the edges $\{a,d\}$ and $\{b,c\}$ in G_i. Indeed, if this is possible for one, say $\{a,d\}$, of the edges $\{a,d\}$ and $\{b,c\}$, then such a path would have extremes in distance two from x, a contradiction to property (b) for G_i. Therefore, if there exist two paths violating (h) in G_i, they should be paths of G_i^- and also paths of G as G_i^- preserves the adjacency of G, a contradiction to property (b). □

For an example of the application of Lemma 3.10, see steps 1, 3, and 4 of Figure 11.

3.5. Prime *D*-dominated Σ-plane graphs. A nicely D-dominated Σ-plane graph G is a *prime D-dominated Σ-plane graph* (or just *prime*) if all its D-triangles and D-hexagons are empty. For example, all the graphs in Figure 9 are prime.

LEMMA 3.12. *Let G be a prime D-dominated Σ-plane graph. If G contains two vertices $x,y \in D$ connected by three paths of length three, then $V(P_1) \cup V(P_2) \cup V(P_3) = V(G)$.*

Proof. By property (h), the paths $P_i, i = 1,2,3$, are mutually internally disjoint. Then $\Sigma - (P_1 \cup P_2 \cup P_3)$ contains three connected components that are open discs. We call them $\Delta_{1,2}$, $\Delta_{2,3}$, and $\Delta_{1,3}$ assuming that they do not contain vertices of P_3, P_1, and P_2, respectively. Let i,j,h be any three distinct indices of $\{1,2,3\}$. As $P_i \cup P_j$ forms an empty D-hexagon, all the vertices of G should be contained in one, say Δ, of the closed discs bounded by the cycle $P_i \cup P_j$. Notice that P_h should be entirely included in $\overline{\Delta}_{i,j}$ because of its internal vertices. Therefore, $\Delta = \overline{\Delta}_{i,j}$ and thus $V(G) = V(G) \cap \overline{\Delta}_{i,j}$. Resuming, we have that $V(G) = V(G) \cap (\overline{\Delta}_{1,2} \cap \overline{\Delta}_{2,3} \cap \overline{\Delta}_{1,3})$ and the lemma follows as $\overline{\Delta}_{1,2} \cap \overline{\Delta}_{2,3} \cap \overline{\Delta}_{1,3}$ contains exactly the vertices of the paths $P_i, i = 1,2,3$. □

The graph Σ_2^3 of Figure 11 is a graph satisfying the conditions of Lemma 3.12.

Let us recall that $\mathcal{C}(G)$ is the set of all cycles consisting of two distinct paths of length three connecting two vertices of D. For a nicely D-dominated Σ-plane graph G, we define its *reduced* graph, $\mathbf{red}(G)$, as the graph with vertex set D and where two vertices $x,y \in D$ are adjacent in $\mathbf{red}(G)$ if and only if the distance between x and y in G is three. Let us stress that $\mathbf{red}(G)$ is a connected graph. The main idea of our proof is that $\mathbf{red}(G)$ expresses a "good" part of the structure of a nicely D-dominated graph G.

An important relation of a prime graph and its reduced graph is provided by the following lemma.

LEMMA 3.13. *Let G be a prime D-dominated Σ-plane graph with $|D| \geq 3$. Then the mapping*

$$\phi : E(\mathbf{red}(G)) \to \mathcal{C}(G), \text{ where } \phi(e) = C \text{ if and only if the endpoints of } e \text{ are in } D \cap C,$$

is a bijection.

Proof. Clearly, any D-hexagon C with poles x and y implies the existence of a (x,y)-path in G and therefore C is the image of $\{x,y\} \in E(\mathbf{red}(G))$. In order to show that ϕ is a bijection, we have to show that for every $e = \{x,y\} \in E(\mathbf{red}(G))$, there exists a *unique* D-hexagon C with poles x and y. By the definition of $\mathbf{red}(G)$, x and y are within distance three in G. By properties (e) and (h) of nicely

D-dominated Σ-plane graphs, there are at least two internally disjoint paths connecting x and y. Suppose to the contrary that G has at least three (x,y)-paths P_1, P_2, P_3. As $|D| \geq 3$, G contains vertices that are not in $V(P_1) \cup V(P_2) \cup V(P_3)$, a contradiction to Lemma 3.12. □

Let G be a prime D-dominated Σ-plane graph with $|D| \geq 3$ and let ϕ be the bijection defined in Lemma 3.13. For every edge $e = \{x,y\} \in E(\mathbf{red}(G))$, we choose a vertex $w \in D - \{x\} - \{y\}$ and define $\Delta(e)$ as the w-avoiding open disc bounded by $\phi(e)$ (because G is prime, the definition does not depend on the choice of w). Observe that for any two different $e_1, e_2 \in E(\mathbf{red}(G))$, it holds that $\Delta(e_1) \cap \Delta(e_2) = \emptyset$.

Some of the properties of prime D-dominated Σ-plane graphs are given by the next two lemmata.

LEMMA 3.14. *Let G be a prime D-dominated Σ-plane graph with $|D| \geq 2$. For any D-triangle $T = (x,a,b)$ with $x \in D$, the edges $\{x,a\}$ and $\{x,b\}$ are also the edges of some D-hexagon of G with poles x and $y \in D$. Moreover, if $|D| \geq 3$, the edge $\{a,b\}$ is in $\Delta(\{x,y\})$.*

Proof. Because G is a prime graph, one of the open discs bounded by T is a face of G. Let r_x, $\widehat{r}_x = T = (x,a,b)$, be such a face. Let r, $r \neq r_x$, be the (unique) face incident to $\{a,b\}$, i.e., $\{a,b\} \subseteq \widehat{r}$. By (d), r is either a triangle or a square face.

We claim that it is a square face. Suppose to the contrary that $\widehat{r} = (a,b,c)$. Then, from property (b), $c \notin D$. Let $y \in D$ be the unique vertex dominating c. We distinguish two cases:

Case 1. $x = y$. In this case all vertices in $V(G) - \{x,a,b,c\}$ are covered (in Σ) by four open discs bounded by triangles $(x,a,b), (x,a,c), (x,b,c)$, and (a,b,c). Since G is prime, all D-triangles $(x,a,b), (x,a,c), (x,b,c)$ are empty. Therefore, all vertices in $V(G) - \{x,a,b,c\}$ are in the x-avoiding open disc Δ bounded by (a,b,c). As $\Delta = r$ is a face of G, we have that $V(G) - \{x,a,b,c\} = \emptyset$, a contradiction to the fact that $|D| \geq 2$.

Case 2. $x \neq y$. Then G contains the paths (x,a,c,y) and (x,b,c,y), a contradiction to property (h), and the claim holds.

As r is a square face, we assume that $\widehat{r} = (a,b,c,d)$. Property (g), together with the fact that a, b are adjacent to x, implies that either all vertices a,b,c,d are adjacent to x, or there is $y \in D$, $y \neq x$, that is adjacent to c and d.

We claim that the first case is impossible. Indeed, if a,b,c,d are adjacent to x, then all the vertices in $V(G) - \{x,a,b,c,d\}$ should be included in the five open discs bounded by triangles $(x,a,b), (x,a,c), (x,b,d)$, (c,d,x) and square (a,b,c,d). Four discs bounded by D-triangles are faces of G (G is prime); thus all the vertices of $V(G) - \{x,a,b,c,d\}$ are in the x-avoiding open disc r bounded by (a,b,c,d). Because r is a face of G, we conclude that $V(G) - \{x,a,b,c,d\} = \emptyset$. Since by property (b), $a,b,c,d \notin D$, we have a contradiction to the fact that $|D| \geq 2$, and the claim holds.

Therefore, there is $y \in D$, $y \neq x$, and y is adjacent to c and d. Because (y,c,d) is a D-triangle in a prime graph, one of the discs r_y bounded by (y,c,d) is the face of G. Hence $C = (x,a,c,y,d,b,x)$ is a D-hexagon containing edges $\{x,a\}$ and $\{x,b\}$, as required. Notice now that $\Delta = r_x \cup \{a,b\} \cup r \cup \{c,d\} \cup r_y$ is one of the open discs bounded by C (here an edge represents an open set). As $V(G) \cap \Delta = \emptyset$, we have that $\Delta(\{x,y\}) = \Delta$ and thus the edge $\{a,b\}$ is contained in $\Delta(\{x,y\})$. □

LEMMA 3.15. *Let G be a prime D-dominated Σ-plane graph with $|D| \geq 2$. Then the endpoints of each edge of G are the vertices of some D-hexagon.*

Proof. Let $e = \{x,y\}$ be an edge of G.

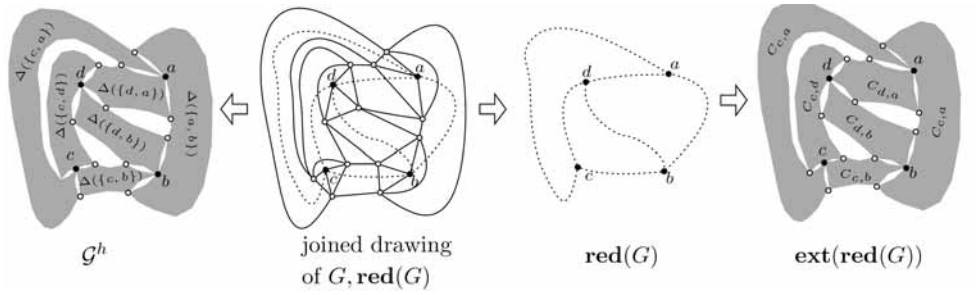


FIG. 12. *An example of the proof of Lemma* 3.17.

Case 1. $\{x, y\} \cap D = \{x\}$ (by property (b), $|\{x, y\} \cap D| \leq 1$). Let r be the face of G incident to $e = \{x, y\}$. From property (f), r is a D-triangle and the result follows from Lemma 3.14.

Case 2. $\{x, y\} \cap D = \emptyset$. Let d_x and d_y be the vertices of D-dominating x and y, respectively. If $d_x = d_y$, then e is incident to the D-triangle (d_x, x, y), and the result follows from Lemma 3.14. Suppose now that $d_x \neq d_y$. Then (d_x, x, y, d_y) is the path connecting two vertices in D. From property (e), $\{x, y\}$ belongs to the union of two distinct paths connecting d_x and d_y. Therefore, $\{x, y\}$ should be an edge of some D-hexagon and the lemma follows. □

3.6. On the structure of nicely D-dominated Σ-plane graphs. For a given nicely D-dominated Σ-plane graph G, we define hypergraph $\mathcal{G}^*$ with the vertex set $V(\mathcal{G}^*) = V(G)$ and edge set $E(\mathcal{G}^*) = E(G) \cup \mathcal{T}(G) \cup \mathcal{C}(G)$; i.e., $\mathcal{G}^*$ is obtained from G by adding all D-triangles and D-hexagons as hyperedges. We also define hypergraph $\mathcal{G}^h$ with the vertex set $V(\mathcal{G}^h) = V(G)$ and the edge set $E(\mathcal{G}^h) = \mathcal{C}(G)$; i.e., $\mathcal{G}^h$ has the vertices of G as vertices and each of its hyperedges contains the vertices of some D-hexagon of G. Observe that $\mathcal{G}^h$ can be obtained from $\mathcal{G}^*$ by removing all the (hyper)edges of size two and three.

LEMMA 3.16. *For any prime D-dominated Σ-plane graph G with $|D| \geq 2$,* $\mathbf{bw}(\mathcal{G}^*) \leq \max\{\mathbf{bw}(\mathcal{G}^h), 3\}$.

Proof. By Lemmata 3.14 and 3.15, we have that for each hyperedge in $\mathcal{G}^*$ there exists some D-hexagon containing all its endpoints. In other words, each hyperedge of $\mathcal{G}^*$ is a subset of some hyperedge of $\mathcal{G}^h$. By applying Lemma 3.1 recursively for every hyperedge f of $\mathcal{G}^*$ that is an edge or a triangle, we arrive at $\mathbf{bw}(\mathcal{G}^*) \leq \max\{\mathbf{bw}(\mathcal{G}^h), 3\}$. □

The following structural result will serve as a base for the recursive application of Lemmata 3.10 and 3.11 in the proof of Lemma 3.21.

LEMMA 3.17. *Let G be a prime D-dominated Σ-plane graph with $|D| \geq 3$. Then* $\mathbf{red}(G)$ *is a connected Σ-plane graph, all vertices of G have degree at least two, and* $\mathcal{G}^h$ *is isomorphic to* $\mathbf{ext}(\mathbf{red}(G))$.

Proof. We define the *joined drawing of G and* $\mathbf{red}(G)$ *in* Σ as follows:

Take a drawing of G on Σ and draw the vertices of $\mathbf{red}(G)$ identically to the vertices of G. For each edge $e_i = \{x, y\} \in E(\mathbf{red}(G))$ we draw $\{x, y\}$ as an I-arc connecting x and y and contained in $\Delta(e_i)$.

For an example of joined drawing, see the second drawing of Figure 12. The following three auxiliary propositions are used in the proof of the lemma.

PROPOSITION 3.18. *If G is a prime D-dominated Σ-plane graph, then* $\mathbf{red}(G)$ *is a Σ-plane graph.*

To prove the proposition, let us take the joined drawing of G and $\mathbf{red}(G)$ in Σ. Observe that, for any pair of edges $e_i, e_i \in E(\mathbf{red}(G))$, $\Delta(e_i) \cap \Delta(e_i) = \emptyset$. Therefore, if in this drawing we delete all the points that are not points of vertices or edges of $\mathbf{red}(G)$, what remains is a planar drawing of $\mathbf{red}(G)$.

PROPOSITION 3.19. *Let G be a prime D-dominated Σ-plane graph where $|D| \geq 3$ and let ϕ be the bijection defined in Lemma* 3.13. *In the joined drawing of G and $\mathbf{red}(G)$ in Σ, for any vertex $x \in D$, of degree at least three, two edges $\{x, y\}$ and $\{x, z\}$ are consecutive if and only if the D-hexagons $\phi(\{x, y\})$ and $\phi(\{x, z\})$ have exactly one edge in common. In the special case where $x \in D$ has degree two, the D-hexagons $\phi(\{x, y\})$ and $\phi(\{x, z\})$ have exactly two edges in common.*

In fact, let $\phi(\{x, y\})$ and $\phi(\{x, z\})$ be two hexagons sharing only x as a common vertex. By property (f), all faces of G incident to x are bordered by triangles that in turn are cyclically ordered according to the cyclic ordering of their edges incident to x. This ordering contains one triangle from $\phi(\{x, y\})$ and one from $\phi(\{x, z\})$. The removal of these triangles from the cyclic ordering breaks it into two nonempty subintervals, such that each of the subintervals contains one of the triangles T_1 and T_2. By Lemma 3.14, each of T_1, T_2 is a part of some D-hexagon $\phi(\{x, z_1\})$ and $\phi(\{x, z_2\})$, respectively, and this implies that the edges $\{x, y\}$ and $\{x, z\}$ cannot be consecutive in $\mathbf{red}(G)$. The inverse direction follows directly by the definition of the joined drawing of G and $\mathbf{red}(G)$.

PROPOSITION 3.20. *Let G be a prime D-dominated Σ-plane graph where $|D| \geq 3$. Then all vertices of $\mathbf{red}(G)$ have degree at least two.*

In fact, let $x \in D$ be a vertex of G incident to a face r. By property (f) of Lemma 3.9, the boundary of r is a triangle $\widehat{r} = (x, a_1, a_2)$. By Lemma 3.14, the edges $\{x, a_1\}$ and $\{x, a_2\}$ are also the edges of some D-hexagon with poles x and y. We distinguish the following cases:

Case 1. x has a neighbor a_3, distinct from a_1 and a_2. We choose a_3 so that a_2 and a_3 are consecutive in the cyclic ordering of the neighbors of x. Note also that the unique face whose boundary contains x, a_2, and a_3 should be a triangle (otherwise we have a contradiction to property (f)). By Lemma 3.14, the edges $\{x, a_2\}$ and $\{x, a_3\}$ are contained in some D-hexagon with poles x and w. Clearly $w \neq y$ (otherwise x and y are connected by three internally disjoint paths), and from Lemma 3.12 we have that $|D| = 2$, a contradiction. We conclude that $\{x, w\}$ is an edge of $\mathbf{red}(G)$, different from $\{x, y\}$.

Case 2. The only neighbors of x are the vertices a_1 and a_2. From property (f), $e = \{a_1, a_2\}$ is an exceptional edge; i.e., there are two lines l_1 and l_2, representing e, whose extremes are a_1 and a_2. Let T^1, T^2 be the triangles containing x and lines l_1 and l_2, respectively. For $i = 1, 2$, we apply Lemma 3.14 for T^i and derive that both $\{x, a_i\}, i = 1, 2$, belong to some D-hexagon C^i of G with poles x and y_i. Moreover, as $|D| \geq 3$, the line l_i is contained in $\Delta(\{x, y_i\})$. Therefore, for the case $y_1 = y_2$ we have that both lines l_1, l_2 are in $\Delta(\{x, y_i\})$, which is impossible. So, x has two neighbors in $\mathbf{red}(G)$, which completes the proof of Proposition 3.20.

Now we are in position to prove Lemma 3.17.

By Proposition 3.18, G is a Σ-plane graph. By Proposition 3.20, all vertices of $\mathbf{red}(G)$ have degree at least two. Therefore, the three transformation steps of **ext** can be applied on $\mathbf{red}(G)$. Consider now the joint drawing of G and $\mathbf{red}(G)$ in Σ. For each edge $e = \{x, y\} \in E(\mathbf{red}(G))$, we use the notation $\phi(x, y) = (x, x^+_{x,y}, y^-_{x,y}, y, x^+_{x,y}, x^-_{x,y}, x)$ (the ordering is clockwise). Apply Steps 1 and 2 of the definition of **ext** on $\mathbf{red}(G)$. During Step 2, identify vertices $x^-_{x,y}$, $x^+_{x,z}$ with the

vertices of G that are denoted in the same way. This is possible because of Proposition 3.19 and because the graph G_2 created after Step 2 has exactly the same vertex set as the graph G. Let us recall that there exists a bijection $\theta : E(G) \to E(\mathbf{ext}(G))$ mapping each edge $e = \{x, y\}$ to the hyperedge formed by the vertices of $C_{x,y}$. Moreover, for any edge $e = \{x, y\} \in E(\mathbf{red}(G))$, the cycle $\theta(x, y) = C_{x,y}$ is identical to the D-hexagon $\phi(x, y)$. Notice now that the application of Step 3 of the definition of **red** on G_2 ignores the edges of G_2 and adds as edges all the cycles $\phi(e), e \in E(\mathbf{red}(G))$. As these cycles are exactly those added toward constructing $\mathcal{G}^h$, the graph $\mathcal{G}^h$ is also identical to the result of Step 3. Thus $\mathcal{G}^h$ is isomorphic to $\mathbf{ext}(\mathbf{red}(G))$. □

3.7. Main combinatorial result.

LEMMA 3.21. *For any nicely D-dominated Σ-plane graph G,* $\mathbf{bw}(G) \leq 3 \cdot \sqrt{4.5 \cdot |D|}$.

Proof. For $|D| = 1$, $G - D$ is outerplanar. It is well known that the branch-width of an outerplanar graph is at most two, implying $\mathbf{bw}(G) \leq 3$.

Suppose that $|D| \geq 2$. Clearly, $\mathbf{bw}(G) \leq \mathbf{bw}(\mathcal{G}^*)$, and to prove the lemma we show that $\mathbf{bw}(\mathcal{G}^*) \leq 3 \cdot \sqrt{4.5 \cdot |D|}$.

Prime case. We first examine the special case where G is a prime D-dominated Σ-plane graph. There are two subcases:

• If $|D| = 2$, then we set $D = \{x, y\}$. If there are only two (x, y)-paths in G, then $G = \Sigma_2^2$. If there are three (x, y)-paths in G, then $G = \Sigma_2^3$ (see Figure 9). Moreover, G cannot contain more than three (x, y)-paths; otherwise it would not be prime. Therefore, $|V(G)| \leq 8$ and thus $\mathbf{bw}(\mathcal{G}^*) \leq 8 \leq 3 \cdot \sqrt{4.5 \cdot 2} = 9$.

• Suppose now that G is a prime D-dominated Σ-plane graph and $|D| \geq 3$. By Theorem 2.4, $\mathbf{bw}(\mathbf{red}(G)) \leq \sqrt{4.5 \cdot |D|}$. By Lemma 3.17, all the vertices $\mathbf{red}(G)$ have degree ≥ 2. Therefore, we can apply Lemma 3.8 on $\mathbf{red}(G)$ (recall that $\mathbf{red}(G)$ is connected) and get $\mathbf{bw}(\mathbf{ext}(\mathbf{red}(G))) \leq 3 \cdot \mathbf{bw}(\mathbf{red}(G))$. By Lemma 3.17, $\mathbf{bw}(\mathcal{G}^h) = \mathbf{bw}(\mathbf{ext}(\mathbf{red}(G)))$ and by Lemma 3.16, $\mathbf{bw}(\mathcal{G}^*) \leq \max\{\mathbf{bw}(\mathcal{G}^h), 3\}$. Resuming, we conclude that if G is prime, then $\mathbf{bw}(\mathcal{G}^*) \leq 3 \cdot \sqrt{4.5 \cdot |D|}$.

General case. Suppose that G is a nicely D-dominated Σ-plane graph. We use induction on the number of vertices of G. If $|V(G)| = 3$, then G is a triangle (the graph Σ_1 of Figure 9) and $\mathbf{bw}(\mathcal{G}^*) = 3 \leq 3 \cdot \sqrt{4.5}$. Suppose that $\mathbf{bw}(\mathcal{G}^*) \leq 3 \cdot \sqrt{4.5 \cdot |D|}$ for every nicely D-dominated graph on $< n$ vertices. Let G be a nicely D-dominated Σ-plane graph where $|V(G)| = n$ and let q be a nonempty D-triangle or D-hexagon (if q does not exist, then the induction step follows by the prime case above). By Lemmata 3.10 and 3.11, we have that if Δ_1, Δ_2 are the discs bounded by q, then, for $i = 1, 2$, $G_i = G[V(G) \cap \Delta_i]$ is a subgraph of a nicely D_i-dominated Σ-plane graph for some $D_i \subseteq D$, $i = 1, 2$, and that $|V(G_i)| < n$ (we use the expression "subgraph" in order to capture the case when q is a D-hexagon). Applying the induction hypothesis, we get that $\mathbf{bw}(G_i^*) \leq 3 \cdot \sqrt{4.5 \cdot |D_i|}, i = 1, 2$. Notice also that $\mathcal{G}^* = \mathcal{G}_1^* \cup \mathcal{G}_2^*$ and that $V(\mathcal{G}_1^*) \cap V(\mathcal{G}_2^*) = q \in E(\mathcal{G}_1^*) \cap E(\mathcal{G}_2^*)$. Therefore, we can apply Lemma 3.1 and we get $\mathbf{bw}(\mathcal{G}^*) \leq 3 \cdot \sqrt{4.5 \cdot |D_i|}$ (recall that $|q| \leq 6$). □

For an example of the induction of the general case in the proof of Lemma 3.21, see Figure 11.

The following is the main combinatorial result of this paper.

THEOREM 3.22. *Let G be a D-dominated Σ-plane graph. Then* $\mathbf{bw}(G) \leq 3\sqrt{4.5 \cdot |D|}$.

Proof. If the branch-width of G is at most one, the theorem is trivial. Suppose that $\mathbf{bw}(G) \geq 2$. Then removing multiple edges does not decrease the branch-width of G, and we can assume that G is simple.

Let A be the set of cut vertices of G. Let G_i be the 2-connected components of G, $D_i = D \cap V(G_i)$, and $A_i = A \cap V(G_i)$, $1 \leq i \leq r$. Let also N_i be the vertices of G_i that are not dominated by D_i, $1 \leq i \leq r$.

Note also that each vertex of N_i is dominated in G by some vertex from $V(G) - V(G_i)$. Moreover, a vertex from $V(G) - V(G_i)$ cannot dominate more than one vertex in G_i. Therefore, $|N_i| \leq |D - D_i|$. Thus for $D'_i = N_i \cup D_i$, we have that G_i is D'_i-dominated and $|D'_i| \leq |D|$.

Consider now two cases for the graph $G_i, 1 \leq i \leq r$.

Case 1. G_i is a D'_i-dominated 2-connected planar graph. We take a drawing of this graph in a sphere Σ and apply Lemma 3.9. In this way, we construct a nicely D'_i-dominated Σ-plane graph H_i containing (property (a)) G_i as a minor. By Lemma 3.21, $\mathbf{bw}(H_i) \leq 3 \cdot \sqrt{4.5 \cdot |D'_i|}$. Since G_i is a minor of H_i, we have that $\mathbf{bw}(G_i) \leq 3\sqrt{4.5 \cdot |D'_i|} \leq 3\sqrt{4.5 \cdot |D|}$.

Case 2. G_i is an induced edge. Clearly, in this case, $\mathbf{bw}(G_i) \leq 3\sqrt{4.5 \cdot |D|}$.

Each graph G_i can be treated as a hypergraph with the ground set $V(G_i)$ and the edge set $E(G) \cup \{\{v\} \mid v \in V(G)\}$. As hypergraphs, graphs G_i have at most one edge (edge consisting of one vertex) in common, and by applying Lemma 3.1 recursively we obtain that $\mathbf{bw}(G) \leq \max\{1, \max_{1 \leq i \leq r} \mathbf{bw}(G_i)\} \leq 3\sqrt{4.5 \cdot |D|}$. □

4. Algorithmic consequences. In this section we discuss an algorithm that, given a planar graph G on n vertices and an integer k, decides whether G has a dominating set of size at most k.

4.1. The general algorithm. The algorithm runs in $O(2^{12.75\sqrt{k}} + n^3)$ steps and works in three phases as follows.

Phase 1. We use the known reduction of Planar Dominating Set problem to a linear problem kernel as a preprocessing procedure. Alber, Fellows, and Niedermeier [3] designed a procedure that, for a given integer k and planar graph G on n vertices, outputs a planar graph H on $\leq 335k$ vertices such that G has a dominating set of size $\leq k$ if and only if H has a dominating set of size $\leq k$. Later, Chen, Fernau, Kanj, and Xia [9] improved this result, providing a reduction to a kernel of a size $\leq 67k$. Each of the aforementioned reductions can be performed in $O(n^3)$ steps.

Phase 2. We compute an optimal branch decomposition of the graph H. For this step, one can use the algorithms due to Seymour and Thomas (algorithms 7.3 and 9.1 of sections 7 and 9 in [39]—for an implementation, see the work of Hicks in [33]). These algorithms need $O(n^2)$ steps for checking and $O(n^4)$ steps for constructing the branch decomposition for graphs on n vertices. We stress that there are no *large hidden constants* in the running time of these algorithms, which is important for practical applications. Thus a branch decomposition of H can be constructed in $O(k^4)$ steps. Check whether $\mathbf{bw}(H) \leq (3\sqrt{4.5})\sqrt{k} < 6.364\sqrt{k}$. If the answer is "no," then by Theorem 3.22 we conclude that there is no dominating set of size k in G. If the answer is "yes," then we proceed with the next phase.

Phase 3. Here we use a dynamic programming approach to solve the Planar Dominating Set problem on graph H. Alber et al. [1] suggested a dynamic programming algorithm based on the so-called monotonicity property of the domination problem. For a graph G on n vertices with a given tree decomposition of width ℓ, the algorithm of Alber et al. can be implemented in $O(2^{2\ell}n)$ steps. There is a well known transformation due to Robertson and Seymour [36] that, given a branch decomposition of width $\leq \ell$ of a graph with m edges, constructs a tree decomposition of width $\leq (3/2)\ell$ in $O(m^2)$ steps. Thus the result of Alber et al. immediately implies that

the DOMINATING SET problem on graphs with n vertices and m edges and of branch-width $\leq \ell$ can be solved in $O(2^{3\ell}n + m^2)$ steps. Notice now that for planar graphs $m = O(n)$. This phase requires $O(2^{3\cdot 3\sqrt{4.5\cdot k}}k+k^2)$ steps. As $3\cdot 3\sqrt{4.5} < 19.1$, we obtain an $O(2^{19.1\sqrt{k}}+n^3)$-step algorithm that finds in planar graph on n vertices a dominating set of size at most k, or reports that no such dominating set exists. However, in the next subsection (Theorem 4.1) we construct a dynamic programming algorithm solving the DOMINATING SET problem on graphs of branch-width $\leq \ell$ in $O(3^{1.5\ell}m)$ steps, where m is the number of edges in a graph. Because $(1.5\cdot\log_2 3)\cdot 3\sqrt{4.5} < 15.13$ and $m = O(k)$, we can reduce the cost of this phase to $O(2^{15.13\sqrt{k}})$ steps and conclude with a time $O(2^{15.13\sqrt{k}} + n^3)$ algorithm.

4.2. Dynamic programming on graphs of bounded branch-width. Let (T', τ) be a branch decomposition of a graph G with m edges and let $\omega' : E(T') \to 2^{V(G)}$ be the order function of (T', τ). We choose an edge $\{x, y\}$ in T', put a new vertex v of degree two on this edge, and make v adjacent to a new vertex r. By choosing r as a root in the new tree $T = T' \cup \{v, r\}$, we turn T into a rooted tree. For every edge of $f \in E(T) \cap E(T')$ we put $\omega(f) = \omega'(f)$. Also we put $\omega(\{x, v\}) = \omega(\{v, y\}) = \omega'(\{x, y\})$ and $\omega(\{r, v\}) = \emptyset$.

For an edge f of T we define E_f (V_f) as the set of edges (vertices) that are "below" f, i.e., the set of all edges (vertices) g such that every path containing g and $\{v, r\}$ in T contains f. With this notation, $E(T) = E_{\{v,r\}}$ and $V(T) = V_{\{v,r\}}$. Every edge f of T that is not incident to a leaf has two children that are the edges of E_f incident to f. We also denote by G_f the subgraph of G formed by edges of G corresponding to the leaves of V_f.

For every edge f of T we color the vertices of $\omega(f)$ in three colors:

black (represented by 1, meaning that the vertex is in the dominating set),

white (represented by 0, meaning that the vertex is dominated at the current step of the algorithm and is not in the dominating set), and

grey (represented by $\hat{0}$, meaning that at the current step of the algorithm we still have not decided to color this vertex white or black).

For every edge f of T we use mapping

$$A_f \colon \{0, \hat{0}, 1\}^{|\omega(f)|} \to \mathbb{N} \cup \{+\infty\}.$$

For a coloring $c \in \{0, \hat{0}, 1\}^{|\omega(f)|}$, the value $A_f(c)$ stores the minimum cardinality of a set $D_f \subseteq V(G_f)$ such that every nongrey vertex of G_f is dominated by a vertex from D_f and all black vertices are in D_f. More formally, $A_f(c)$ stores the minimum cardinality of a set $D_f(c)$ such that

- every vertex of $V(G_f) \setminus \omega(f)$ is adjacent to a vertex of $D_f(c)$,
- for every vertex $u \in \omega(f)$, $c(u) = 1 \Rightarrow u \in D_f(c)$ and $c(u) = 0 \Rightarrow (u \notin D_f(c)$ and u is adjacent to a vertex from $D_f(c))$.

We put $A_f(c) = +\infty$ if there is no such set $D_f(c)$. Because $\omega(\{r, v\}) = \emptyset$ and $G_{\{r,v\}} = G$, we have that $A_{\{r,v\}}(c)$ is the smallest size of a dominating set in G.

Let f be a nonleaf edge of T and let f_1, f_2 be the children of f. Define $X_1 = \omega(f) - \omega(f_2)$, $X_2 = \omega(f) - \omega(f_1)$, $X_3 = \omega(f) \cap (\omega(f_1) \cap \omega(f_2))$, and $X_4 = (\omega(f_1) \cup \omega(f_2)) - \omega(f)$.

Notice that $X_i \cap X_j \neq \emptyset$, $1 \leq i \neq j \leq 4$, and

$$\omega(f) = X_1 \cup X_2 \cup X_3. \tag{1}$$

Notice now that by the definition of ω it is impossible that a vertex belongs in exactly one of $\omega(f), \omega(f_1), \omega(f_2)$. Therefore, condition $u \in X_4$ implies that $u \in \omega(f_1) \cap \omega(f_2)$.

Hence

$$\omega(f_1) = X_1 \cup X_3 \cup X_4, \tag{2}$$

and

$$\omega(f_2) = X_2 \cup X_3 \cup X_4. \tag{3}$$

We say that a coloring c of $\omega(f)$ is *formed* from coloring c_1 of $\omega(f_1)$ and coloring c_2 of $\omega(f_2)$ if the following hold:

- **[F1]** For every $u \in X_1$, $c(u) = c_1(u)$.
- **[F2]** For every $u \in X_2$, $c(u) = c_2(u)$.
- **[F3]** For every $u \in X_3$, $(c(u) \in \{\hat{0}, 1\} \Rightarrow c(u) = c_1(u) = c_2(u))$ and $(c(u) = 0 \Rightarrow [c_1(u), c_2(u) \in \{\hat{0}, 0\} \wedge (c_1(u) = 0 \vee c_2(u) = 0)])$. (The color 1 ($\hat{0}$) can appear only if both colors in c_1 and c_2 are 1 ($\hat{0}$). The color 0 appears when both colors in c_1, c_2 are not 1 and at least one of them is 0.)
- **[F4]** For every $u \in X_4, (c_1(u) = c_2(u) = 1) \vee (c_1(u) = c_2(u) = 0) \vee (c_1(u) = 0 \wedge c_2(u) = \hat{0}) \vee (c_1(u) = \hat{0} \wedge c_2(u) = 0)$. This property says that every vertex u of $\omega(f_1)$ and $\omega(f_2)$ that does not appear in $\omega(f)$ (and hence does not appear further) should be finally colored either by 1 (if both colors of u in c_1 and c_2 are 1) or 0 (0 can appear if both colors of u in c_1 and c_2 are not 1 and at least one color is 0).

Notice that every coloring of f is formed from some colorings of its children f_1 and f_2. We start computations of values $A_f(c)$ from leaves of T. For every leaf f, $|\omega(f)| \leq 1$, and the number of colorings of $\omega(f)$ is at most three. Thus all possible values of $A_f(c)$ can be computed in $O(m)$ steps.

Then we compute the values of the corresponding functions in bottom-up fashion. The main observation here is that if f_1 and f_2 are the children of f, then the vertex sets $\omega(f_1)$, $\omega(f_2)$ "separate" subgraphs G_1 and G_2; thus the value $A_f(c)$ can be obtained from the information on colorings of $\omega(f_1)$ and $\omega(f_2)$. More precisely, let $\#_1(X_i, c)$, $1 \leq i \leq 4$, be the number of vertices in X_i colored by color 1 in coloring c. For a coloring c we assign

$$A_f(c) = \min\{A_{f_1}(c_1) + A_{f_2}(c_2) - \#_1(X_3, c_1) - \#_1(X_4, c_1) | c_1, c_2 \text{ form } c\}. \tag{4}$$

(Every 1 from X_3 and X_4 is counted in $A_{f_1}(c_1) + A_{f_2}(c_2)$ twice, and $X_3 \cap X_4 = \emptyset$.) The number of steps to compute the minimum in (4) is given by

$$O\Big(\sum_c |\{c_1, c_2\} \colon c_1, c_2 \text{ form } c|\Big).$$

Let $x_i = |X_i|$, $1 \leq i \leq 4$. For a fixed coloring c of $\omega(f)$, let p be the number of vertices of X_3 colored with 0. By [F3], every 0 of a vertex $u \in X_3$ can be "formed" in three ways, from $\hat{0}$ and 0, or from 0 and 0, or from 0 and $\hat{0}$. By [F4], a color of $u \in X_4$ can be obtained in four ways: 1 can be obtained from 1 and 1; 0 can be obtained either from 0 and 0, or from 0 and $\hat{0}$, or from $\hat{0}$ and 0. Then by [F1]–[F4], the number of colorings that form a fixed coloring c with exactly p vertices of X_3 of color 0 is equal to $3^p 4^{x_4}$. Every vertex of $\omega(f) = X_1 \cup X_2 \cup X_3$ can be colored in one of the three colors. The number of operations needed to estimate (4) for all possible colorings of $\omega(f)$ is

$$\sum_{p=0}^{x_3} 3^{x_1+x_2} \cdot 2^{x_3-p} \cdot 3^p \binom{x_3}{p} 4^{x_4} = 3^{x_1+x_2} 5^{x_3} 4^{x_4}.$$

The obtained bound can be reduced by using the trick due to Alber et al. [1]. The trick is based on the following observation. If for some coloring c of f we replace a color of a vertex u from $\hat{0}$ to 0, then for the new coloring c', $A_f(c) \leq A_f(c')$. Thus in (4) we can replace "c_1, c_2 form c" with "c_1 and c_2 satisfies **[F1]**, **[F2]**, **[F3′]**, and **[F4′]**," where **[F3′]** and **[F4′]** are as follows:

[F3′] For every $u \in X_3$, $(c(u) \in \{\hat{0}, 1\} \Rightarrow c(u) = c_1(u) = c_2(u))$ and $(c(u) = 0 \Rightarrow [c_1(u), c_2(u) \in \{\hat{0}, 0\} \wedge (c_1(u) \neq c_2(u))])$.

[F4′] For every $u \in X_4$, $(c_1(u) = c_2(u) = 1) \vee [c_1(u), c_2(u) \in \{\hat{0}, 0\} \wedge (c_1(u) \neq c_2(u))]$.

The purpose of properties **[F3′]** and **[F4′]** is to reduce the search space from all coloring forming c to the smaller set of colorings. Thus the number of steps for evaluating $A_f(c)$ is bounded by

$$\sum_{p=0}^{x_3} 3^{x_1+x_2} \cdot 2^{x_3-p} \cdot 2^p \binom{x_3}{p} 3^{x_4} = 3^{x_1+x_2} 4^{x_3} 3^{x_4}.$$

Let ℓ be the branch-width of G. By (1), (2), and (3),

$$\begin{aligned} x_1 + x_2 + x_3 &\leq \ell, \\ x_1 + x_3 + x_4 &\leq \ell, \\ x_2 + x_3 + x_4 &\leq \ell. \end{aligned} \tag{5}$$

The maximum value of the linear function $x_1 + x_2 + x_4 + x_3 \cdot \log_3 4$ subject to constraints (5) is $\frac{3\log_4 3}{2}\ell$. (This is because the value of the corresponding linear program achieves maximum in $x_1 = x_2 = x_4 = 0.5\ell$, $x_3 = 0$.) Thus

$$3^{x_1+x_2} 4^{x_3} 3^{x_4} \leq 4^{\frac{3\log_4 3}{2}\ell} = 3^{\frac{3\ell}{2}}.$$

It is easy to check that the number of edges in T is $O(m)$ and the number of steps needed to evaluate $A_{\{r,v\}}(c)$ is $O(3^{\frac{3\ell}{2}} m)$. Summarizing, we get the following theorem.

THEOREM 4.1. *For a graph G on m edges and given a branch decomposition of width $\leq \ell$, the dominating set of G can be computed in $O(3^{\frac{3\ell}{2}} m)$ time.*

5. Concluding remarks and open problems. We start this section with a discussion on the optimality of our results. We then give a presentation on several open problems and results that were motivated by this work.

5.1. Can Theorem 3.22 be improved? We have proved that for any planar graph with a dominating set of size $\leq k$, $\mathbf{bw}(G) \leq 3\sqrt{4.5 \cdot k} < 6.364\sqrt{k}$. The first of the multiplicative factors 3 follows from our results on the structure of planar graphs with a given dominating set in section 3. The second factor $\sqrt{4.5} \approx 2.121$ follows from [28] and is the bound on branch-width of planar graphs (Theorem 2.4). Any improvement to any of these two factors immediately implies an improvement to the time analysis of our fixed-parameter algorithm for a dominating set. However, our approach cannot be strongly improved because the upper bound of Theorem 3.22 is not far from the optimal.

LEMMA 5.1. *There exist planar graphs with a dominating set of size $\leq k$ and with branch-width $> 3\sqrt{k}$.*

Proof. Let G be a $(3n+2, 3n+2)$-grid for any $n \geq 1$. Let V' be the vertices of G of degree < 4. Let also V'' be the set of all vertices adjacent to V' in G. We define D as the unique $S \subseteq V(G) - V' - V''$, where $|S| = n^2$ and such that the

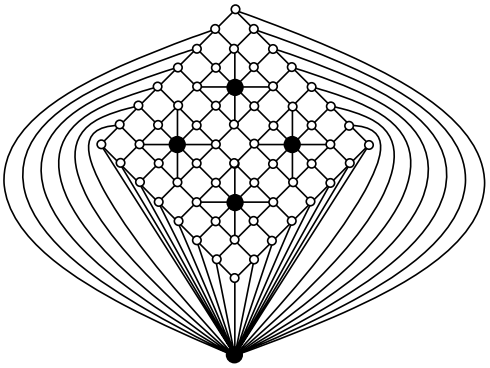


FIG. 13. *An example of the proof of Lemma* 5.1.

distance in G of all pairs $v, u \in D$ in G is a multiple of three. Then for any vertex $v \in D$, and for any possible cycle (square) (v, x, y, z, v) add the edge $\{x, z\}$. The construction is completed by connecting all the vertices in V' with a new vertex v_{new} (see Figure 13). We call the resulting graph J_n. Clearly, $D \cup \{v_{\text{new}}\}$ is a dominating set of J_n of size $k = n^2 + 1 \geq 2$. As the $(3n + 2, 3n + 2)$-grid is a subgraph of J_n we have that $\mathbf{bw}(J_n) \geq 3n + 2 \geq 3\sqrt{k-1} + 2 > 3\sqrt{k}$ (from [36], the (ρ, ρ)-grid has branch-width ρ). □

5.2. Open problems and extensions of our results. A (k, r)-*center* in a graph G is a set of at most k vertices, which we call centers, such that any vertex of G is within distance at most r from some center. Extending the results of section 4, [13] gives an algorithm that outputs, if it exists, a (k, r)-center of a planar graph in $r^{O(r\sqrt{k})} + n^{O(1)}$ steps (according to [13], the same result also holds for map graphs). The constants hidden in the first O-notation are based on an extension of Lemma 2.2, bounding the branch-width of any planar graph containing a (k, r)-center by $4(2r + 1)\sqrt{k} + O(r)$. We conjecture that this bound (and subsequently the running time of the algorithm in [13]) can be improved to $(2r + 1)\sqrt{4.5 \cdot k}$. We also suspect that a proof of this conjecture could be based on the same steps as those we used for Theorem 3.22.

An approach similar to the one of section 4 has been applied for a wide number of problems related to the PLANAR DOMINATING SET problem. In this way, our upper bound improves the algorithm complexity analysis for a series of problems when their inputs are restricted to planar graphs. As a sample we mention the following: INDEPENDENT DOMINATING SET, PERFECT DOMINATING SET, PERFECT CODE, WEIGHTED DOMINATING SET, TOTAL DOMINATING SET, EDGE DOMINATING SET, FACE COVER, VERTEX FEEDBACK SET, VERTEX COVER, MINIMUM MAXIMAL MATCHING, CLIQUE TRANSVERSAL SET, DISJOINT CYCLES, and DIGRAPH KERNEL (see [17] for details and extensions to more general graph classes). However, in all of the aforementioned problems, the time analysis is based on algorithms and combinatorial bounds for tree-width. It is an interesting problem whether better speed-up is possible using branch-width instead of tree-width, as we did in this paper. To our knowledge, not much progress has been noted so far on the design of algorithms on graphs of bounded branch-width (see [11, 13, 20]).

It appears that the planarity is not a limit for the existence of bounds like the one in Theorem 3.22. In [27], it was proved that for any D-dominated graph G,

$\mathbf{bw}(G) \leq 3(\sqrt{4.5}+2\sqrt{2 \cdot \mathbf{eg}(G)})\sqrt{|D| + 6 \cdot \mathbf{eg}(G)} = O(\sqrt{|D| \cdot \mathbf{eg}(G)}+\mathbf{eg}(G))$, where $\mathbf{eg}(G)$ is the Euler genus of G. The proof of this bound uses Theorem 3.22 as a basic ingredient. As a consequence of [27], most of the applications mentioned in the previous paragraph also can be extended for graphs of bounded genus. For discussions on the limits of this approach, see [26].

Finally, the idea behind Lemma 2.2 offers a mechanism for proving similar bounds for a wide family of parameters. This is the general family of *bidimensional parameters* introduced in [12] that unified the framework where the algorithmic paradigm of section 4 can be applied. Recent research on bidimensionality extends to several results such as [14, 15, 16, 18].

Acknowledgments. We are grateful to Hans Bodlaender, Ton Kloks, and Robin Thomas for answering our questions.

REFERENCES

[1] J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, and R. Niedermeier, *Fixed parameter algorithms for dominating set and related problems on planar graphs*, Algorithmica, 33 (2002), pp. 461–493.

[2] J. Alber, H. Fan, M. R. Fellows, H. Fernau, R. Niedermeier, F. A. Rosamond, and U. Stege, *Refined search tree technique for dominating set on planar graphs*, in Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science (MFCS 2001), Lecture Notes in Comput. Sci. 2136, Springer, Berlin, 2001, pp. 111–122.

[3] J. Alber, M. R. Fellows, and R. Niedermeier, *Polynomial-time data reduction for dominating set*, J. ACM, 51 (2004), pp. 363–384.

[4] J. Alber, H. Fernau, and R. Niedermeier, *Parameterized complexity: Exponential speed-up for planar graph problems*, J. Algorithms, 52 (2004), pp. 26–56.

[5] M. Alekhnovich and A. A. Razborov, *Satisfiability, branch-width and Tseitin tautologies*, in Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2002), IEEE Computer Society, 2002, pp. 593–603.

[6] N. Alon, P. Seymour, and R. Thomas, *Planar separators*, SIAM J. Discrete Math., 7 (1994), pp. 184–193.

[7] H. L. Bodlaender and D. M. Thilikos, *Graphs with branchwidth at most three*, J. Algorithms, 32 (1999), pp. 167–194.

[8] M. S. Chang, T. Kloks, and C. M. Lee, *Maximum clique transversals*, in Proceedings of the 27th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2001), Lecture Notes in Comput. Sci. 2204, Springer, Berlin, 2001, pp. 32–43.

[9] J. Chen, H. Fernau, I. A. Kanj, and G. Xia, *Parametric duality and kernelization: Lower bounds and upper bounds on kernel size*, in Proceeding of the 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS 2005), Lecture Notes in Comput. Sci. 3404, Springer, Berlin, 2005, pp. 269–280.

[10] W. Cook and P. D. Seymour, *An algorithm for the ring-routing problem*, Bellcore technical memorandum, Bellcore, Morristown, NJ, 1993.

[11] W. Cook and P. D. Seymour, *Tour merging via branch-decomposition*, INFORMS J. Comput., 15 (2003), pp. 233–248.

[12] E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos, *Subexponential parameterized algorithms on graphs of bounded genus and H-minor-free graphs*, J. ACM, 52 (2005), pp. 866–893.

[13] E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos, *Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs*, ACM Trans. Algorithms, 1 (2005), pp. 33–47.

[14] E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos, *Bidimensional parameters and local treewidth*, SIAM J. Discrete Math., 18 (2004), pp. 501–511.

[15] E. D. Demaine and M. T. Hajiaghayi, *Equivalence of local treewidth and linear local treewidth and its algorithmic applications*, Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA 2004), 2004, pp. 833–842.

[16] E. D. Demaine and M. T. Hajiaghayi, *Graphs excluding a fixed minor have grids as large as treewidth, with combinatorial and algorithmic applications through bidimensionality*, in Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005), 2005, pp. 682–689.

[17] E. D. Demaine, M. T. Hajiaghayi, and D. M. Thilikos, *Exponential speedup of fixed-parameter algorithms for classes of graphs excluding single-crossing graphs as minors*, Algorithmica, 4 (2005), pp. 245–267.

[18] E. D. Demaine, M. T. Hajiaghayi, and D. M. Thilikos, *The bidimensional theory of bounded-genus graphs*, in Proceedings of the 29th International Symposium on Mathematical Foundations of Computer Science (MFCS 2004), Lecture Notes in Comput. Sci. 3153, Springer, Berlin, 2004, pp. 191–203.

[19] F. Dorn, E. Penninkx, H. Bodlaender, and F. V. Fomin, *Efficient exact algorithms on planar graphs: Exploiting sphere cut branch decompositions*, in Proceedings of the 13th Annual European Symposium on Algorithms (ESA 2005), Lecture Notes in Comput. Sci. 3669, Springer, Berlin, 2005, pp. 95–106.

[20] F. Dorn and J. A. Telle, *Two birds with one stone: The best of branchwidth and treewidth with one algorithm*, in Proceedings of the 7th Latin American Theoretical Informatics Symposium (LATIN 2006), Lecture Notes in Comput. Sci. 3887, Springer, Berlin, 2006, pp. 386–397.

[21] R. G. Downey and M. R. Fellows, *Parameterized Complexity*, Springer, New York, 1999.

[22] H. Fernau and D. W. Juedes, *A geometric approach to parameterized algorithms for domination problems on planar graphs*, in Proceedings of the 29th International Symposium on Mathematical Foundations of Computer Science (MFCS 2004), Lecture Notes in Comput. Sci. 3153, Springer, Berlin, 2004, pp. 488–499.

[23] M. R. Fellows, *Parameterized complexity: The main ideas and some research frontiers*, in Proceedings of the 12th Annual International Symposium on Algorithms and Computation (ISAAC 2001), Lecture Notes in Comput. Sci. 2223, Springer, Berlin, 2001, pp. 291–307.

[24] J. Flum and M. Grohe, *The parameterized complexity of counting problems*, SIAM J. Comput., 33 (2004), pp. 892–922.

[25] F. V. Fomin and D. M. Thilikos, *Dominating sets in planar graphs: Branch-width and exponential speed-up*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2003), pp. 168-177.

[26] F. V. Fomin and D. M. Thilikos, *Dominating sets and local treewidth*, in Proceedings of the 11th Annual European Symposium on Algorithms (ESA 2003), Lecture Notes in Comput. Sci. 2832, Springer, Berlin, 2003, pp. 221–229.

[27] F. V. Fomin and D. M. Thilikos, *Fast parameterized algorithms for graphs on surfaces: Linear kernel and exponential speed-up*, in Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004), Lecture Notes in Comput. Sci. 3142, Springer, Berlin, 2004, pp. 581–592.

[28] F. V. Fomin and D. M. Thilikos, *New upper bounds on the decomposability of planar graphs*, J. Graph Theory, 51 (2006), pp. 53–81.

[29] J. F. Geelen, A. M. H. Gerards, and G. Whittle, *Branch-width and well-quasi-ordering in matroids and graphs*, J. Combin. Theory Ser. B, 84 (2002), pp. 270–290.

[30] Q.-P. Gu and H. Tamaki, *Optimal branch-decomposition of planar graphs in $O(n^3)$ time*, in Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP 2005), Lecture Notes in Comput. Sci. 3580, Springer, Berlin, 2005, pp. 373–384.

[31] G. Gutin, T. Kloks, C. M. Lee, and A. Yeo, *Kernels in planar digraphs*, J. Comput. System Sci., 71 (2005), pp. 174–184.

[32] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater, *Fundamentals of Domination in Graphs*, Marcel Dekker, New York, 1998.

[33] I. V. Hicks, *Planar branch decompositions.* I. *The ratcatcher*, INFORMS J. Comput., 17 (2005), pp. 402–412.

[34] P. Hliněný, *Branch-width, parse trees, and monadic second-order logic for matroids over finite fields*, in Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2003), Lecture Notes in Comput. Sci. 2607, Springer, Berlin, 2003, pp. 319–330.

[35] I. Kanj and L. Perković, *Improved parameterized algorithms for planar dominating set*, in Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science (MFCS 2002), Lecture Notes in Comput. Sci. 2420, Springer, Berlin, 2002, pp. 399–410.

[36] N. Robertson and P. D. Seymour, *Graph minors.* X. *Obstructions to tree-decomposition*, J. Combin. Theory Ser. B, 52 (1991), pp. 153–190.

[37] N. Robertson and P. D. Seymour, *Graph minors. XI. Circuits on a surface*, J. Combin. Theory Ser. B, 60 (1994), pp. 72–106.
[38] N. Robertson, P. D. Seymour, and R. Thomas, *Quickly excluding a planar graph*, J. Combin. Theory Ser. B, 62 (1994), pp. 323–348.
[39] P. D. Seymour and R. Thomas, *Call routing and the ratcatcher*, Combinatorica, 14 (1994), pp. 217–241.

SIAM J. COMPUT.
Vol. 36, No. 2, pp. 310–325

BETWEEN O(nm) AND O(n^α)*

DIETER KRATSCH† AND JEREMY SPINRAD‡

Abstract. This paper uses periodic matrix multiplication to improve the time complexities for a number of graph problems. The time for finding an asteroidal triple is reduced from O(nm) to O($n^{2.82}$), and the time for finding a star cutset, a two-pair, and a dominating pair is reduced from O(nm) to O($n^{2.79}$). It is also shown that each of these problems is at least as hard as one of three basic graph problems for which the best known algorithms run in times O(nm) and O(n^α). We note that the fast matrix multiplication algorithms do not seem to be practical because of the enormous constants needed to achieve the asymptotic time bounds. These results are important theoretically for breaking the n^3 barrier rather than giving efficient algorithms for a user.

Key words. algorithms, graphs, reductions, AT-free graphs, two-pair, star cutset, dominating pair

AMS subject classification. 05C85

DOI. 10.1137/S0097539704441435

1. Introduction. We will describe a method which takes an undirected graph $G = (V, E)$ as input and finds for all vertices $x \in V$ the connected components of $G - \mathrm{N}[x]$. Here $\mathrm{N}(x) = \{y \in V : \ (x, y) \in E\}$ denotes the open neighborhood, and $\mathrm{N}[x] = \mathrm{N}(x) \cup \{x\}$ denotes the neighborhood. We will call the set of connected components for $G - \mathrm{N}[x]$ for all $x \in V$ the *neighborhood-deleted components* of G. Clearly, the neighborhood-deleted components can be found in O(nm) time by simply computing for each x the components of $G - \mathrm{N}[x]$ using breadth-first search. Our algorithm will use fast matrix multiplication to reduce the time complexity on dense graphs to O($n^{2.79}$). The computation of the neighborhood-deleted components is the bottleneck step for a number of apparently unrelated algorithms on graphs. Thus we improve the best known running time for a collection of graph problems. We describe some of these applications below.

Three vertices x, y, z are an *asteroidal triple* in a graph if they are pairwise nonadjacent and between any two of them there is a path avoiding the neighborhood of the third. This concept was introduced by Lekkerkerker and Boland [18] to characterize interval graphs. It was also used by Gallai in his study of comparability graphs [14, 10]. Graphs without asteroidal triple, called *AT-free graphs*, were first studied as a separate class of graphs by Corneil, Olariu, and Stewart [6]. Many interesting properties and characterizations of AT-free graphs have been discovered [7, 19], and these are now recognized as an important graph class. It is not hard to see that x, y, z form an asteroidal triple iff x and y are in the same component of $G - \mathrm{N}[z]$, x and z are in the same component of $G - \mathrm{N}[y]$, and y and z are in the same component of $G - \mathrm{N}[x]$. Thus, the running time of algorithms for recognizing AT-free graphs was given as O(n^3). Early efforts to establish a linear time recognition algorithm for

*Received by the editors February 27, 2004; accepted for publication (in revised form) December 9, 2005; published electronically June 19, 2006. A preliminary version appeared in Proceedings of the 14th Annual SIAM-ACM Symposium on Discrete Algorithms [16].
http://www.siam.org/journals/sicomp/36-2/44143.html
†LITA, Université de Metz, 57045 Metz Cedex 01, France (kratsch@sciences.univ-metz.fr).
‡Department of EECS, Vanderbilt University, Nashville, TN 37235 (spin@vuse.vanderbilt.edu).
This author started research on this work while visiting the University of Metz and was supported by NSF grant 9820840.

AT-free graphs were dismissed after Spinrad showed that recognizing AT-free graphs is at least as hard as recognizing a triangle [21] (see our Theorem 5.1). We note that triangle recognition is a fundamental and well-studied graph problem and that its best known algorithm on dense graphs has running time O(n^α).[1] Using Gallai's concept of the knotting graph, Köhler [15] obtained a recognition algorithm for AT-free graphs with running time O($n^{2.82}$) plus the time to compute the neighborhood-deleted components. Thus we reduce the time bound for recognizing AT-free graphs to O($n^{2.82}$). It remains an interesting open question whether there is a recognition algorithm for AT-free graphs with running time O(n^β) such that $\alpha \leq \beta < 2.82$.

A *two-pair* in a graph is a pair of vertices x, y such that every chordless path between x and y has length two. Hayward [12] showed that every weakly chordal graph is either a clique or has a two-pair. Two-pairs are fundamental for recognition and optimization algorithms on weakly chordal graphs; they are also used as a tool for showing perfection in graph classes. We show that the neighborhood-deleted components can be used to reduce the time complexity of finding a two-pair in a graph.

Similar results can be obtained for problems on *dominating pairs*, *star cutsets*, and *extremities*; these terms will be defined in section 3.

We note that an earlier version of this paper [16] also gave an algorithm which used periodic matrix multiplication to improve the running time of an algorithm for finding a clique cutset in a graph. Since that time, we have discovered an improved algorithm for finding a minimal fill in a graph [17], which leads to a simpler algorithm achieving the same time bound for the clique cutset problem.

The problems solved in this paper all make use of fast matrix multiplication to beat bounds of O(nm). It is natural to ask whether these problems can be solved even faster (perhaps in linear time), or solved efficiently without using matrix multiplication. Motivated by the reduction from triangle recognition to AT-free graph recognition (Theorem 5.1), we show that each of the abovementioned problems is as hard as one of the following three basic graph problems:

- *determining whether a graph has a triangle*;
- *determining whether a graph has a simplicial vertex*, i.e., a vertex x such that N(x) is a clique;
- *determining whether a graph has a dominated vertex*, i.e., a vertex x such that for some y, N(x) $\subset$ N[y].

This gives evidence that significant improvements over our results, e.g., solving any of our problems in time o(n^α), or solving any of our problems in O($n^{3-\epsilon}$), $\epsilon > 0$, time without using matrix multiplication, would require a major breakthrough (such as, e.g., a faster algorithm for finding a triangle).

2. Neighborhood-deleted components. Before presenting our algorithm for finding the neighborhood-deleted components of G, we introduce some notation. Let $G = (V, E)$ be a graph. For $A \subseteq V$, we denote by $G[A]$ the subgraph of G induced by A. We write $G - A$ instead of $G[V - A]$. We denote by N(A) the set of all neighbors of A, i.e., N(A) = $\bigcup_{a \in A}$ N(a). A connected component C of a graph G is a maximal subset of vertices of G such that $G[C]$ is connected, thus components are vertex sets. We also call $B \subseteq V$ a *connected subset* of a graph $G = (V, E)$ if $G[B]$ is a connected graph. For standard graph theory notation we refer to [2, 11].

[1] α denotes the best known exponent of an algorithm to multiply two binary n by n matrices. Currently $\alpha = 2.376...$ [5].

To understand the motivation of several steps of our algorithm, we first discuss its fundamental idea. Suppose we have found a partition of $V - \mathrm{N}[v]$ into connected subsets of $G - \mathrm{N}[v]$ for all $v \in V$; initially we might have a partition into singletons. We can perform a single matrix multiplication and get all subset pairs which are joined by at least one edge when using a vertex-subset incidence matrix, as we will show later. However, if this is done from the initial condition, each $G - \mathrm{N}[v]$ may have $\Theta(n)$ components, and we would be performing matrix multiplication on two $\Theta(n)$ by $\Theta(n)$ matrices, which is too expensive.

Therefore, we want to reduce the number of initial connected subsets for each $G - \mathrm{N}[v]$. To do this, for each pair v, w we want to find a large enough set of edges $(u, w) \in E$ with $u \in \mathrm{N}(w) - \mathrm{N}[v]$ such that after using these edges only for merging connected subsets, w is guaranteed to be in a large component of $G - \mathrm{N}[v]$. The difficulty lies in finding $\mathrm{N}(w) - \mathrm{N}[v]$ without stepping through $\mathrm{N}[v]$ and $\mathrm{N}(w)$. Instead, we will use matrix multiplication to determine regions of a suitable matrix which contain at least one vertex of $\mathrm{N}(w) - \mathrm{N}[v]$, thus avoiding the $\mathrm{O}(n)$ expense of identifying $\mathrm{N}(w) - \mathrm{N}[v]$.

We now make the outline above more specific. We want to select a function $f(n)$, and for each pair of vertices v, w we will find either all vertices of $\mathrm{N}(w) - \mathrm{N}[v]$ if $|\mathrm{N}(w) - \mathrm{N}[v]| < f(n)$, or $f(n)$ vertices from $\mathrm{N}(w) - \mathrm{N}[v]$ otherwise. Let $\mathrm{MM}(i, j, k)$ be the time of an algorithm which multiplies an i by j matrix and a j by k matrix.

LEMMA 2.1. *Let G be a graph with n vertices, and let $f(n)$, $g(n)$ be functions of n. There is an $\mathrm{O}((n/g(n))\mathrm{MM}(n, g(n), n) + n^2 f(n) g(n))$ time algorithm to find for all pairs of vertices v, w of G a set $S_{v,w}$ such that $S_{v,w} \subseteq \mathrm{N}(w) - \mathrm{N}[v]$ and $|S_{v,w}| = \min\{f(n), |\mathrm{N}(w) - \mathrm{N}[v]|\}$.*

Proof. We describe the procedure to search for vertices of $\mathrm{N}(w) - \mathrm{N}[v]$ for all pairs v, w. Partition V into subsets of size $g(n)$ (possibly one subset will be of smaller size). Let S be a set of $g(n)$ vertices. For each vertex v in G, count the number of neighbors of v in S. Create an n by $g(n)$ binary matrix M_S with a row for each vertex of G and a column for each vertex of S, with $M_S[v, j] = 1$ iff vertex v of G is adjacent to the jth vertex of S. Let M'_S be the result of multiplying M_S by its transpose; thus $M'_S[v, w] = |\mathrm{N}(v) \cap \mathrm{N}(w) \cap S|$. Determine for each pair v, w whether there is any vertex of $\mathrm{N}(w) - \mathrm{N}[v]$ in S using $M'_S[v, w]$ and the number of neighbors of w in S; store this information for each set S.

Let v, w be a pair of vertices. Consider each set S in turn until we have found $f(n)$ vertices from $\mathrm{N}(w) - \mathrm{N}[v]$ or considered all sets S. If there is at least one vertex of $\mathrm{N}(w) - \mathrm{N}[v]$ in S, look at every $s \in S$, and add each vertex $s \in S$ to $S_{v,w}$ (initially empty) if $(s, w) \in E$, $(s, v) \notin E$, and $s \neq v$. At the end of this procedure for each pair of vertices v, w, clearly the set $S_{v,w}$ has $\min\{f(n), |\mathrm{N}(w) - \mathrm{N}[v]|\}$ vertices and $S_{v,w} \subseteq \mathrm{N}(w) - \mathrm{N}[v]$.

We now analyze the running time of the procedure. We first analyze the time needed to find whether $\mathrm{N}(w) - \mathrm{N}[v]$ is empty for all pairs of vertices v, w. The number of neighbors of a vertex v in S can be computed in $\mathrm{O}(|S|)$ time, so all these values, for all v and all S, can be computed in time $\mathrm{O}(n^2)$. The dominant cost of the procedure is finding the matrices M'_S for each S. Each of these $n/g(n)$ matrices is the result of multiplying an n by $g(n)$ matrix by a $g(n)$ by n matrix, giving us the first term in the statement of the theorem.

It takes time $\mathrm{O}(n^3/g(n))$ to determine for which sets S $\mathrm{N}(v) \cap \mathrm{N}(w) \cap S \neq \emptyset$. Since $n^3/g(n)$ is the size of the output matrices of $n/g(n)$ matrix multiplications as above, this term can be ignored with respect to order notation complexity of the algorithm. Then some of these sets S are checked to find actual vertices of $\mathrm{N}(w) - \mathrm{N}[v]$.

For each pair v, w, at most $f(n)$ sets are checked, taking O($g(n)$) time for a set; thus, the total time is O($n^2 f(n)g(n)$). □

LEMMA 2.2. *Let G be a graph with n vertices, and let $f(n)$ and $g(n)$ be functions of n. There is an* O($n/g(n)$MM$(n, g(n), n) + n^2 f(n)g(n) +$ MM$(n^2/f(n), n, n)$) *time algorithm to compute the neighborhood-deleted components of G.*

Proof. In phase 1 of our algorithm the procedure of Lemma 2.1 is used to compute the sets $S_{v,w}$. In phase 2, initial connected subsets for all $G - \mathrm{N}[v]$ are created as follows. For each v, start with each vertex of $G - \mathrm{N}[v]$ in its own connected subset and label each connected subset by the corresponding v. For each x in $S_{v,w}$, unify those connected subsets of $G - \mathrm{N}[v]$ containing x and w. At the end of phase 2, suppose that some connected subset C of $G - \mathrm{N}[v]$ contains fewer than $f(n)$ vertices. Then $|\mathrm{N}(c) - \mathrm{N}[v]| < f(n)$ and $\mathrm{N}(c) - \mathrm{N}[v] = S_{v,c}$ for every vertex c of C. Thus, there is no edge joining (a vertex of) C and a vertex of $V - (C \cup \mathrm{N}[v])$, and C is a connected component of $G - \mathrm{N}[v]$.

For the remainder, the algorithm treats only connected subsets of size at least $f(n)$. In phase 3, create a directed graph D with 3 levels of vertices as follows. At the first level of D, create a vertex for each connected subset of size at least $f(n)$ found in phase 2. At the second and third levels of D, create a vertex for each vertex in G. Add an edge from C at the first level to x at the second level if x is in C. Add an edge from x at the second level to y at the third level if x is adjacent to y. Now there is an edge joining a connected subset C and a vertex u in G iff there is a path from C in the first level of D to the copy of u in the third level of D. Let M_{CtoG} be a binary matrix with a row for each connected subset and a column for each vertex of G, with $M_{CtoG}[C, u] = 1$ iff $(u, v) \notin E$ (here v is the vertex that C is labeled with) and there is an edge joining C and u in G. Computing M_{CtoG} involves multiplying an $n^2/f(n)$ by n matrix representing the edges between the first two levels of D and an n by n matrix representing the edges between the second and third levels of D.

Recall that C is a connected subset of $G - \mathrm{N}[v]$ for some specific vertex v and that C is labeled by v. In phase 4, for every vertex u and every connected subset C, if there is an edge joining u and C in G, then unify C and the connected subset containing u in $G - \mathrm{N}[v]$. Since each vertex u is compared to $n^2/f(n)$ connected subsets there are $n^3/f(n)$ such operations; the cost of performing these union and find operations is dominated by the time for performing the matrix multiplication in phase 3. Therefore, the running time of the algorithm is O($n/g(n)$MM$(n, g(n), n) + n^2 f(n)g(n) +$ MM$(n^2/f(n), n, n)$). □

THEOREM 2.3. *The neighborhood-deleted components of G can be computed in* O($n^{2.79}$) *time.*

Proof. Let $f(n) = n^{.575}$, $g(n) = n^{.2125}$. It is clear that $n^2 f(n)g(n)$ is O($n^{2.79}$) for these values. It is known that MM$(n, g(n), n))$ is O($n^{2+o(1)}$) whenever $g(n) < n^{.294}$ [13, 4], so the $n/g(n)$ multiplications of n by $g(n)$ and $g(n)$ by n matrices take O($n^{2.79}$) time. The best algorithm for multiplying rectangular matrices of the form $n^2/f(n)$ by n and n by n is given in [13].

The time complexity from [13, p. 273] is somewhat complex to state. For chosen values of q, the exponent of MM$(n, n^{1.425}, n)$ is at most $(1/\log q)\log(2.425^{2.45}(q + 2)^{3.425}/3.425^{3.425})$. Using $q = 7$, as suggested on page 280 of [13], this works out to approximately 2.787, implying that the overall cost of the algorithm for computing neighborhood-deleted components is O($n^{2.79}$). □

3. Algorithmic consequences. We show that the neighborhood-deleted components of G can be used to solve a variety of well-known graph problems.

Our first application is recognition of AT-free graphs.

The *knotting graph* $\mathrm{K}(G)$ is defined as follows. For each vertex v of G and each connected component i of $\overline{G}[\mathrm{N}(v)]$ (i.e., the complement of the graph induced by $\mathrm{N}(v)$), there is a vertex v_i of $\mathrm{K}(G)$. If (u, v) is an edge of G, $\mathrm{K}(G)$ has an edge between the copy of u corresponding to the connected component containing v in $\overline{G}[\mathrm{N}(u)]$ and the copy of v corresponding to the connected component containing u in $\overline{G}[\mathrm{N}(v)]$.

Knotting graphs were first defined by Gallai [10], who showed that G is a comparability graph iff $\mathrm{K}(G)$ is bipartite.

COROLLARY 3.1. *AT-free graphs can be recognized in* $\mathrm{O}(n^{2.82})$ *time.*

Proof. Köhler [15] showed that you can recognize AT-free graphs in $\mathrm{O}(n^{2.82})$ time plus the time to compute a graph known as the knotting graph. Components of $\overline{G}[N(v)]$ correspond exactly to neighborhood-deleted components caused by deletion of neighbors of v in $\overline{G}$. Therefore, we can compute $\mathrm{K}(G)$ by finding neighborhood-deleted components in $\overline{G}$, and using our earlier theorem knotting graphs can be constructed in $\mathrm{O}(n^{2.79})$ time. □

An *extremity* of a graph is a vertex v such that $G - \mathrm{N}[v]$ is connected. Thus, knowing the neighborhood-deleted components, we can easily identify the extremities.

COROLLARY 3.2. *All extremities of a graph can be listed in time* $\mathrm{O}(n^{2.79})$.

A *star cutset* in a graph is a set S of vertices such that $G - S$ is disconnected, and there is a vertex s in S which is adjacent to every other vertex in S. Chvátal showed that a minimal imperfect graph cannot have a star cutset [3], and generalizations of this star cutset lemma played a crucial role in resolving the strong perfect graph conjecture. Since the generalizations, called skew partitions, can only be found using algorithms with incredibly high time complexity [8], algorithms can be simplified greatly on classes which can be decomposed using the simpler star cutset decomposition. Chvátal also gave a lemma which showed that star cutsets can be found in polynomial time; a star cutset corresponds either to a pair of neighbors x, y such that $\mathrm{N}(x)$ is a subset of $\mathrm{N}[y]$, or to a nonextremity.

COROLLARY 3.3. *It is possible to determine whether a graph has a star cutset in* $\mathrm{O}(n^{2.79})$ *time.*

Proof. Since we can determine all neighborhood containments $\mathrm{N}(x) \subseteq \mathrm{N}[y]$ for all pairs x, y in a graph from a single matrix multiplication, the time complexity of finding a star cutset is dominated by the time needed to find whether the graph has a nonextremity. □

A *dominating pair* in a graph is a pair x, y of vertices such that for every path P from x to y and every vertex z of G, z is adjacent to at least one vertex of P.

COROLLARY 3.4. *It is possible to determine whether* x, y *is a dominating pair in* $\mathrm{O}(n^{2.79})$ *time.*

Proof. The vertices x, y are a dominating pair exactly when x and y are not in the same connected component of $G - \mathrm{N}[z]$ for any z. □

The following theorem needs a more sophisticated use of the neighborhood-deleted components.

THEOREM 3.5. *All dominating pairs of* G *can be listed in* $\mathrm{O}(n^{2.79})$ *time.*

Proof. First, we note that all adjacent dominating pairs can be determined in $\mathrm{O}(n^2)$ time given the square of the adjacency matrix. The number of vertices adjacent to x or y for neighbors x, y is equal to $a(x, y) = |\mathrm{N}(x)| + |\mathrm{N}(y)| - A^2[x, y]$; the latter term is equal to the number of common neighbors of x and y. Neighbors x and y form a dominating pair iff $a(x, y) = n$.

In the remainder of this proof, we show how to determine whether nonadjacent vertices are dominating pairs.

Construct a graph G' with a vertex w for each vertex of G, and a vertex c for each component of all $G - \mathrm{N}[v]$. Add an edge from c to w iff w is in component c. Determining whether w and x are in a common component (and thus are not a dominating pair) is equivalent to asking whether there is a path from w to x of length 2 in this graph, which can be solved by matrix multiplication. However, if the number of components is large, then the cost of these multiplications is too high; we will reduce the number by solving the problem in a different way for components with a small number of vertices.

We will not optimize this step, since it takes less time than the bottleneck step of computing the components. For any component c with fewer than $n^{0.7}$ vertices, we step through all pairs of vertices w, x in c, and mark w, c in a matrix to indicate that these cannot form a dominating pair. On each vertex we spend at most $n^{1.7}$ time marking other vertices during this step (at most $n^{0.7}$ time for each $G - \mathrm{N}[v]$), so the time spent eliminating vertex pairs as candidates because they are in a common small component is $\mathrm{O}(n^{2.7})$. Vertices corresponding to components with fewer than $n^{0.7}$ vertices are then removed from G'.

Testing whether vertices are in a common large component is equivalent to testing for paths of length 2 in G', and can be solved using matrix multiplication of an n by $n^{1.3}$ matrix by an $n^{1.3}$ by n matrix, which takes $\mathrm{o}(n^{2.7})$ time.

Therefore, the bottleneck step for determining whether a graph has a dominating pair is the computation of neighborhood-deleted components of G, and this also takes $\mathrm{O}(n^{2.79})$ time. □

4. Two-pairs. A more sophisticated algorithm based on our fast algorithm for finding neighborhood-deleted components is an improved algorithm to list all two-pairs of a graph. The current best algorithm for finding either a single two-pair, or all two-pairs, in a graph is based on the following observation [1]: v, w is a two-pair of G iff $\mathrm{N}(w) \cap \mathrm{N}(v) = \mathrm{N}(C) \cap \mathrm{N}(v)$, where C is the connected component of $G - \mathrm{N}(v)$ containing w. This gave an $\mathrm{O}(nm)$ algorithm for finding all two-pairs of G.

Suppose that all neighborhood-deleted components of G are known. It is not hard to identify for all vertices v a subset of vertices S_v such that v, w form a two-pair implies $w \in S_v$. We call a vertex w a *candidate* for v if w has the maximum number of common neighbors with v from any vertex in the component of $G - \mathrm{N}[v]$ containing w. The number of common neighbors $|\mathrm{N}(v) \cap \mathrm{N}(w)|$ for each pair of vertices v, w can be determined from the square of the adjacency matrix of G. It is also not hard to see that if two vertices w, x in the same component are candidates, then v, w is a two-pair iff v, x is a two-pair. Assuming that we know the number of common neighbors of each pair of vertices (which takes $\mathrm{O}(n^\alpha)$ time), it takes $\mathrm{O}(n^2)$ time to find all candidates; simply step through each neighborhood-deleted component C of v and choose the vertex of C which has the most common neighbors with v. We may limit our further search to one candidate for each component of $G - \mathrm{N}[v]$. If the number of components is too large, the time complexity can become too large. Therefore, we must once again use balancing, finding the candidates in large components using one form of matrix multiplication, and using a different technique for small components.

THEOREM 4.1. *There is an* $\mathrm{O}(n^{2.79})$ *algorithm to list all two-pairs in* G.

Proof. Recall that each component C is associated with a vertex v, such that C is a connected component of $G - \mathrm{N}(v)$.

Let $f(n)$ be a function to be fixed later. For a candidate w in a small component

C of $G-\mathrm{N}[v]$, i.e., $|C| < f(n)$, test as follows whether v, w is a two-pair of G. Compute $|\mathrm{N}(w) \cap \mathrm{N}(w') \cap C|$ for all vertices $w' \in C$. This takes $\mathrm{O}(|C|^2)$ time for a component C. To test whether v, w is a two-pair of G, for each $w' \in C$ calculate the number of common neighbors of w and w' in $\mathrm{N}(v)$ (which is $|\mathrm{N}(w) \cap \mathrm{N}(w')| - |\mathrm{N}(w) \cap \mathrm{N}(w') \cap C|$). If there is a $w' \in C$ such that $|\mathrm{N}(v) \cap \mathrm{N}(w')| > |\mathrm{N}(w) \cap \mathrm{N}(w') \cap \mathrm{N}(v)|$, then v, w is not a two-pair of G, otherwise v, w is a two-pair of G.

To test candidates from large components C, i.e., $|C| \geq f(n)$, set up the following digraph D. At level 1 of D, make a copy of each vertex of G. At level 2, make a second copy of each vertex of G, with an edge from x at level 1 to y at level 2 in D iff x is adjacent to y in G. At level 3 of D, create a vertex for each component of size at least $f(n)$. Add an edge from a vertex at level 2 to a component at level 3 in D iff the vertex belongs to the component. Perform a single multiplication of an n by n matrix representing the edges between levels 1 and 2 of D by an n by $n^2/f(n)$ matrix representing the edges between levels 2 and 3 of D. (Note that there are at most $n/f(n)$ large components.) The resulting matrix tells you which vertices have a neighbor in component C for each component C; for a vertex u which is not in C, entry $[u, C]$ of the result matrix is nonzero iff u has an edge to a vertex in C. Create a matrix E by changing every nonzero value of the result matrix to 1.

Perform another matrix multiplication, multiplying the adjacency matrix of G with E. Entry v, C of this matrix tells you how many neighbors of v have an edge to component C, i.e., $|\mathrm{N}(v) \cap \mathrm{N}(C)|$.

Finally v, w is a two-pair in G for a candidate w belonging to a component C of $G - \mathrm{N}[v]$ iff $|\mathrm{N}(v) \cap \mathrm{N}(w)| = |\mathrm{N}(C) \cap \mathrm{N}(v)|$.

The total time for testing candidates from large components is $\mathrm{O}(n^{3.376}/f(n))$. The time for testing candidates from small components is the sum of the squares of the component sizes. Maximizing the sum of squares given a fixed sum of numbers is achieved by making the numbers as large as possible; thus the time spent on small components can be bounded by making each component size $f(n)$, taking time $f^2(n)$ on $n^2/f(n)$ components, for an overall bound of $\mathrm{O}(n^2 f(n))$. By choosing $f(n)$ from a particular range ($f(n) = n^{0.6}$ is one possible value), the time for finding all two-pairs given the components is smaller than the time to compute the neighborhood-deleted components of G. Hence all two-pairs of a graph can be listed in $\mathrm{O}(n^{2.79})$ time. □

5. Reductions. The problems solved in this paper all make use of fast matrix multiplication algorithms to beat bounds of $\mathrm{O}(nm)$. It is natural to ask whether these problems can be solved even faster (perhaps in linear time), or solved efficiently without using matrix multiplication. In this section, we give evidence that finding an $o(n^\alpha)$ time algorithm for solving any of the problems solved in this paper by using fast matrix multiplication, or solving them efficiently without using matrix multiplication in time $O(n^{3-\epsilon})$ for some $\epsilon > 0$, would require a major breakthrough. We do this by showing that certain simple, famous problems currently solved most efficiently by matrix multiplication can be reduced to our problems. This still leaves open the question as to whether the time complexity for any of these problems can be reduced to the time for performing a single matrix multiplication of n by n matrices.

Our first basic problem which can be solved using matrix multiplication is recognizing triangle-free graphs. It is easy to see that G has a triangle iff there is a pair of vertices x, y joined by both a path of length 2 and a path of length 1. Therefore, it is easy to determine whether a graph is triangle-free in $\mathrm{O}(n^2)$ time if we are given both the adjacency matrix and the square of the adjacency matrix. There is no known algorithm which solves the problem in $\mathrm{O}(n^{3-\epsilon})$ time for some $\epsilon > 0$ without using

matrix multiplication.

Our second basic problem which can be solved using matrix multiplication is determining whether there is a pair of vertices x, y such that $N(x) \subset N[y]$; we call this the problem of determining whether a graph has a dominated vertex. It is not hard to see that the dominated vertex problem can be solved using matrix multiplication; $N(x)$ is a subset of $N[y]$ iff the number of paths of length 1 or 2 from x to y is equal to the degree of x.

Our third basic problem is the problem of determining whether a graph has a simplicial vertex. This can be solved easily in $O(n^2)$ time if the cube of the adjacency matrix is given, since v is simplicial iff the number of paths of length 3 from v to v is equal to $|N(v)|(|N(v)| - 1)$.

Although no faster algorithm is known to solve the second and third problems, these are not as well studied as triangle-free recognition. For each of the problems Π solved in the previous sections of the paper, we will show that there is an $O(n^2)$ time reduction which takes a graph G as input and produces a graph G' with $O(n)$ vertices as output, where either G has a triangle iff G' has property Π, G has a dominated vertex iff G' has property Π, or G has a simplicial vertex iff G' has property Π. This will show that faster algorithms to solve problem Π may be hard to find, since they would imply faster algorithms for one of our fundamental problems.

THEOREM 5.1 (see [21]). *Testing whether a graph with $2n$ vertices has an asteroidal triple is at least as hard as testing whether a graph with n vertices has a triangle.*

Proof. We reduce the triangle finding problem to the asteroidal triple finding problem.

Let G be a graph with n vertices. Create a graph G' as follows. Create n vertices corresponding to vertices of G, where x and y are adjacent in G' iff x and y are nonadjacent in G. Add n new vertices, where each new vertex has degree $2n-2$, and new vertex i is nonadjacent only to the vertex of G' corresponding to the ith vertex of G; we will call this new vertex i'.

Suppose that G has a triangle x, y, z. Then x, y, z are pairwise nonadjacent in G', and the paths x, y', z; x, z', y; y, x', z show that G' has an asteroidal triple. Suppose that G' has an asteroidal triple. Since each new vertex (i.e., the vertices labeled v') of G' is nonadjacent only to one vertex of G', the independent vertices of this asteroidal triple in G' must correspond to a triangle in G. □

THEOREM 5.2. *Determining whether a graph with $3n+4$ vertices has a dominating pair (or whether a particular pair of vertices b, z is a dominating pair of a graph with $3n+4$ vertices) is at least as hard as determining whether a graph with n vertices has a triangle.*

Proof. Let G be a graph with n vertices. Create a graph G' with vertices x_1, x_2, x_3 for each x in G, and vertices b, c, y, z. Add edges from b to c, y to z, c to x_1 for all x, and x_3 to y for all x. For all $v, w \in V$, add edges (v_1, w_1), (v_2, w_2), and (v_3, w_3). Add edges between v_1 and w_2, v_2 and w_3 if v and w are adjacent in G and from v_1 and w_3 iff v and w are distinct and nonadjacent in G.

Suppose that G has an independent set v, w, x. Then the path b, c, v_1, x_3, y, z avoids all neighbors of w_2, so b, z is not a dominating pair. Suppose that b, z is not a dominating pair. Since any path from b to z which uses vertices of all 3 indices is a dominating path, there must be some path from b to z which uses an edge (v_1, x_3) such that there is a vertex w_2 being nonadjacent to v_1 and to x_3. Hence there is an independent set v, w, x in G.

Therefore, G has an independent set of size 3 iff b, z is not a dominating pair in G'. It is clear that if there is a dominating pair in G', then b, z is a dominating pair, so G' has a dominating pair iff G has no independent set of size 3. □

We note that it is easy to find a dominating pair in linear time in a graph which is known to have no asteroidal triples [7]. Since the above reductions show that neither AT-free recognition nor dominating pair recognition can be solved in linear time without a major breakthrough, it is an interesting open question as to whether there is a robust linear time algorithm for finding a dominating pair in an AT-free graph. Robust is used here in the sense of [20]; a robust linear time algorithm must take an arbitrary (i.e., not necessarily AT-free) graph, and should in linear time either find a dominating pair or answer that the graph is not AT-free. The difference from other algorithms is that if the input graph is not AT-free but has a dominating pair, such an algorithm is allowed to answer either that G has a dominating pair or that G is not AT-free.

THEOREM 5.3. *Identifying all extremities in a graph on $3n$ vertices is at least as hard as determining whether a graph with n vertices has a triangle.*

Proof. We reduce the triangle recognition problem to the question of identifying all extremities.

Let G be a graph with n vertices. Create 3 copies x_1, x_2, x_3 in G' for every x in G. Add edges from x_1 to y_2 and from x_2 to y_3 iff x is nonadjacent to y in G, or if $x = y$. Add edges from x_1 to y_3 iff x and y are adjacent in G. Add edges (x_1, y_1), (x_2, y_2), and (x_3, y_3) for all x, y.

We will show that x is in some triangle of G iff x_2 is an extremity of G'. Note that vertices of $G' - \mathrm{N}[x_2]$ correspond to u_1 and u_3 such that u is adjacent to x in G.

Suppose that x, y, z form a triangle in G. Since there is an edge from y_1 to z_3, and vertices with each index 1 and 3 form a clique, x_2 is an extremity of G'.

Suppose that x is not part of a triangle in G. Then there is no edge between vertices with index 1 and vertices with index 3 in $G' - \mathrm{N}[x_2]$; any such edge between neighbors of x would be a triangle in G. Therefore, x_2 is a nonextremity of G'. □

We note that the important special case of determining whether a graph has a nonextremity can be solved more efficiently than recognizing triangle-free graphs remains open.

THEOREM 5.4. *Testing whether a graph with $4n + 6$ vertices has a two-pair is at least as hard as testing whether a graph with n vertices has a dominated vertex.*

Proof. Let G be a graph with n vertices; we will assume that G has at least one edge, or finding a dominated vertex is trivial. Create a graph G' as follows. For each vertex x of G, create vertices x_1, x_2, x_3, x_4 in G'. G' will consist of two disconnected subgraphs, one containing each x_1 and x_2, the other containing x_3 and x_4. The two disconnected subgraphs of G' are constructed similarly, but differ in one detail.

Add an edge from x_1 to y_2 iff x and y are adjacent in G. Similarly, add an edge from x_3 to y_4 iff x and y are adjacent in G.

Add a vertex w_1 which is adjacent to every vertex x_1, and a vertex z_2 adjacent to every vertex y_2. Add a vertex v adjacent only to w_1 and z_2. Similarly, add a vertex w_3 which is adjacent to every vertex x_3, a z_4 which is adjacent to every vertex y_4, and a vertex u which is adjacent only to w_3 and x_4.

The constructions of the two subgraphs differ only in that we now add edges from x_3 to x_4 for every x in G; i.e., the two copies of a vertex are made adjacent in one subgraph, but not the other. Intuitively, one subgraph translates nonadjacent pairs $\mathrm{N}(x) \subset \mathrm{N}[y]$ into two-pairs, while the other translates adjacent pairs with the

property into two-pairs.

Suppose that G has a dominated vertex x, i.e., that $\mathrm{N}(x) \subset \mathrm{N}[y]$. If x and y are nonadjacent, then every neighbor of x_1 is a neighbor of y_1 in G', so x_1,y_1 form a two-pair in G'. If x and y are adjacent, then every neighbor of x_3 is adjacent to y_3, and x_3,y_3 forms a two-pair of G'.

Suppose that G' has a two-pair b', c'. Neither b' nor c' can be w_1; w_1 is adjacent to every vertex x_1, has a path w_1, v, z_2, y_2 to every vertex y_2, is adjacent to v, and has a path of length 3 to z_2 using any edge (x_1, y_2). Symmetrically, neither b' nor c' can be z_2, and similar arguments show that neither b' nor c' can be v, u, w_3, or z_4. It is also impossible to have $b' = x_1$, $c' = y_2$, since either x_1 is adjacent to y_2 or there is a chordless path x_1, w, v, z, y_2 between the vertices; the same reasoning shows that we cannot have $b' = x_3$, $c' = y_4$. Thus, we assume without loss of generality that the two-pair is of the form x_1, y_1 or x_3, x_4.

Suppose that $b' = x_1$ and $c' = y_1$. If there is a vertex i adjacent to x but not y and a vertex j adjacent to y but not x, then the path x_1, i_2, z_2, j_2, y_1 contradicts the assumption that x_1, y_2 is a two-pair. Similarly, two-pairs x_3, y_3 must correspond to dominated vertices, or we get a path from x_3 to y_3 using i_4, z_4, j_4.

Therefore, G' has a two-pair iff G has a dominated vertex. □

THEOREM 5.5. *Testing whether a graph on $8n + 2$ vertices has a star cutset is at least as hard as testing whether a graph G with n vertices has a dominated vertex.*

Proof. Let G be a connected graph with n vertices, such that all vertices have degree at least 2.

Our construction is similar to that of the previous theorem, in that we create two subgraphs, one to deal with adjacent pairs such that $\mathrm{N}(x) \subset \mathrm{N}[y]$, and the other when x and y are nonadjacent. In this construction, the two subgraphs are connected by a complete join between the vertex sets, rather than being disconnected. We give the construction for the first subgraph before discussing the modification for the second subgraph.

The first subgraph is created as follows. Create four vertices x_1, x_2, x_3, x_4 of G' for each vertex x of G. Add an edge from x_1 to y_2 in G' iff x and y are adjacent in G. Add an edge from x_2 to x_3 and from x_3 to x_4 for each x in G. Add edges (x_4, y_4) for every x, y. Add a single vertex w, with edges from w to x_1 for all x.

The second subgraph is created in the same fashion as the first, except that x_1 is adjacent to x_2 in the second subgraph. The entire graph G' is formed by adding all edges between the first and second subgraphs.

Suppose that $\mathrm{N}(x) \subset \mathrm{N}[y]$ in G. If x and y are nonadjacent, then G' contains the star cutset $\{y_1\} \cup \mathrm{N}(x_1)$. If x and y are adjacent, then G' contains the star cutset consisting of the copies of $\{y_1\} \cup \mathrm{N}(x_1)$ from the second subgraph. This is because if we exclude x_2 and y_2, x_1's neighborhood is contained in y_1's neighborhood in both subgraphs. If x and y are nonadjacent, then x_1 is not adjacent to either x_2 or y_2 in the first subgraph, so we get neighborhood containment, while if x is adjacent to y, then the copy of y_1 in the second subgraph is adjacent to both x_2 and y_2, and thus its neighborhood contains that of the corresponding x_1.

Suppose that G' has a star cutset. Then by Chvátal's theorem [3] G' has either a vertex which is dominated by a neighbor, or a nonextremity. It is not hard to see that no vertex can dominate a neighbor in G', so G' must contain a vertex v such that $G - \mathrm{N}[v]$ is disconnected. One cannot disconnect G' by removing neighbors of a vertex with index 3 or 4; all remaining vertices clearly have a path to w. One cannot disconnect G' by removing w, since all remaining vertices can reach a vertex

with index 4, and these vertices form a clique. G' cannot become disconnected by removing any vertex x_2; all remaining vertices with index 2 remain connected since they have paths to level 4, and all remaining vertices with index 1 remain connected through w. Since x_1 remains in $G' - \mathrm{N}[x_2]$ and has an edge to some y_2, $G' - \mathrm{N}[x_2]$ is connected.

Therefore, we can assume that G' can be disconnected by removing all neighbors of some vertex x_1. If there is no vertex y such that $\mathrm{N}[x] \subseteq \mathrm{N}(y)$, then every vertex of $G' - \mathrm{N}[x_1]$ has a path to a vertex with index 4 (any v_2 is adjacent to v_3 and v_3 to v_4, and any y_1 has an edge to some z_2 which is nonadjacent to x_1, and we can reach z_4 from z_2); since vertices with index 4 form a clique, $G' - \mathrm{N}[x_1]$ is connected. Therefore, if G' has a star cutset, G has a dominated vertex. □

In the conference version of this paper, we presented an algorithm using periodic matrix multiplication to determine whether a graph has a clique cutset. Since that time, we devised a new algorithm to find a minimal fill and a clique cutset in a graph. These algorithms became sufficiently different from the techniques used here (they were based on an implementation of Tarjan's LEX-M algorithm, and were unrelated to neighborhood-deleted components) so that the algorithms were put in a separate paper [17]. However, the evidence that the problem requires the use of matrix multiplication seems to be much more closely related to this paper than the algorithmic work in the other paper; we feel that it is more consistent to talk about difficulty of the clique cutset problem here.

THEOREM 5.6. *Determining whether a graph on $4n + 2$ vertices has a clique cutset is at least as hard as determining whether an n vertex graph has a simplicial vertex.*

Proof. Let $G = (V, E)$ be a connected n vertex graph. Create a graph $G' = (V', E')$ as follows. For every x in G, create vertices x_1, x_2, x_3, x_4 in G'. Add edges from x_1 to x_2 and from x_3 to x_4 for every x in G. Add edges (x_1, y_3) and (x_2, y_3) to G' iff x is adjacent to y in G. Add adjacent vertices b and z, and edges (b, x_1) and (x_4, z) for all x. Add edges (x_3, y_3) iff x and y are adjacent in G. Note that $\mathrm{N}_{G'}[x_1] = \mathrm{N}_{G'}[x_2] \cup \{b\}$ for each x in G.

Suppose that G has a simplicial vertex v. Then $K' = \{v_1\} \cup \{w_3 : (v, w) \in E\}$ is a clique, and a cutset since $\{v_2\}$ is a component of $G' - K'$.

Suppose that G' has a clique cutset. It is clear that no clique cutset contains one or more of the vertices b,z, or any x_4; the only cliques involving these vertices correspond to edges, and it is easy to see that removing any adjacent vertex pair cannot disconnect G'. Any $K' \subseteq \{x_2, x_3 : x \in V\}$ is not a cutset, since each remaining vertex w_2 would have an edge to w_1 and thus a path to b, while each remaining vertex w_3 has an edge to w_4 and thus to z. Therefore, any clique cutset contains a vertex v_1. The only vertex which could not have a path to b if neighbors of v_1 are removed is v_2, and this will have a path to b unless all w_3 adjacent to v_1 are removed. Therefore, the only possible clique cutset in G' corresponds to a simplicial vertex v and its neighbors in G. Thus G has a simplicial vertex. □

It would be nice if we could show that all these problems were at least as hard as a single problem, of which the most natural would seem to be determining whether a graph has a triangle, which would be used as general evidence of difficulty. Although we were not able to show that determining whether a graph has a simplicial vertex is as hard as recognizing triangle-free graphs, the following theorem is an attempt to show a relationship between triangle-free recognition and finding simplicial vertices.

THEOREM 5.7. *The problem of counting the number of simplicial vertices in a*

graph with $3n + 2$ vertices is at least as hard as determining whether a graph on n vertices has a triangle.

Proof. Let G be a graph on n vertices, none of which is isolated. Create three vertices x_1, x_2, x_3 for each vertex x in G. Add edges (w_1, x_2) and (w_2, x_3) iff w is adjacent to x in G, and add an edge (w_1, x_3) iff w is nonadjacent to x in G, or $w = x$. Add edges (w_1, x_1) and (w_3, x_3) for all vertices w, x of G. Add adjacent vertices b and z, where b is adjacent to all vertices x_1 of G' and z is adjacent to all vertices x_3.

Suppose that v is contained in a triangle v, w, x of G. Then v_2 is not simplicial in G', since w_1 is nonadjacent to x_3.

Suppose that v is not contained in a triangle of G. Then v_2 is simplicial in G', since all neighbors w_1 of v_2 are adjacent to all neighbors x_3 of v_2, and vertices of the same index form a clique.

No vertex x_1 is simplicial in G', since b and v_2 are nonadjacent for any neighbor v_2 of x_1. Similarly, no x_3 is simplicial in G', and clearly b and z are nonsimplicial.

Therefore, the number of simplicial vertices in G' is equal to the number of vertices which are not contained in a triangle of G', and G is triangle-free iff G' has n simplicial vertices. □

The final reduction is related to comments received about clique cutset decomposition. Some researchers seem to feel that clique cutsets are so closely tied to minimal elimination orderings that fast algorithms to compute a minimal elimination ordering would give fast algorithms for finding a clique cutset. The following reduction shows that beyond a certain speed, faster minimal elimination orderings are not the only issue in finding a clique cutset.

THEOREM 5.8. *Finding a clique cutset in a graph with $2n+2$ vertices is at least as hard as finding a simplicial vertex in an n vertex graph, even if a minimal elimination ordering is given as part of the input.*

Proof. Let G be an arbitrary graph. Let G' be a graph with two copies x_1, x_2 of each x in G, plus two nonadjacent vertices y, z such that y and z are adjacent to every vertex x_2. Let X_1 be the set of vertices of G' with index 1, and let X_2 be the vertices with index 2. v_1 is adjacent to w_2 iff v is adjacent to w in G, and v_2 is adjacent to w_2 iff v is adjacent to w in G. A minimal elimination ordering of G' can be formed by adding edges between every pair of vertices from X_2 (all fill edges (v, w) are unique chords of 4-cycles v, y, w, z in G'), and y, z, X_1, X_2 is a minimal elimination ordering of G'.

The only possible clique cutset of G' is the set of all neighbors of x_1 such that x is simplicial in G. We know a minimal elimination ordering of G', but any algorithm for finding a clique cutset in G' given this minimal elimination ordering would give a fast algorithm for finding a simplicial vertex in G. □

6. Comments on reductions. We begin this section by summarizing the results of the previous section.

Problems as hard as triangle-free graph recognition:

(1) Recognizing AT-free graphs.
(2) Determining whether a graph has a dominating pair.
(3) Determining whether a specific vertex pair u, v is a dominating pair.
(4) Identifying all extremities in a graph.
(5) Counting the number of simplicial vertices.

Problems as hard as testing for a dominated vertex:

(1) Determining whether a graph has a two-pair.
(2) Determining whether a graph has a star cutset.

Problems as hard as testing for a simplicial vertex:

(1) Determining whether a graph has a clique cutset.

Ideally, we would have a single problem P, and show that all these problems are as hard as problem P. Since triangle-free graph recognition seems more fundamental than the other problems, we might try to show that all the problems above are as hard as triangle-free graph recognition. We now explain why we believe it would be very difficult to show that all these problems were as difficult as determining whether a graph has a triangle. Fundamentally, this deals with incomparability of proof sizes with respect to the question of whether a graph does/does not satisfy a particular property.

Let us consider certificates which prove whether a graph does/does not have each of the properties in question. The measurements size of the certificate and time to verify that a property holds are often used interchangeably in the literature; we will phrase the issues in terms of size of the certificate.

There is a certificate of constant size which shows that a graph has a triangle. No deterministic certificate of size $o(n^{2.376})$ is known for showing that a graph has no triangle. We note that although there is a randomized $O(n^2)$ certificate showing that a graph has no triangle (one provides the square of the adjacency matrix, and multiply by random n-vectors to verify that the results are the same for your proposed A^2 times the vector and A times (A times the vector)) [9], this involves producing the square of the matrix as part of your certificate.

For each problem P, Certsize(P) will be a pair, consisting of the best times known for a deterministic certificate for yes instances of P and a deterministic certificate for no instances of P. Thus, Certsize(triangle-free) = ($O(1)$,$O(n^{2.376})$). We will order the pair so that the smaller of these two values comes first in the ordered pair.

Let us consider Certsize(P) for other problems P which were considered in this paper. For simplicity, we limit ourselves to the decision problems, as opposed to counting and enumeration problems.

There is a certificate of size $O(n)$ that a graph has an asteroidal triple (the paths between each pair of vertices in the triple), but no $o(n^{2.376})$ certificate is known for showing that a graph has no asteroidal triple. Thus Certsize(AT-free) = ($O(n)$,$\Omega(n^{3.376})$).

There is a certificate of size $O(n)$ showing that x, y is not a dominating pair; it is only necessary to check that a path from x to y has no neighbors of z. There is no known certificate of size $o(n^{2.376})$ proving that x, y is a dominating pair. For showing that G has no dominating pair, we do not know of any certificate of either yes or no of size $o(n^{2.376})$. Thus, Certsize(xy-dominating pair) = ($O(n)$,$\Omega(n^{2.376})$), while Certsize(Exists dominating pair) = ($\Omega(n^{2.376})$,$\Omega(n^{2.376})$).

For all these problems P, Certsize(triangle-free) $\leq$ Certsize(P), in the sense that both the first and second values of the ordered pair Certsize(triangle-free) were less than or equal to the corresponding values for Certsize(P). In all of these cases, we were able to find a reduction showing that these problems were as hard as recognizing triangle-free graphs.

We now consider some of the problems which we chose to reduce from existence of a dominated vertex or existence of a simplicial vertex rather than triangle-free recognition. First, let us examine the "basic" problems themselves with respect to certificate size.

There is a certificate of size $O(n)$ showing that G has a dominated vertex; given x and y, it takes only $O(n)$ time to verify that $N(x) \subset N[y]$. There is also an $O(n^2)$

certificate showing that a graph has no dominated vertex; for each x, y, give a neighbor of x but not y and a neighbor of y but not x. Thus, Certsize(dominated vertex) = (O(n),O(n^2)).

There is a certificate of size O(n^2) that G contains a simplicial vertex (the simplicial vertex plus the list of all edges between its neighbors) and a certificate of size O(n^2) that G has no simplicial vertex (a pair of nonadjacent neighbors of each vertex). Thus, Certsize(simplicial vertex) = (O(n^2),O(n^2)).

The star cutset problem also has certificates of both yes and no of size O(n^2). One certificate is obvious; to verify that G has a star cutset, we verify that a set is a star and that the vertices not in the cutset induce a disconnected graph. Verifying that the graph has no star cutset uses Chvátal's theorem on star cutsets being either nonextremities or neighbors such that one neighborhood dominates the other; we give vertices in $\mathrm{N}[x] - \mathrm{N}[y]$ and $\mathrm{N}[y] - \mathrm{N}[x]$ for each pair of adjacent vertices x, y, and we show that each vertex is an extremity by providing a spanning tree for $G - \mathrm{N}[v]$ for each v. Thus, Certsize(star cutset) = (O(n^2),O(n^2)).

Showing that x, y form a two-pair can be done in O(n^2) time, by showing that there is no path from x to y in $G - (\mathrm{N}(x) \cap \mathrm{N}(y))$. An O($n^2$) certificate that a graph has no two-pair can also be given, though since it is more involved we will simply outline it here. For each v, one can provide an O(n) certificate that v is not part of a two-pair. A spanning tree is given for each connected component of $G - \mathrm{N}(v)$. For each nonneighbor w of v, give an edge from some x in the same component as w to a nonneighbor of w which is adjacent to both x and v. It is not difficult to argue that such an x exists iff v, w is not a two-pair. Therefore, Certsize(two-pair) = (O(n^2),O(n^2)).

Similarly, Certsize(clique cutset) = (O(n^2),O(n^2)), though this is not obvious unless you are familiar with an algorithm such as that of [24] for constructing a clique cutset.

In all these cases, Certsize(triangle-free) is incomparable with Certsize(P), in the sense that the first value in the pair was smaller for Certsize(triangle-free), while the second value was smaller for Certsize(P).

These observations about certificate sizes made us realize that it would be very difficult to find an O(n^2) reduction from triangle-free graph recognition to the problems of finding a clique cutset, star cutset, or a two-pair in a graph. Suppose that such a reduction existed for one of these problems; for example, assume that there was an O(n^2) reduction which took an n vertex graph G and produced an O(n) vertex graph G' such that G contains a triangle iff G' contains a two-pair. Any such reduction would give us an O(n^2) proof that a graph contains no triangle, since this would be equivalent to proving that G' has no two-pair, and we gave a simple proof of this fact above. Therefore, finding a reduction from recognizing two-pair free graphs would involve a major breakthrough, and the same would be true if two-pair was replaced by clique cutset or star cutset.

Therefore, for each of our problems solved, we either showed the problem was as hard as triangle-free recognition or are able to show using certificate sizes that such a reduction is likely to be difficult to find.

We were left with the problem of showing difficulty of the problems star cutset and existence of a two-pair. We introduce as a new concept the problems of being as hard as finding a simplicial vertex or as hard as finding a dominated vertex. There seems to be no particular reason to prefer one over the other as a basic problem for evidence of difficulty. We also would like to raise the general issue of which problems

that have $O(n^2)$ yes and no certificates, but can be solved using matrix multiplication, are appropriate to use as evidence of difficulty of beating matrix multiplication.

Finally, we would like to note that we also believe it will be difficult to find reductions in either direction between triangle-free recognition and dominated vertex/simplicial vertex existence. We explain above why we believe that reducing triangle-free recognition to the other problems is hard. However, reductions in the other direction may also prove to be difficult, since there is a constant size certificate that a graph contains a triangle. Suppose we had a reduction from G to G' such that G contains a two-pair iff G' is triangle-free. It would be possible to show that G has no two-pair by showing a triangle in G', although it is impossible to prove that G has no two-pair in constant time. Thus, any such reduction cannot be "local"; by this, we mean that there must be vertex pairs x, y of G' such that adjacency between x and y in G' cannot be determined from G in constant time. Such transformations cannot be ruled out but would necessarily be more complex than those that generally come to mind when the term reduction is used, and reductions in this direction will not be as simple as the reduction used in this paper.

It might be possible, however, to find reductions in one or both directions between finding a dominated vertex and finding a simplicial vertex. It might also be possible to find reductions from one of these problems to such problems as finding an asteroidal triple for which we chose a reduction from triangle-free recognition. In our opinion, it is perhaps too soon to consider such reductions; while it seems to be generally agreed upon that recognizing triangle-free graphs in less time than matrix multiplication is hard, we would like to find a community consensus on fundamental hard problems with short certificates before producing too many reductions which might not convince other researchers of the hardness of a problem.

Acknowledgment. We would like to thank one of the referees for a particularly careful reading of the manuscript, leading to a number of improvements in the paper.

REFERENCES

[1] S. R. Arikati and C. P. Rangan, *An efficient algorithm for finding a two-pair, and its applications*, Discrete Appl. Math., 31 (1991), pp. 71–74.
[2] A. Brandstädt, V. B. Le, and J. P. Spinrad, *Graph Classes: A Survey*, SIAM, Philadelphia, 1999.
[3] V. Chvátal, *Star-cutsets and perfect graphs*, J. Combin. Theory Ser. B, 39 (1985), pp. 189–199.
[4] D. Coppersmith, *Rectangular matrix multiplication revisited*, J. Complexity, 13 (1997), pp. 42–49.
[5] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, J. Symbolic Comput., 9 (1990), pp. 251–280.
[6] D. G. Corneil, S. Olariu, and L. Stewart, *Asteroidal triple-free graphs*, SIAM J. Discrete Math., 10 (1997), pp. 399–430.
[7] D. G. Corneil, S. Olariu, and L. Stewart, *Linear time algorithms for dominating pairs in asteroidal triple-free graphs*, SIAM J. Comput., 28 (1999), pp. 1284–1297.
[8] C. M. H. de Figueiredo, S. Klein, Y. Kohayakawa, and B. A. Reed, *Finding skew partitions efficiently*, J. Algorithms, 37 (2000), pp. 505–521.
[9] R. Freivalds, *Fast Probabilistic Algorithms*, Lecture Notes in Comput. Sci. 74, Springer, Berlin, 1979, pp. 57–69.
[10] T. Gallai, *Transitiv orientbare Graphen*, Acta Math. Acad. Sci. Hungar., 18 (1967), pp. 25–66.
[11] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.
[12] R. B. Hayward, *Weakly triangulated graphs*, J. Combin. Theory Ser. B, 39 (1985), pp. 200–208.
[13] X. Huang and V. Y. Pan, *Fast rectangular matrix multiplication and applications*, J. Complexity, 14 (1998), pp. 257–299.

[14] D. Kelly, *Comparability graphs*, in Graphs and Order, NATO Adv. Sci. Inst. Ser. C Math Phys. Sci. 147, I. Rival, ed., Reidel, Dordrecht, 1985, pp. 3–40.
[15] E. Köhler, *Recognizing graphs without asteroidal triples*, in Graph-Theoretic Concepts in Computer Science (Konstanz, 2000), Lecture Notes in Comput. Sci. 1928, Springer, Berlin, 2000, pp. 255–266.
[16] D. Kratsch and J. Spinrad, *Between* O(nm) *and* O(n^α), in Proceedings of the 14th Annual SIAM-ACM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2003, pp. 709–716.
[17] D. Kratsch and J. Spinrad, *Minimal fill in* O($n^{2.69}$) *time*, Discrete Math., 306 (2006), pp. 366–371.
[18] C. G. Lekkerkerker and J. Ch. Boland, *Representation of a finite graph by a set of intervals on the real line*, Fund. Math., 51 (1962/1963), pp. 45–64.
[19] R. H. Möhring, *Triangulating graphs without asteroidal triples*, Discrete Appl. Math., 64 (1996), pp. 281–287.
[20] V. Raghavan and J. Spinrad, *Robust algorithms for restricted domains*, in Proceedings of the 12th Annual SIAM-ACM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2001, pp. 460–467.
[21] J. P. Spinrad, *Efficient Graph Representations*, AMS, Providence, RI, 2003.
[22] R. E. Tarjan, *Decomposition by clique separators*, Discrete Math., 55 (1985), pp. 221–232.
[23] M. Thorup, *Decremental dynamic connectivity*, J. Algorithms, 33 (1999), pp. 229–243.
[24] S. H. Whitesides, *An algorithm for finding clique cut-sets*, Inform. Process. Lett., 12 (1981), pp. 31–32.

SIAM J. COMPUT.
Vol. 36, No. 2, pp. 326–353

CERTIFYING ALGORITHMS FOR RECOGNIZING INTERVAL GRAPHS AND PERMUTATION GRAPHS*

DIETER KRATSCH†, ROSS M. MCCONNELL‡, KURT MEHLHORN§, AND JEREMY P. SPINRAD¶

Abstract. A *certifying algorithm* for a problem is an algorithm that provides a certificate with each answer that it produces. The certificate is a piece of evidence that proves that the answer has not been compromised by a bug in the implementation. We give linear-time certifying algorithms for recognition of interval graphs and permutation graphs, and for a few other related problems. Previous algorithms fail to provide supporting evidence when they claim that the input graph is not a member of the class. We show that our certificates of nonmembership can be authenticated in $O(|V|)$ time.

Key words. certificates, certifying algorithms, interval graphs, permutation graphs

AMS subject classifications. 68W40, 05C85, 68N30

DOI. 10.1137/S0097539703437855

1. Introduction.

A recognition algorithm is an algorithm that decides whether some given input (graph, geometrical object, picture, etc.) has a certain property. Such an algorithm *accepts* the input if it has the property or *rejects* it if it does not. A *certifying algorithm* for a decision problem is an algorithm that provides a certificate with each answer that it produces. The certificate is a piece of evidence that proves that the answer has not been compromised by a bug in the implementation.

We give linear-time certifying algorithms for recognition of interval graphs and permutation graphs. Previous algorithms fail to provide supporting evidence of nonmembership. We show that our certificates of nonmembership can be authenticated in $O(n)$ time, where n is the number of vertices.

A familiar example of a certifying recognition algorithm is a recognition algorithm for bipartite graphs that computes a 2-coloring for bipartite input graphs and an odd cycle for nonbipartite input graphs. A more complex example is the linear-time planarity test which is part of the library of efficient data structures and algorithms (LEDA) system [19, section 8.7]. It computes a planar embedding for planar input graphs and a *Kuratowski subgraph* (a subdivision of K_5 or $K_{3,3}$) for nonplanar input graphs.

Certifying versions of recognition algorithms are highly desirable in practice; see [28, 20, 21] and [19, section 2.14] for general discussions on result checking. Consider a planarity testing algorithm that produces a planar embedding if the graph is planar, and simply declares it nonplanar otherwise. Though the algorithm may have been proven correct, the implementation may contain bugs. When the algorithm

*Received by the editors November 23, 2003; accepted for publication (in revised form) August 12, 2005; published electronically June 19, 2006.
http://www.siam.org/journals/sicomp/36-2/43785.html

†Université de Metz, LITA, 57045 Metz Cedex 01, France (kratsch@sciences.univ-metz.fr).
‡Computer Science Department, Colorado State University, Fort Collins, CO 80523-1873 (rmm@cs.colostate.edu).
§Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany (mehlhorn@mpi-sb.mpg.de).
¶Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN 37235 (spin@vuse.vanderbilt.edu).

declares a graph nonplanar, there is no way to check whether it did so because of a bug.

Given the reluctance of practitioners to assume on faith that a program is bug-free, it is surprising that the theory community has often ignored the question of requiring an algorithm to certify its output, even in cases when the existence of adequate certificates is well known.

An *authentication algorithm* is an algorithm that check the validity of the certificate. In contrast to the case of an algorithm that solves the problem from scratch, an authentication algorithm may reject the certificate if it is invalid, leaving open the answer to the original problem. However, it must be the case that the authentication algorithm provides a correct answer to the original problem if the certificate is valid, and never declares an incorrect answer to the problem due to an invalid certificate. The certificate has practical value when the algorithm for solving the problem from scratch is difficult but the authentication algorithm is trivial.

For instance, in the case of planarity testing, the authentication algorithm cycles through the edges of the Kuratowski subgraph, verifying that they are, in fact, present in the graph and form a Kuratowski subgraph as claimed. Though planarity testing in linear time is a complicated problem, checking a Kuratowski subgraph is a trivial task, especially if the edges are supplied in a convenient order by the certificate.

The notion of a certifying algorithm is related to the concept of run-time checking. Typically, run-time checking has been applied to very small fragments of code; for example, a program to calculate square roots might check at run-time that the square of the result equals the input value. An attempt to create a corresponding model of run-time checking for more complex algorithms is given in [28] with motivations which are very similar to our own. The authors conceive of a method for checking whether a (possibly complex) program that computes a function has produced a correct output. Their model does not require an algorithm to provide supporting evidence with its answer, but subsequent calls to it are allowed in order to present it with nondeterministically generated challenges after it has already returned a solution. If it has answered incorrectly, it can pass with only a certain probability of success. Confidence in the answer to within any desired probability can be attained through a sufficiently large set of challenges.

An important distinction between run-time checking and certifying algorithms is that run-time checking is generally conceived as being separate from the process of producing the output, while certifying algorithms produce proofs of correctness together with their output.

To clarify the difference between these two approaches, let us consider two valid, but opposing, views on proofs of output correctness. One can take the view that it is desirable to have a mechanism that uses as little information as possible in verifying the correctness of the output, preferably only the standard format of the solution. This approach has the advantage of enabling a user to check, using a single program, the correctness of the output of any program that has been designed to solve the problem.

Papers such as [1] have devised clever mechanisms of this form for verifying that the output of any program that solves a diverse set of problems, including such problems as graph isomorphism and equivalence search, is correct. If such a mechanism exists, one can take the position that there is no reason for the creator of the program to work on a technique for verifying that the output is correct, since any errors will be caught in a separate checking phase.

We instead want to shift the burden of creating proof of correctness to the software

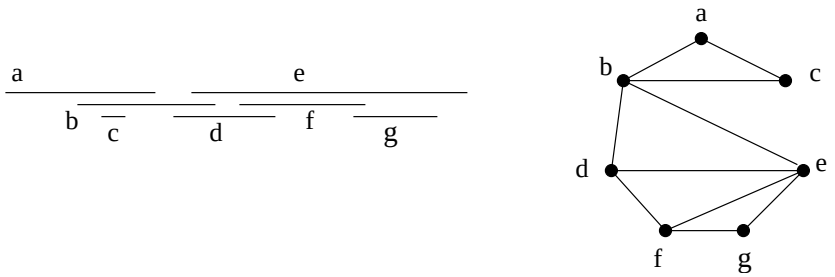


FIG. 1. *An interval graph is the intersection graph of a set of intervals on a line. The graph has one vertex for each of the intervals, and two vertices are adjacent if the corresponding intervals intersect.*

engineering phase as part of the design of the algorithm for solving the problem upon which to base the program. In finding the solution to the problem, the algorithm often goes through steps that can be used to create a simple certificate of correctness of the output. We believe that it should be part of the routine process of algorithm design to look for this certificate, and provide one wherever possible, rather than leaving this to a separate checking phase.

An interval graph is the intersection graph of intervals on a line (see Figure 1). That is, each vertex corresponds to an associated interval, and two vertices are adjacent iff the corresponding intervals intersect. The intervals constitute an *interval model* of the graph. Interval graphs have applications in molecular biology and scheduling.

Several linear-time recognition algorithms for interval graphs are known [2, 4, 9, 11, 12, 13, 18]. These graphs come up in the context of a variety of problems in scheduling and molecular biology; see [8, 24] for surveys. When a graph is an interval graph, these algorithms produce a certificate in the form of an interval model. When a graph is not an interval graph, none of these algorithms provides a certificate. However, the existence of certificates of rejection in the form of a forbidden substructure characterization is also well known. We extend the linear-time algorithm of Korte and Möhring [13] so that it also produces a certificate when a graph is not an interval graph; see section 5.

A permutation graph is the graph of inversions in a permutation. That is, each vertex corresponds to an element of the ground set of a permutation, and two vertices are adjacent iff the permutation reverses the relative order of the two corresponding elements (see Figure 2). If a graph is a permutation graph, we may show this by giving a *permutation model*, which consists of two linear orderings $(v_1, v_2, \ldots, v_n)$ and $(v_{\pi(1)}, v_{\pi(2)}, \ldots, v_{\pi(n)})$ of the vertices, such that two vertices v_i and v_j are adjacent iff v_i is before v_j in exactly one of the orderings.

The only previous linear-time algorithm for recognizing permutation graphs is given in [18]. This algorithm produces a permutation model if the graph is a member of the class, and presents its failure to produce such a model as the only evidence that a graph is not a member of the class. We give a linear-time algorithm that produces a certificate also in the case of nonmembership; see section 6. The algorithm is based on the following connection to comparability graphs. A graph G is a permutation graph iff G and its complement $\overline{G}$ are comparability graphs.

A dag is *transitive* if, whenever (a, b) and (b, c) are directed edges of the dag, (a, c) is also a directed edge. A transitive dag is a graphical representation of a

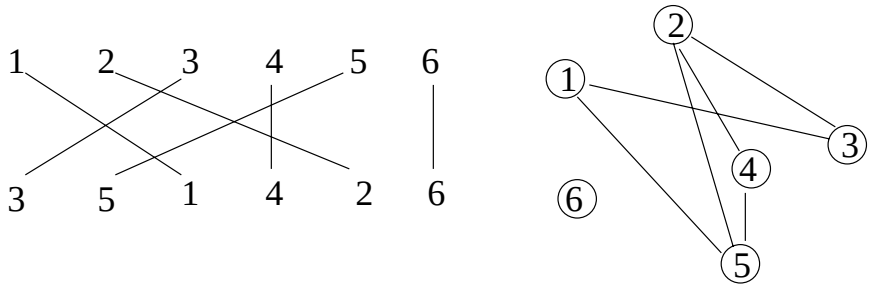


FIG. 2. *When a permutation acts on a sequence of elements, the corresponding permutation graph has one vertex for each of the elements. Two vertices are adjacent if the permutation swaps the relative order of the two elements.*

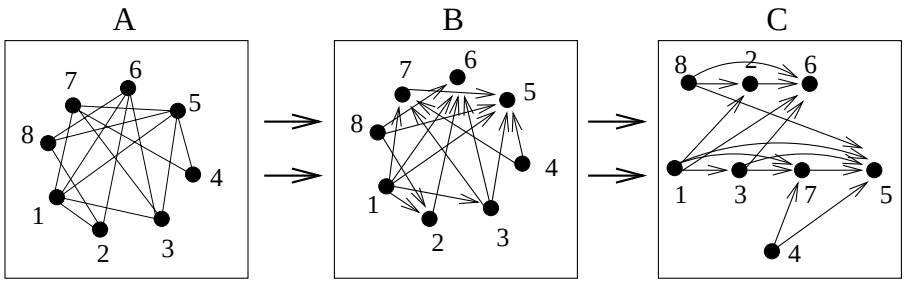


FIG. 3. *A comparability graph is a graph whose edges can be oriented so that the resulting digraph is acyclic and transitive. Part A depicts a comparability graph, part B is a transitive orientation, and part C is a redrawing of part B that makes it easy to see that the result is transitive.*

partial order (poset) relation. A graph is a *comparability graph* if orientations can be assigned to its edges so that the resulting digraph is a transitive dag (see Figure 3). Such an orientation is called a *transitive orientation*. A transitive orientation of a comparability graph can be found in linear time [18].

Gallai gave a forbidden substructure characterization of comparability graphs [7]. Since a graph G is a permutation graph iff G and $\overline{G}$ are comparability graphs, this forbidden structure in G or in $\overline{G}$ serves as a certificate that a graph is not a comparability graph. We do not know how to obtain this certificate in linear time for noncomparability graphs; no linear-time recognition algorithm is known. However, we show that when G is not a permutation graph, we may produce this certificate for G or for $\overline{G}$ in linear time. The algorithm fails to give a linear-time certifying algorithm for comparability graphs only because we cannot control whether the algorithm will find the certificate in G or in $\overline{G}$ when both G and $\overline{G}$ fail to be comparability graphs.

Certifying algorithms also exist for optimization problems; linear programming duality is a prime example. Primal and dual optimal solutions of a linear program certify each other. A *proper coloring* of a graph is a coloring of the vertices such that no adjacent vertices have the same color, and it is *minimum* if there is no proper coloring that uses fewer colors. A linear-time algorithm is known for finding a maximum clique and a minimum proper coloring of a comparability graph [18]. In a comparability graph, the size of the maximum clique is always the same as the minimum number of colors in a proper coloring. Since the vertices of a clique must all be colored differently in any proper coloring, the clique is a certificate of minimality of the coloring, and the coloring is a certificate that the clique is of maximum size. We give an algorithm that

provides a certificate that the input graph is not a comparability graph whenever this algorithm fails to find the desired coloring and clique.

2. Preliminaries. We consider only finite, undirected, and simple graphs. Let $G = (V, E)$ be a graph. We let n denote the number of vertices and m denote the number of edges.

For $W \subseteq V$ we denote by $G[W]$ the subgraph of G induced by W and we write $G - W$ instead of $G[V - W]$. The neighborhood of v is $N(v) = \{u \in V | uv \in E\}$ and $N[v] = N(v) \cup \{v\}$. If G is a directed graph, $N^-(v) = \{u \in V | (u, v) \in E\}$ and $N^+(v) = \{u \in V | (v, u) \in E\}$.

If $P = (v_1, v_2, \ldots, v_k)$ is a path, $u \in N(v_1)$, and $w \in N(v_k)$, then uP denotes the path $(u, v_1, v_2, \ldots, v_k)$, and Pw denotes the path $(v_1, v_2, \ldots, v_k, w)$. A P_4 is an induced subgraph on four vertices that is a path.

For an arc $e = (a, b)$, let its *transpose* e^T denote (b, a).

2.1. Representation of graphs. Analysis of the running time of an algorithm requires an agreement between the user and the designer of an algorithm about an appropriate format for the input data. No graph algorithm can claim an $O(n + m)$ running time if the input is provided in the form of an adjacency matrix, so a consensus has arisen that it is reasonable to require that the input be an adjacency list representation of the graph. If the user must spend $\Theta(n^2)$ time converting an adjacency matrix representation to this form before giving it as an input to the graph algorithm, this cost is not attributed to the running time of the graph algorithm.

Let us view a graph as an abstract data type that supports the following queries, possibly in addition to other queries:

- `Neighbors(`x`):` Given a vertex x, find its neighbors.
- `Adjacency(`x, y`):` Given two vertices, x and y, determine whether they are adjacent.

Algorithm texts usually contain a discussion of the trade-offs in space and time bounds for these two operations that arise in choosing an adjacency matrix representation or an adjacency list representation of a graph. The list of neighbors of a vertex in an adjacency list representation is typically unordered. This gives an extremely poor time bound for the `Adjacency` operation ($\Theta(n)$ in the worst case as opposed to $O(1)$ for the adjacency matrix).

This can easily be remedied by assuming that the adjacency lists are represented by a standard representation of an ordered set, such as a sorted array or a balanced binary search tree. This improves the time for the `Adjacency` query from $\Theta(n)$ in the worst case to $O(\log n)$, without increasing the space requirements or the time bound for the `Neighbors` operation. Moreover, it is possible to convert the unsorted variant to the sorted variant in $O(n + m)$ time by labeling the vertices with identifiers from 1 to n and then radix sorting the set of all edges according to the identifiers of their endpoints.

The only disadvantages to adopting this *ordered adjacency list representation* as a standard representation for sparse graphs seem to be cases where the data type must also support dynamic graph operations such as insertion of edges or contraction of vertices. Maintaining the sorted variant of the data structure can add an $O(\log n)$ factor to the cost of such operations. However, most graph algorithms deal with static graphs, and in the cases where the costs of these dynamic operations are an issue, an algorithm can ignore the ordering in the input data and neglect to maintain the invariant that it remain ordered as it executes.

Why has this ordered adjacency list representation not been adopted as a standard representation over the seemingly less desirable unordered one? This is probably because any algorithm that makes repeated use of the `Adjacency` query can produce the ordered representation from the unordered one in linear time before beginning work on the problem. Since a graph algorithm is assumed to run in $\Omega(n+m)$ time, the issue has been deemed inconsequential to the analysis of running times.

The possibility of graph algorithms that run in sublinear time, however, requires a reexamination of this issue. In contrast to algorithms that solve a problem on a graph from scratch, it is often the case that authentication algorithms can run in sublinear time.

An example of this occurs in connection with the class of *cographs*, which are the class of graphs that have no induced subgraph that is a P_4. That is, they have no induced subgraph that consists of exactly three edges forming a path (v_1, v_2, v_3, v_4) on four vertices. It takes $O(n+m)$ time to determine whether a given graph G is a cograph without the aid of a certificate [5]. However, given a legitimate P_4 that has been pointed out by a distrusted source, it takes $O(n)$ time to verify it on an unordered adjacency list representation, and $O(\log n)$ time on the ordered representation, since the `Adjacency` query takes $O(\log n)$ time. We therefore have a case where the choice of an unordered adjacency list representation versus an ordered one makes a difference in the analysis of the time bound of an algorithm.

We think that an ordered adjacency list representation would be a reasonable alternative to the unsorted one as a general-purpose representation of sparse graphs. In any case, there are compelling reasons to use the ordered representation when analyzing sublinear algorithms, and we will assume this representation throughout the paper. To be specific, since we deal only with problems involving static graphs, let us assume that each adjacency list is implemented as a sorted array of neighbors.

Before leaving this topic, we should mention that although the `Adjacency` operation takes $O(\log n)$ time, we can improve this to $O(1)$ by supplying what amounts to a certificate of the answer to the `Adjacency` query. If the answer to `Adjacency`(x, y) is `true`, the certificate consists of the location of the edge in the data structure, and if the answer is `false`, the certificate consists of the location where it would appear if it occurred. In either case, the user can check the result in $O(1)$ time. The certificate of a `false` answer cannot be made to work in this way on an unordered adjacency list representation.

Given this, it is easy to see that verifying the presence of a P_4 in a graph can be improved to $O(1)$ by building a certificate out of a collection of "subcertificates" for individual `Adjacency` queries on all pairs of elements of $\{v_1, v_2, v_3, v_4\}$. The same technique is useful for checking any induced subgraph that has been pointed out as a certificate of an answer to a graph problem.

3. What constitutes a certificate. Since software that generates a certificate could have a bug, a proposed certificate must be authenticated by verifying that it does, in fact, prove the result. For instance, if an odd cycle in a graph is presented as a certificate that the graph is not bipartite, authentication consists of verifying that it is a cycle, it has odd length, and that the claimed edges occur in the graph. The cycle can be given as a sequence of pointers to its edges in the input data structure, and takes $O(n)$ time for the user to authenticate, which is better than the $O(n+m)$ bound for checking whether a graph is bipartite.

A good certificate has an authentication algorithm that is conceptually simpler than algorithms for the original problem, or has a better time bound, or has both. If

the authentication step is simple and efficient enough, it may be possible to perform the check by visual inspection.

When the certificate is checked automatically, determining reliability of the implementation of the authentication algorithm is an obvious goal. Otherwise, consider the following scenario. The implementations of the certifying algorithm and of the authentication algorithm are both faulty. The certifying algorithm produces both an erroneous answer to the decision problem and an erroneous certificate, while the faulty authentication algorithm then claims that the certificate proves the given answer. The user is led to believe an erroneous conclusion.

A certificate is *sublinear* if its authentication algorithm has a tighter time bound than a linear one. For instance, for a problem where arbitrary graphs can be given as input, a certificate with an $O(n)$ authentication algorithm is sublinear, since this bound is never worse than linear and is often better. A certificate is *strong* (with respect to current algorithmic knowledge) if its authentication algorithm has a better time bound than that of the current fastest algorithm that solves the problem without a certificate. It is *weak* if it takes the same time to authenticate as it does to solve the original problem without the certificate, but may have other advantages such as greatly simplifying the task conceptually.

One of the anonymous referees has inquired about the possibility of certifying algorithms or authentication algorithms that require asymptotically more time or space than a noncertifying algorithm. Given the importance of software reliability in some industrial settings, the advantages of using these algorithms might outweigh the extra costs. We have not yet found interesting examples to support this, however.

Some of these notions are related to concepts that come up in the theory of NP-completeness. The class NP of decision problems are those for which, whenever the answer is *true*, this answer can be confirmed in polynomial time if one is supplied an appropriate certificate. The question of whether this is always possible without a certificate is the famous question of whether P = NP. The notion of a certificate and the time bound that it makes possible provides a precise mathematical definition of the class NP.

A certifying algorithm that returns a strong or sublinear certificate can be distinguished on objective mathematical grounds from a noncertifying algorithm. To be able to claim that certifying algorithms are a formal class of algorithms, we could require all certifying algorithms to produce sublinear certificates. Unfortunately, this restriction excludes many algorithms that produce certificates that are clearly useful in practice.

In addition, though sublinear certificates appear to be common, if the rejection certificate is sublinear, the acceptance certificate often fails to be sublinear, and if the acceptance certificate is sublinear, the rejection certificate often fails to be sublinear. For instance, the problem of recognizing bipartite graphs has a sublinear rejection certificate (an odd cycle) and a weak acceptance certificate (a 2-coloring). Recognizing connected graphs has a sublinear acceptance certificate (a spanning tree) and a weak rejection certificate (a cut $\{V', V - V'\}$ that has no edges across it). Directed acyclic graphs have a weak acceptance certificate (a topological sort) and a sublinear rejection certificate (a directed cycle). A similar phenomenon is seen in interval graphs and permutation graphs where the rejection certificate can be checked in $O(n)$ time, whereas the acceptance certificate is an $O(n)$ representation of G that defines the graph class and that requires $O(n + m)$ time to verify that it faithfully represents G.

What constitutes a weak certificate lacks the formal criteria that define sublinear or strong certificates. A weak certificate has value if it dramatically simplifies the task of solving the problem, without necessarily yielding a better time bound. There is no satisfactory formal measure of the conceptual difficulty of an algorithm, though differences are often obvious.

The value of proposed weak certificates must therefore be debated on a case-by-case basis in much the way that the ill-defined notion of "significance" of any new mathematical result is debated and evaluated through the peer-review process. The certificates in our examples were recognized previously as having theoretical relationships to the problems in question, independently of their use in algorithm design or error checking.

We would therefore describe certifying algorithms as embodying an algorithm-design philosophy, rather than a formal class of algorithms. We show through examples that the approach has been largely overlooked by both algorithm designers and software engineers, and we argue that it can have significant economic impact when algorithms are implemented. Moreover, in the case of graph algorithms, there is an extensive literature on forbidden subgraph characterizations of graph classes that promises to be a rich source of potential certificates for future work on certifying variants of existing graph algorithms.

3.1. Proofs of correctness of authentication algorithms. An interesting question raised by one of the anonymous referees of this paper is what relevance the simplicity of the proof of the authentication algorithm should have. Suppose the input is x, the correct output is $y = f(x)$, and the certificate, w, proves that $y = f(x)$. The proof that the authentication algorithm accepts only valid triples (x, y, w) should be trivial. The proof that the existence of such a triple proves that $y = f(x)$ does not need to be easy for the certificate to be useful.

Consider the question of deciding whether a graph is planar. A noncertifying algorithm for solving this problem inputs a graph G and outputs a bit $y = f(G)$ that is equal to 1 iff the graph is connected and planar. A certifying algorithm for planarity testing implemented in the LEDA package returns a Kuratowski subgraph when the input graph is nonplanar. A Kuratowski subgraph K is difficult to find, but easy to check once it is pointed out. Therefore, the proof that the authentication algorithm accepts only valid triples $(G, 0, K)$ is trivial. However, that K proves that $f(G) = 0$ is also easy to understand. It is not necessary for a user to understand why such a K appears in every nonplanar graph in order to understand that it proves that such an instance of a graph is nonplanar and that the program has answered truthfully.

What happens when G is planar gives a better illustration of the point raised by the referee. In this case, LEDA returns a *planar combinatorial embedding*. The validity of a planar combinatorial embedding is also easy to authenticate once it is pointed out, but it is more difficult to explain why it proves that the graph is planar.

For simplicity, assume for the moment that G is connected. Each undirected edge uv of G can be viewed as consisting of two directed edges (u, v) and (v, u); let $twin((u, v)) = (v, u)$. A planar embedding of G induces a cyclic order on the directed edges directed out of each vertex. Let $\pi_v((u, v))$ be the permutation of the edges that maps in clockwise order each edge out of a vertex to the next edge out of the vertex. Clearly, π_v has one cycle for each vertex. Given π_v, let $\pi_f((u, v)) = \pi_v(twin((u, v))$; it is easy to see that this is a permutation that maps each edge to the next edge about the face that lies immediately to the left of the edge. Therefore, π_f has a cycle for each face.

An abstraction of this is a *combinatorial embedding* of G, which consists of an *arbitrary* cyclic order π_v of the directed edges emanating from each vertex, but which may or may not correspond to the cyclic orders induced by any embedding of G in the plane. This nevertheless defines a second permutation $\pi_f((u,v)) = \pi_v(twin((u,v)))$ as before. By a well-known theorem of Euler, a combinatorial embedding is planar iff $2c - m - n - f = 0$, where f is the number of cycles π_f, c is the number of connected components of G, m is the number of edges, and n is the number of vertices.

To present this certificate, it is necessary only to give π_v as a cyclic order of edges out of each vertex and a pointer from each directed edge (u,v) to its twin (v,u). An authentication algorithm must verify that π_v induces a cyclic order on the edges directed out of each vertex, count the cycles in π_v and π_f, and verify that Euler's relation applies.

Though this test is simple, the proof that it implies that G is planar is not easy to explain to an unsophisticated programmer or user. (See [14] for a typical proof.) The usefulness of the certificate lies in the fact that Euler's formula applies to all instances of the problem and has received the thorough scrutiny of many experts. A person who trusts the theorem does not need to understand its proof in order to make use of the certificate. The certificate is useful because it allows trust in a well-known theorem to be substituted for trust in a program of dubious origin.

3.2. Preconditions and postconditions. One can imagine programs where different subproblems are solved by procedures that produce certificates, and where the program contains embedded code to authenticate each certificate before proceeding.

Suppose that an $O(\log n)$ binary search procedure is asked to search a sorted array of integers for an element i, and that the procedure returns with the answer that i does not occur in the array. It could be the case that the array was not correctly sorted, causing the binary search procedure to falsely declare that i does not occur in it, even though the binary search procedure is correctly implemented. Checking for this error would take $O(n)$ time, which would obviate the advantages of using a binary search procedure instead of a linear search.

Instead of this, we could define the binary search procedure as one that has a precondition that the input array be sorted *and a postcondition that its answer is correct whenever the precondition is met.* An error has occurred in the binary search procedure only if the precondition is met and the returned answer is incorrect.

Let us formally define an *error* in the binary search procedure to be a circumstance where the precondition is met but the returned value is incorrect. As in the case of all certifying algorithms, the question of whether a procedure has erred in some circumstance is separated from the question of whether it contains bugs. The binary search procedure can provide a certificate that it has not erred by returning either the index where i occurs, or else the index where i would occur if it is absent in the array. If the procedure claims that i occurs, the authentication algorithm checks the location to make sure that it is there. If the procedure claims that i does not occur, the authentication algorithm checks the location to make sure that the element at that position is smaller than i and the element at the next position is larger i. (Trivial special cases occur when the index is the last in the array.)

Though this example is trivial, it illustrates the possibility of designing certificates that show that *either* the returned value is correct *or* that the preconditions were not met, without distinguishing which of these cases occurred. This exonerates a procedure as the ultimate source of an error.

As a nontrivial example of such a certificate, the transitive orientation algorithm of [18] finds a transitive orientation of a comparability graph in $O(n+m)$ time. If a graph is given to the procedure that is not a comparability graph, then no transitive orientation exists, and the procedure produces an orientation in $O(n+m)$ time that is not transitive, without recognizing this. Various nonlinear bounds for checking whether an orientation is transitive are known, such as $O(nm)$, $O(m^{3/2})$, and the time to multiply two $n \times n$ matrices, $O(n^{2.376})$. These are the best bounds for recognizing comparability graphs.

However, it is reasonable to view as a precondition the requirement that the input to the transitive orientation algorithm be a comparability graph, since the orientation is meaningless if this is not the case. Below, we show how to provide a certificate, called an *orientation tree*, that shows that either the precondition has not been met or the returned orientation is transitive. This certificate is quite simple to check in $O(n+m)$ time.

4. Chordal bipartite graphs. A *chord* in a simple cycle is an edge that is not an edge of the cycle, but whose endpoints are both vertices in the cycle. In a bipartite graph, every cycle is even, so every cycle has length at least four. A graph is *chordal bipartite* if it has no chordless cycle of length greater than four.

Chordal bipartite graphs provide an easy introduction to techniques that we will use to produce certifying algorithms. Lubiw [16] gives an $O(n+m\log^2 n)$ noncertifying algorithm for recognizing them. We show how to modify it to obtain a certifying algorithm.

If M is a matrix, let $M_{i,j}$ denote the element in row i, column j. The rows of a matrix M are in *lexical order* if they are sorted in dictionary order. That is, whenever $\{i, i+1\}$ are a consecutive pair of rows and j is the first column where the rows differ, then $M_{i+1,j} > M_{i,j}$. The columns are in lexical order if, as rows, they are in lexical order in the transpose of the matrix.

A bipartite graph G with bipartition U, W can be represented with a Boolean *bipartite adjacency matrix* A, which has one row i for each element of $u_i \in U$, one column for each element $w_i \in W$, and for each pair $\{u_i, w_j\}$, $A_{i,j} = 1$ if $u_i w_j$ is an edge of G, and $A_{i,j} = 0$ otherwise.

Given a graph G, Lubiw's algorithm tests whether G is bipartite, and, if so, finds a bipartition and a *doubly lexical ordering* of the resulting bipartite adjacency matrix. A doubly lexical ordering of a matrix is a permutation of the rows and columns such that the resulting matrix passes the following two tests:

1. When the order of the columns is reversed, the rows are in lexical order.
2. When the order of the rows is reversed, the columns are in lexical order.

Every matrix has a doubly lexical ordering, so this is not a test of whether G is chordal bipartite. Lubiw's algorithm then searches this doubly lexical matrix A for a configuration called a *Gamma* (Γ). A Γ is a pair (h, i) of rows and a pair (j, k) of columns such that $h < i$, $j < k$, $A_{h,j} = A_{h,k} = A_{i,j} = 1$, and $A_{i,k} = 0$.

The critical theorem is that a bipartite graph is chordal bipartite iff a doubly lexical ordering of its bipartite adjacency matrix has no Γ. Lubiw's algorithm for finding the doubly lexical ordering and testing for Γ's uses a sparse representation of A and takes $O(n+m\log^2 n)$, which gives this bound for recognition of chordal bipartite graphs. The bound for finding the doubly lexical ordering has been improved to $O(n+m\log n)$ by Paige and Tarjan [22], and to $O(n^2)$ by Spinrad [26], and these give bounds of $O(n+m\log n)$ and $O(n^2)$, respectively, for recognizing chordal bipartite graphs.

Though its proof of correctness is not obvious, Lubiw's algorithm for searching for a Γ with respect to given orderings of the bipartition classes is trivial to program and runs in $O(n+m)$ time. Therefore, a doubly lexical ordering is a strong certificate that a graph is chordal bipartite. The certificate is represented by a bipartition of the vertices and two orderings of the bipartition classes. The authentication algorithm verifies that each of the two bipartition classes is an independent set and that the graph has no Γ with respect to the given orderings of the bipartition classes. All of this takes $O(n+m)$ time. The doubly lexical ordering is therefore a strong certificate, but not a sublinear one.

When a graph is not chordal bipartite, the same certificate can be used to show this, so it is a strong certificate for this case also. However, with some additional effort, we can obtain a sublinear certificate. Note that this precludes verifying the correctness of a doubly lexical ordering, which requires $\Theta(n+m)$ time.

Lubiw [16] proves that when a Γ occurs in a doubly lexical ordering, it is part of a chordless cycle that has at least six vertices. To find such a cycle, let $\{(a,b),(b,c),(c,d)\}$ be the Γ, and remove $N(b)\cup\{b\}\cup N(c)\cup\{c\}-\{a,d\}$ from the graph. Since the Γ is part of a chordless cycle, there exists a path from a to d in this induced subgraph. Using breadth-first search (BFS) starting at a, we may find a shortest path P from a to d in this induced subgraph. Since $\{(a,b),(b,c),(c,d)\}$ is a P_4 and all members of P other than a and d are nonneighbors of b and c, the union of P and the $\{(a,b),(b,c),(c,d)\}$ is a chordless cycle of length at least six. This chordless cycle can be returned as a certificate that the input graph is not chordal bipartite.

On first inspection, this does not seem like a sublinear certificate. To verify that the returned cycle C is indeed chordless, a skeptical user must spend $O(n+m)$ time in the worst case to verify that the cycle has no chords. The key to an $O(n)$ authentication algorithm is the observation that its purpose is to verify that the graph has a chordless cycle and to not verify that C is an example of one.

An $O(n)$ authentication algorithm first verifies that C is a cycle in G of size greater than four. It then selects any four consecutive vertices (u,x,y,w) on the cycle and verifies that x and y have no neighbors on C other than the ones that are supposed to have. That is, it verifies that x has no neighbors on C other than u and y, and y has no neighbors on C other than x and w, and that u and w are nonadjacent. If these tests fail, the certificate is ruled faulty. Otherwise, even if C still has undetected chords, the existence of a path from u to w that avoids the neighborhoods of x and y proves the existence of a shortest such path, P. The union of (u,x,y,w) and such a shortest path is a chordless cycle of length greater than four. Therefore, the user may be certain that the graph has a chordless cycle based on this authentication algorithm, even though the algorithm does not fully verify that C is itself chordless.

5. Interval graphs. An undirected graph is *chordal* if it has no chordless cycle of length at least four. Three independent vertices x,y,z of a graph G are an *asteroidal triple* (AT) of G if, between each pair of these vertices, there is a path that contains no neighbors of the third. A graph is said to be *AT-free* if it has no AT. For more information on these and other graph classes we refer the reader to [3, 8]. We will rely on the following well-known theorem.

THEOREM 5.1 (see [15]). *A graph is an interval graph iff it is chordal and AT-free.*

A graph is chordal iff it has a *perfect elimination ordering*, which is an ordering $(v_1,v_2,\ldots,v_n)$ of the vertices such that for each v_i, $N(v_i)\cap\{v_{i+1},v_{i+2},\ldots,v_n\}$ is a clique [6]. A perfect elimination ordering of a chordal graph can be found in

linear time by the lexicographical BFS (LexBFS) algorithm [25, 8]. A modification of this algorithm points out a chordless cycle of length at least four as a certificate of nonmembership [27]. Hence, this is a linear-time certifying recognition algorithm for chordal graphs.

5.1. The certificates. When the graph is an interval graph, we produce an interval model, just as the prior algorithms do. For the authentication step, it is easy to check whether this model corresponds to the input graph in time that is linear in the size of the input graph. The basic trick is to work left-to-right through the model, generating edges implied by the model and rejecting the certificate immediately if the number of edges exceeds the number of edges in the graph. Otherwise, when finished, verify that the generated edges are the same as those in the graph by labeling the vertices with identifiers from 1 to n, and then radix sorting the set of all edges of G according to the identifiers of their endpoints. Since authentication takes linear time, the interval model is a weak certificate.

When the graph is not an interval graph, Theorem 5.1 provides the basis of our certificate: we produce either a chordless cycle or an AT. Despite initial appearances, these can be turned into sublinear certificates. For the AT, we accomplish this by returning not only the triple, but for each pair in the triple, the sequence of edges of a simple path between them that avoids the neighborhood of the third. The sequence of edges may be given by pointers to the corresponding edge structures in the user-supplied data. Given the triple, it is easy to find these three paths in linear time. The authentication algorithm must verify that each proposed path is, in fact, a path, that its edges occur in G, and that no neighbors of the third vertex occur on it. If each path is given by pointers to edges in the input structure in the order in which they occur on the path, this is accomplished in $O(n)$ time by marking the neighbors of each vertex in the triple. This is a sublinear certificate because $O(n)$ is a better bound than $O(n + m)$.

As in the case of chordal bipartite graphs, a chordless cycle can be used to verify in $O(n)$ time that such a cycle exists, without fully verifying that the given cycle is an example of one. The authentication algorithm is the same as in the chordal bipartite case, except that when (u, x, y, w) are consecutive vertices, the certificate is not rejected if u and w are adjacent.

5.2. Generating the certificates. For our certifying algorithm, we use the linear-time algorithm of Korte and Möhring [13] as a subroutine. Though this is not a certifying algorithm, it produces a certificate in the form of an interval representation in the case where the graph is an interval graph.

Suppose the input graph is not an interval graph. Using the algorithm of [27], we return a chordless cycle if the graph is not chordal.

It remains to show how to produce a certificate in the case where the graph is chordal, but not an interval graph. Henceforth, we will assume that this case applies.

The algorithm of Korte and Möhring produces a perfect elimination ordering $(v_1, v_2, \ldots, v_n)$, and incrementally decides whether the subgraph induced by the vertices $\{v_n, \ldots, v_i\}$ is an interval graph. Since we now assume that the graph is not an interval graph, it fails when considering a particular vertex v_{i-1}. The subgraph induced by the vertices $\{v_n, \ldots, v_i\}$ is an interval graph and the subgraph induced by $\{v_n, \ldots, v_i, v_{i-1}\}$ is not.

In the remainder of this section, we use G to denote the subgraph induced by $\{v_n, \ldots, v_i\}$ and we let $x = v_{i-1}$. The graph $G + x$ is a chordal graph, but not an interval graph, and hence it must contain an AT by Theorem 5.1. The neighbors of x

form a clique in G since $(v_1, v_2, \ldots, v_n)$ is a perfect elimination ordering of the input graph.

The Korte–Möhring algorithm provides an interval model of G. We may assume without loss of generality that all endpoints in the interval model of G are pairwise distinct, since, when they are not, they can be perturbed to make this true without altering the represented graph. Let us then number the endpoints in left-to-right order, and for each vertex, let $l(v)$ and $r(v)$ denote the numbers of the left and right endpoints of the interval corresponding to v. This gives a "normalized" interval model where $I(v) = [l(v), r(v)]$ is the interval that corresponds to v, all endpoints are distinct, and $l()$ and $r()$ are integer-valued functions from $V(G)$ to $\{1, 2, \ldots, 2n\}$.

LEMMA 5.2. *Let G be an interval graph such that $G' = G + x$ is not an interval graph and $N(x)$ is a clique. Then x is a member of every AT of G'.*

Proof. Suppose $\{a, b, c\}$ is an AT of G' and $x \notin \{a, b, c\}$. There is a path P from a to b in G' that avoids the neighborhood of c. If P contains x, then, since the neighborhood of x is a clique, x's predecessor and successor on P must be adjacent, and x can be spliced out of P to yield a path in G. Thus, there is a path in G from a to b that avoids the neighborhood of c. By symmetry among the members of $\{a, b, c\}$, $\{a, b, c\}$ is an AT of G, contradicting the assumption that G is an interval graph. □

DEFINITION 5.3. *Let us say that interval $[x_1, x_2]$* precedes *interval $[y_1, y_2]$ iff $x_1 < y_1$ and $x_2 < y_2$. Let P be a path in G. Let the* rightward extent *$R(P)$ of P denote* $\texttt{max}\ \{r(u) | u$ *is a vertex on $P\}$, and let the* leftward extent *$L(P)$ of P denote* $\texttt{min}\ \{l(w) | w$ *is a vertex in $P\}$. Let $D(x) = \bigcap\{I(v) | v \in N(x)\}$ be the intersection of the intervals representing the neighbors of x. Then $D(x) \neq \emptyset$ since the neighbors of x form a clique.*

Since an AT is an independent set, in any AT $\{x, y, z\}$ of G', one of $I(y)$ and $I(z)$ precedes the other, and if $I(y)$ precedes $I(z)$, then $r(y) < l(z)$.

LEMMA 5.4. *Let $\{x, y, z\}$ be an AT in G', where $I(y)$ precedes $I(z)$. Then*

$$r(y) < l(D(x)) < r(D(x)) < l(z).$$

Proof. Assume otherwise, say, $l(D(x)) < r(y)$, and consider any path P from z to $v \in N(x)$ avoiding $N(y)$. Then $v \notin N(y)$. Together with $l(v) \leq l(D(x))$, this implies $r(v) < l(y)$. Since $r(y) < l(z)$, P must contain a neighbor of y, a contradiction. □

DEFINITION 5.5. *If $r(y) < l(D(x))$, then let $R(y) =$* $\texttt{min}\ \{R(P) | P$ *is a path from y to a neighbor of $x\}$. That is, $R(y)$ is the minimum rightward extent of any path from y to a neighbor of x. Similarly, if $r(D(x)) < l(z)$, then let $L(z) =$* $\texttt{max}\ \{L(P) | P$ *is a path from z to a neighbor of $x\}$. That is, $L(z)$ is the maximum leftward extent of any path from z to a neighbor of x.*

LEMMA 5.6. *If $r(y) < l(D(x)) < r(D(x)) < l(z)$, then $\{x, y, z\}$ is an AT in G' iff y and z are in the same component of $G - N(x)$ and $[r(y), R(y)]$ precedes $[L(z), l(z)]$.*

Proof. If $\{x, y, z\}$ is an AT in G', then y and z are in the same component of $G - N(x)$, since there is a path of G' that avoids the neighborhood of x. There is a path from y to x in G' that avoids the neighborhood of z, and hence a path to a neighbor of x in G that contains no neighbor of z. Since $r(y) < l(z)$, the intervals of all vertices on this path have their right endpoint to the left of $l(z)$. Therefore, $R(y) < l(z)$. By mirror symmetry, $r(y) < L(z)$.

If $\{x, y, z\}$ is not an AT in G', then since $\{x, y, z\}$ is an independent set, every path of G' between some pair of the vertices contains a neighbor of the third. If every path between y and z contains a neighbor of x, then y and z are in different components of $G - N(x)$. If all paths from y to a neighbor of x contain a neighbor of

z, then the rightward extent of all such paths is greater than $l(z)$, and $R(y) > l(z)$. By mirror symmetry, $r(y) < L(z)$. □

Lemma 5.6 gives the strategy of our approach for finding an AT in G'. Removing the intervals corresponding to $N(x)$ from the interval model of G gives an interval model of $G - N(x)$. We look for a component whose intervals span $[l(D(x)), r(D(x))]$. For each y in the component such that $r(y) < l(D(x))$ we compute $R(y)$, and for each z in the component such that $r(D(x)) < l(z)$ we compute $L(z)$. We then look for a pair $[r(y), R(y)]$, $[L(z), l(z)]$, such that $[r(y), R(y)]$ precedes $[L(z), l(z)]$, using Lemma 5.7, which we give first.

LEMMA 5.7. *Given two sets $\mathcal{X}$ and $\mathcal{Y}$ of intervals, where the right endpoints of $\mathcal{X}$ are given in ascending order and the left endpoints of $\mathcal{Y}$ are given in descending order, it takes $O(|\mathcal{X}| + |\mathcal{Y}|)$ time to either determine that no interval in $\mathcal{X}$ precedes any interval in $\mathcal{Y}$, or else return $a \in \mathcal{X}$ and $b \in \mathcal{Y}$ such that a precedes b.*

Proof. As a base case, if $\mathcal{X}$ or $\mathcal{Y}$ is empty, there is no such pair. Otherwise, select $u \in \mathcal{X}$ that minimizes $r(u)$, and select $w \in \mathcal{Y}$ that maximizes $l(w)$. If $l(u) < l(w)$ and $r(u) < r(w)$, then return (u, w) as (a, b). Otherwise, if $l(u) \geq l(w)$, then u is not a candidate for a since its left endpoint does not lie to the left of any left endpoint in $\mathcal{Y}$. Let $\mathcal{X} := \mathcal{X} - \{u\}$. By mirror symmetry, if $r(u) \geq r(w)$, then w is not a candidate for b, so let $\mathcal{Y} := \mathcal{Y} - \{w\}$. By induction on the size of $|\mathcal{X}| + |\mathcal{Y}|$, a recursive call on the new $\mathcal{X}$ and $\mathcal{Y}$ solves the original problem. Because of the way the data are sorted, it takes $O(1)$ time to select u and w and to prepare the recursive call, in which $|\mathcal{X}| + |\mathcal{Y}|$ has been reduced by at least 1. □

To use Lemma 5.7, we let $\mathcal{X} = \{[r(y), R(y)] | r(y) < l(D(x))\}$ and let $\mathcal{Y} = \{[L(z), l(z)] | r(D(x)) < l(z)\}$. Since $r()$ and $l()$ are integer functions from 1 to $2n$, sorting the endpoints as required by the lemma takes $O(n)$ time.

Therefore, by Lemmas 5.6 and 5.7, the problem of finding an AT reduces in linear time to computing $R(y)$ at each y such that $r(y) < l(D(x))$ and $L(z)$ at each z such that $r(D(x)) < l(z)$. We give the procedure for $R()$, and by mirror symmetry, this gives the procedure for $L()$.

DEFINITION 5.8. *A path P in G is* increasing *if, whenever u is earlier than v on P, $r(u) < r(v)$. Let D_r be the orientation of $G - N(x)$, where (u, v) is an arc in D_r iff uv is an edge in $G - N(x)$ and $r(u) < r(v)$.*

Our strategy for computing $R()$ is to find a way to restrict our attention to increasing paths from y to $N(x)$, which allows us to work in D_r rather than in G. Since D_r is a dag, this simplifies the problem.

LEMMA 5.9. *If there is a path P from u to v with rightmost extent $R(P) = r(v)$, then there is an increasing path P' from u to v such that $R(P') = R(P)$.*

Proof. If $u = v$, P is vacuously increasing. Suppose P has length greater than 0 and the lemma is true for shorter paths. Let w be the first vertex on P with $r(w) > r(u)$. $R(P) = r(v)$ implies $r(u) < r(v)$, so w exists. Then $I(u)$ and $I(w)$ intersect, and hence u and w are neighbors. By induction, there is an increasing path P'' from w to v and $P' = uP''$ satisfies the lemma. □

LEMMA 5.10. *Let $r(y) < l(D(x))$ and let P be a path in G from y to a neighbor v of x. Then there is a path $P'v$ from y to v such that $R(P'v) \leq R(P)$ and P' is increasing.*

Proof. If $R(P) = r(v)$, then the claim follows from Lemma 5.9. Otherwise, let w be the first vertex on P such that $I(w)$ intersects $D(x)$, and let P'' be the portion of P from y to w. Since $R(P'') = r(w)$, there is an increasing path P' from y to w, and since $I(w)$ intersects $D(x)$, w and v are adjacent. □

LEMMA 5.11. *It takes linear time to compute $R(y)$ for every $y \in V(G)$ such that $r(y) < l(D(x))$.*

Proof. By Lemma 5.10, we need only consider *well-behaved* paths in G that consist of a directed path in D_r, followed by a single edge of G to a neighbor of x. For any such well-behaved path P, $R(P)$ is the maximum of the last two values of $r()$ on the path. For each $y \in V(G) - N(x)$, let $R_i(y) = \texttt{min}\{R(P)|P$ is a well-behaved path of length at most $i\}$, or ∞ if there is no such path. The value of $R_1(y)$ is trivial to compute at all nodes in $V - N(x)$ in linear time. Let $(u_p, u_{p-1}, \ldots, u_1)$ be a topological sort of D_r. Then $R_i(u_i) = \texttt{min}\ \{R_1(u_i)\} \cup \{R_j(u_j)|(u_i, u_j)$ is an arc of $D_r\}$. This may be computed by induction on i in linear time. For any $y = u_k$ such that $r(y) < l(D(x))$, $R(y) = R_k(u_k)$, since y has at most $k - 1$ successors in D_r. □

5.3. Comparison with the original Korte–Möhring algorithm. One of the anonymous referees has asked for a clarification of why the Korte–Möhring algorithm is not already a certifying algorithm, producing a sublinear certificate of rejection, just as ours does.

Answering this question requires more details about how the algorithm works. The Korte–Möhring algorithm paper [13] defines a *modified PQ tree* (*MPQ tree*) for an interval graph G, which gives a way of representing implicitly all possible interval representations of G. Vertices of G are associated with nodes of the MPQ tree; details about this representation are given in the paper.

As described above, the Korte–Möhring algorithm works by induction on a perfect elimination order $(v_1, v_2, \ldots, v_n)$. Let G_i denote the subgraph of G induced by $\{v_n, v_{n-1}, \ldots, v_i\}$, and let T_i be its MPQ tree. At each step $i - 1$ it produces the MPQ tree T_{i-1} of G_{i-1} from the MPQ tree T_i of G_i, unless G_{i-1} fails to be an interval graph, in which case it rejects G_{i-1}, and hence G is not an interval graph.

At each step, recognizing whether G_{i-1} is an interval graph is trivial, since this is the case iff the neighbors of v_{i-1} in G_i have a certain relationship to T_i that is quite easy to check in $O(n)$ time. If G is not an interval graph, the test will fail at some step $i - 1$.

Since it is possible to verify that G is not an interval graph in $O(n)$ time using T_i and v_{i-1}, the referee has suggested that (T_i, V_{i-1}) be returned as a sublinear certificate of rejection.

Though compelling, this idea has the subtle flaw that it violates the principle that it must not be possible for an authentication algorithm to be induced to make a false declaration by an erroneous or counterfeit certificate. It must either declare a correct answer to the original problem, or it must correctly declare that the claimed certificate contains an error and fails to show what it claims to show. Showing that (T_i, v_{i-1}) fails the simple $O(n)$ test demonstrates only that *either* G_{i-1} is not an interval graph, *or* that T_i is not the correct MPQ tree for G_i, *or* that T_i is the correct MPQ tree for G_i but that $(v_1, v_2, \ldots, v_n)$ is not a perfect elimination order. In fact, the authors make clear that an additional requirement for the algorithm to work is for $(v_1, v_2, \ldots, v_n)$ to be a special case of a perfect elimination order produced by the LexBFS algorithm.

Therefore, in addition to verifying that (T_i, v_{i-1}) fails to have the required simple relationship to T_i, it is also necessary to verify that T_i is the correct MPQ tree for G_i and that $(v_1, v_2, \ldots, v_n)$ is a LexBFS order.

A necessary step in checking that T_i is the correct MPQ tree for G_i is to check that an interval model it implies correctly reflects every edge of G_i. Otherwise, it leaves open the possibility that T_i is the MPQ tree for some interval graph $G_i' \neq G_i$.

The authentication algorithm will falsely declare that G_{i-1} is not an interval graph if $G'_i + v_{i-1}$ is not an interval graph, but $G_i + v_{i-1}$ is. There is little hope of finding a sublinear algorithm for checking whether a given interval model faithfully represents a given graph G, though this is easily accomplished in linear time. The claim that it is a sublinear certificate cannot be made, given what is currently known, and there are reasons to believe that this will not change in the future.

Whether (T_i, v_{i-1}) is useful as a weak certificate depends on the question of whether it clearly simplifies conceptually the task of verifying that G is not an interval graph. In constructing T_i, checking whether T_j and v_{j-1} satisfy the required test for $j > i$ is quite simple, but updating T_j to produce T_{j-1} when it passes this test is considerably more complicated. Though it takes linear time, the problem of checking that an interval model $\mathcal{I}_i$ implied by the MPQ tree T_i faithfully represents G_i is a fairly simple task. However, once this has been accomplished, one must still verify that T_i is the correct MPQ tree for $\mathcal{I}_i$. This problem is one of the main subjects of a forthcoming paper [17]; it is shown there that it can be solved in $O(n)$ time. The algorithm is quite involved, though it is possible that a simpler algorithm could accomplish the task in $O(n + m)$ time. As for verifying the LexBFS order, there is an algorithm for checking whether an order is a perfect elimination order [8], but we do not know of an algorithm for verifying a LexBFS order other than rerunning the LexBFS algorithm that produced it. An interesting question is whether the interval representation could help.

In summary, the usefulness of (T_i, v_{i-1}) in simplifying the problem significantly without leaving open the possibility of giving erroneous output has not been demonstrated. If this is accomplished, it appears unlikely that the resulting authentication will compete in simplicity with the one for checking a given AT or given chordless cycle, or that it will be sublinear.

6. Comparability graphs. Let us consider an undirected graph G to be a special case of a digraph, namely, the symmetric digraph where if (x, y) is a directed arc, then so is (y, x). The undirected edge xy is just the pair $\{(x, y), (y, x)\}$. Finding a transitive orientation of an undirected graph G amounts to deleting one arc from each symmetric pair so that the remaining arcs form a transitive digraph.

Suppose (a, b) and (b, c) are arcs of G and (a, c) is not. Any orientation of G where both (a, b) and (b, c) appear must fail to be transitive, due to the forced absence of (a, c) in the orientation. Then (b, c) is *incompatible* with (a, b), as is (b, a), since a transitive orientation must be antisymmetric. We may represent the incompatibility relation with an *incompatibility graph* whose vertices are the arcs of G and whose (undirected) edges are the incompatible pairs of arcs of G. (See Figure 4.)

We show in section 6.3 that the following is an immediate corollary of a result from [7, 8].

THEOREM 6.1. *An undirected graph G is a comparability graph iff its incompatibility graph is bipartite.*

The following is a consequence of Theorem 6.18, which we give in section 6.3, and gives a sublinear certificate that a graph is not a comparability graph, since it can be checked in $O(n)$ time.

THEOREM 6.2. *When G fails to be a comparability graph, its incompatibility graph has an odd cycle of length $O(n)$.*

A linear-time algorithm for finding a transitive orientation of a comparability graph is given in [18]. This transitive orientation algorithm represents the orientation it assigns to the edges implicitly by giving a linear extension (topological sort) of the

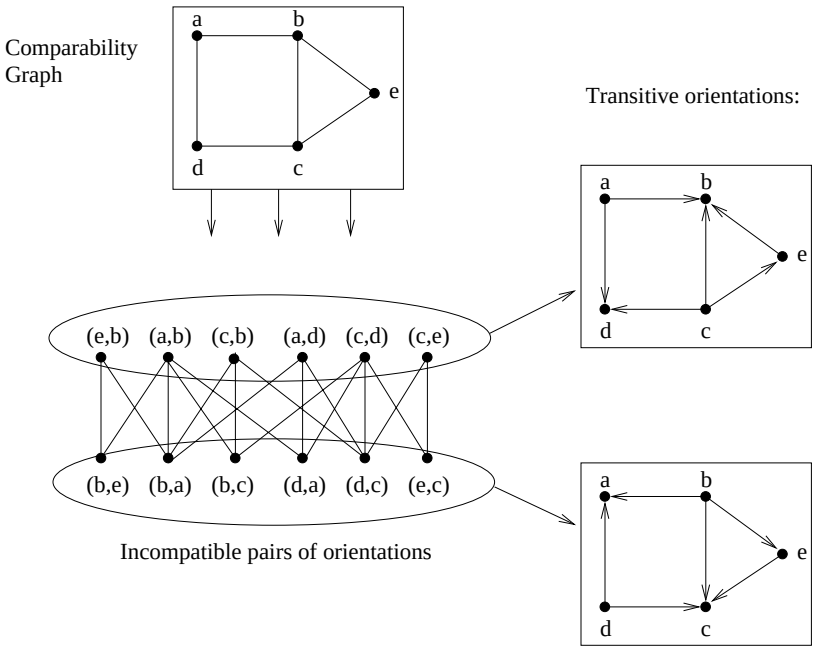


FIG. 4. *The incompatibility graph of a graph. There is one vertex for each directed edge, and two are adjacent if one is the transpose of the other, or if they are of the form* $\{(a,b),(b,c)\}$ *and* (a,c) *is not an edge. Two adjacent edges cannot appear in a transitive orientation. Therefore, a transitive orientation must be an independent set in the incompatibility graph. Reversing the direction of the edges in a transitive orientation yields a new transitive orientation. From this observation, it is easy to see that the incompatibility graph of a comparability graph is bipartite.*

orientation that it produces. This allows it to be applied to $\overline{G}$ in time linear in the size of G.

When the transitive orientation algorithm is asked to provide a transitive orientation of a graph G that is not a comparability graph, it produces an acyclic orientation of the graph, which it represents with a linear extension. This orientation must contain an incompatible pair, namely, a pair $\{(a,b),(b,c)\}$ of directed edges in series such that ac is not an edge of G, and hence (a,c) is not a directed edge in the orientation. No general linear-time algorithm for finding an incompatible pair in a dag is known. Because of this, no linear-time algorithm is known for recognizing comparability graphs, even though a linear-time algorithm for transitively orienting them is available.

We prove the following lemma in section 6.3.

LEMMA 6.3. *Given an incompatible pair in the orientation of a graph* G *produced by the transitive orientation algorithm of* [18]*, it takes* $O(n+m)$ *time to find an odd cycle of length* $O(n)$ *in* G*'s incompatibility graph, and given an incompatible pair in an orientation of* $\overline{G}$ *by the algorithm, it takes* $O(n+m)$ *time to find an odd cycle of length at* $O(n)$ *in* $\overline{G}$*'s incompatibility graph.*

6.1. Minimum proper coloring and maximum clique. The algorithm of [18] for finding a minimum proper coloring and maximum clique in a comparability graph proceeds as follows. Given an arbitrary input graph G, it finds an acyclic orientation that will be transitive if the input is a comparability graph. It then labels each vertex

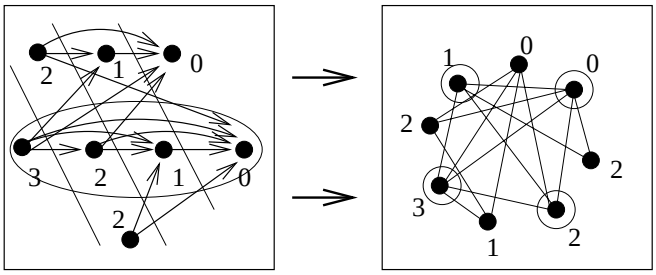


FIG. 5. *Finding a maximum clique and minimum proper coloring in a comparability graph. The color of each vertex is the length of the longest directed path originating in a transitive orientation. A longest path is a clique. Since the size of a clique is a lower bound on the number of colors in a proper coloring, the coloring is a certificate that the clique is of maximum size and the clique is a certificate that the proper coloring is a minimum one.*

with the length of the longest directed path originating at the vertex in the resulting directed acyclic graph. This labeling is easily accomplished in $O(n + m)$ time by labeling each vertex as it finishes during a depth-first search of the graph. (See Figure 5.) Each neighbor of the vertex is already labeled at that point, so the vertex can be labeled with one plus the maximum of the labels of its neighbors. This is a proper coloring, since for any edge xy of G, the label of one of x and y will exceed the other's by at least 1.

If the longest path is a clique, then the coloring is a minimum one and the path is a maximum clique. This is because the size of a clique is a lower bound on the number of colors needed, and since the clique has the same number of vertices as the number of colors used, the coloring serves as a certificate that the clique is a maximum one and the clique serves as a certificate that the coloring is a minimum one. These are weak certificates, since it takes $O(n+m)$ time for a user to verify that the coloring is a proper one and that the path is a clique.

If G is not a comparability graph, the algorithm still assigns an acyclic orientation, which allows the vertices to be colored as before, and finds a maximum-length directed path P. If P is a clique in G, then it is a maximum clique and the coloring is a minimum proper coloring, so these may be returned as each other's certificate just as in the case where G is a comparability graph. If P is not a clique in G, then there exist two consecutive arcs (a, b) and (b, c) on P such that (a, c) is not an edge, and these are easy to find in linear time. Since (a, b) and (b, c) are an incompatible pair, an odd cycle in the incompatibility graph may then be found in $O(n+m)$ time, by Lemma 6.3. As pointed out above, this is a sublinear certificate that G is not a comparability graph.

6.2. Permutation graphs. In [18], it is shown that the transitive orientation algorithm given there gives rise to a linear-time bound for recognizing permutation graphs. The algorithm is based on the following characterization of permutation graphs.

THEOREM 6.4 (see [23, 8]). *An undirected graph G is a permutation graph iff G and its complement $\overline{G}$ are both comparability graphs.*

When G is a permutation graph, the algorithm finds a topological sort of a transitive orientation D of G and a topological sort of a transitive orientation D' of $\overline{G}$. $D \cup D'$ is a *tournament* (an orientation of a complete graph) and acyclic. It then finds the unique topological sort of $D \cup D'$ to yield a linear ordering of V, and the unique

topological sort of $D^T \cup D'$ to give a second linear arrangement of V. By results from [23, 8], these two linear arrangements are a permutation model of G.

When G is not a permutation graph, the procedure in [18] produces a faulty permutation model of G. Success or failure of an authentication algorithm on the permutation model it produces provides the basis for deciding whether the graph is a permutation graph in that algorithm. The procedure is not a certifying algorithm, since the permutation model could also have been faulty due to a bug in the implementation.

A linear-time authentication algorithm for a proposed permutation model is given in [18]. The permutation model is therefore a weak certificate.

The algorithm for recognizing permutation graphs given in [18] uses the transitive orientation algorithm to find linear extensions of orientations D and D' of G and of $\overline{G}$. Since it provides a certificate if G is a permutation graph, we will assume in the remainder of the paper that G is not. In this case, at least one of D and D' has an incompatible pair.

We now describe how to find an incompatible pair in D or D' in time linear in the size of G, given G and linear extensions π and τ of D and of D'. This constitutes proof that the implementation of the transitive orientation algorithm failed to produce an orientation of G or of $\overline{G}$ that is transitive. However, it is not a certificate that G is not a permutation graph, since the failure could be due to a bug in the implementation of the algorithm.

LEMMA 6.5. *Let G be a graph, and let D and D' be acyclic orientations of G and $\overline{G}$. Then $D \cup D'$ and $D^T \cup D'$ are both acyclic iff D and D' are each transitive.*

Proof. Since $D \cup D'$ is a tournament, then if it has a cycle, it has a three-cycle. Suppose there is a directed three-cycle $(x, y), (y, z), (z, x)$ in $D \cup D'$. Since D and D' are both acyclic, one of these arcs belongs to D and another belongs to D'. Suppose without loss of generality that $(x, y), (y, z)$ belong to D. Then since $(x, z) \notin D$, D is not transitive. An identical argument applies if $D^T \cup D'$ contains a directed three-cycle.

Next, suppose that one (or both) of D and D' fails to be transitive. Assume without loss of generality that D fails to be transitive. Then there exists an incompatible pair $\{(a, b), (b, c)\}$. Therefore (a, c) is not an arc of D, and since D is acyclic, (c, a) is not an arc of D. Therefore, (a, c) or (c, a) is an arc of D'; if (a, c) is an arc of D', then $\{a, b, c\}$ induces a three-cycle in $D^T \cup D'$, and if (c, a) is an arc of D', then $\{a, b, c\}$ induces a three-cycle in $D \cup D'$. □

LEMMA 6.6. *Let G, D, and D' be as in Lemma 6.5. Given a three-cycle in $D \cup D'$ or $D^T \cup D'$, it takes $O(1)$ time to return an incompatible pair in D or in D'.*

Proof. Suppose the three-cycle occurs in $D \cup D'$. Since each of D and D' is acyclic, two of the arcs of the cycle occur in one of D and D' and give an incompatible pair in it. □

Let $p(x)$ be the number of predecessors of $x \in V$ in $D \cup D'$. This is just $|N^-(x)|$ in D plus $|N^-(x)|$ in D'.

LEMMA 6.7. *If for each $i \in \{0, 1, \ldots, n-1\}$ there exists $x \in V$ such that $p(x) = i$, then $D \cup D'$ is acyclic.*

Proof. (By induction on i.) Suppose $D \cup D'$ has n vertices and satisfies the conditions of the lemma. The claim is immediate if $n = 1$. Suppose $n > 1$ and the claim holds for $n-1$. There is a vertex s in $D \cup D'$ such that $p(s) = n-1$. Since every other vertex is a predecessor of s, no directed cycle of $D \cup D'$ contains s. However, removal of s leaves an induced subgraph of $D \cup D'$ on $n - 1$ vertices that satisfies

the condition of the lemma, so by the induction hypothesis, there can be no directed cycle that excludes s. □

LEMMA 6.8. *Let G, D, and D' be as in Lemma 6.5, and let π and τ be topological sorts of D and D', respectively. Given G, π, and τ, it takes $O(n+m)$ time to find a three-cycle in $D \cup D'$ or else determine that $D \cup D'$ is acyclic.*

Proof. In $O(n)$ time, we may label the elements of V with their position numbers in π and in τ. In $O(n+m)$ time, we can then label every $x \in V$ with the value of $|N^-(x)|$ in D by counting, for each vertex, the neighbors in G with earlier position numbers. To find $N^-(x)$ in D' we cannot do this directly in linear time, since D' is an orientation of $\overline{G}$, which might not have $O(n+m)$ size. Instead, let $i(x)$ be the number of predecessors of x in τ; $i(x)$ is just the position number of x in τ, minus one. Let $q(x)$ denote the number of neighbors of x in G that have earlier position numbers in τ. We can then compute $|N^-(x)|$ in D' as $i(x) - q(x)$. It takes $O(n+m)$ time to compute $q(x)$ for all $x \in V$, and hence $O(n+m)$ time to label each $x \in V$ with $|N^-(x)|$ in D'.

If the condition of Lemma 6.7 holds, then $D \cup D'$ is acyclic. Otherwise, there exist $x, y \in V$ such that $p(x) = p(y)$. Without loss of generality, suppose that $(x, y) \in D \cup D'$. Since y has x as a predecessor, and x and y have the same number of predecessors, then in $D \cup D'$, x must have a predecessor z that y does not have. In $O(n)$ time, we may list the predecessors of x in D and in D', do the same for y, and compare these two lists to find such a z. Then (x, y), (y, z), (z, x) is a three-cycle. □

By symmetry, Lemma 6.8 also applies to $D^T \cup D'$. The linear time bound for finding the incompatible pair now follows by Lemma 6.6.

6.3. Proof of Theorem 6.1, Theorem 6.2, and Lemma 6.3. In this subsection, we give a linear-time algorithm to find an odd cycle of length $O(n)$ in the incompatibility graph of G, given an incompatible pair in the orientation assigned to it by the transitive orientation algorithm. We show how to apply the algorithm to $\overline{G}$ in time linear in the size of G.

Let Γ be the relation on arcs, where $(u, w)\Gamma(x, y)$ if $u = x$ and w and y are nonadjacent or $w = y$ and u and x are nonadjacent. (This accepted term has nothing to do with the Γ's defined in section 4.) When $(u, w)\Gamma(x, y)$, any transitive orientation that contains one of the arcs must also contain the other. Let $G = (V, E)$ be an arbitrary undirected graph, and let A_G be its directed arcs. Let Γ_G be the graph (A_G, Γ) whose vertices are the arcs of G and whose edges are the pairs of elements in Γ.

DEFINITION 6.9. *A* transposed path *is a path in Γ_G between an arc (x, y) of G and its transpose (y, x).*

The following is well known.

THEOREM 6.10 (see [7, 8]). *An undirected graph G is a comparability graph iff it has no transposed path.*

LEMMA 6.11. *If there is a transposed path of length at most k in Γ_G, then there is an odd cycle in G's incompatibility graph of length at most $k+1$.*

Proof. Note that $e_1 \Gamma e_2$ iff $e_1 \neq e_2^T$ and $e_1 e_2^T$ is an edge of the incompatibility graph. A path $(e_0, e_1, \ldots, e_k)$ in Γ_G can be turned into a path in Γ_G by replacing each edge of odd index with its transpose. An odd-length path from e_0 to e_0^T in Γ maps to an odd-length cycle from e_0 to e_0 in the incompatibility graph, and an even-length path from e_0 to e_0^T in Γ maps to an even-length path from e_0 to e_0^T in the incompatibility graph, which, together with the edge (e_0^T, e_0), defines an odd cycle in the incompatibility graph. □

Proof of Theorem 6.1. The proof is immediate from Lemma 6.11.

Re-expressing Theorem 6.10 as Theorem 6.1 makes it immediately obvious to a user that the certificate proves that G has no transitive orientation. Since a skeptical user can check an edge of the incompatibility graph in $O(1)$ time, an odd cycle of size $O(n)$ in the incompatibility graph is a sublinear certificate that G is not a comparability graph if the odd cycle has size $O(n)$. However, Γ_G is useful for explaining the algorithm to generate the certificate. The constructive proof of Lemma 6.11 shows how to convert a transposed path to an odd cycle of the incompatibility graph in time proportional to the length of the transposed path.

A *module* of an undirected graph $G = (V, E)$ is a set X of vertices such that for each vertex $y \in V - X$, either every element of X is a neighbor of y or no member of X is a neighbor of y. V, the empty set, and the singleton subsets $\{\{x\}|x \in V\}$ are *trivial modules*. G is *prime* if it has only trivial modules. A set of vertices in G is a module iff it is a module in $\overline{G}$, and hence G is prime iff its complement is prime.

The problem of verifying that G is a comparability graph reduces in linear time to the problem of verifying that a set of prime induced subgraphs is a set of comparability graphs [7, 18]. A transposed path in an induced subgraph is also a transposed path in G. Therefore, when G or $\overline{G}$ is not a comparability graph, producing a transposed path in G or in $\overline{G}$ reduces in linear time to the same problem in the special case where G is prime.

THEOREM 6.12 (see [7, 8]). *Let G be a prime undirected graph. If G is not a comparability graph, then Γ_G has one connected component. Otherwise, Γ_G has two components, where one component contains the transposes of the arcs in the other component.*

We show how to modify the transitive orientation algorithm of [18] so that it creates a record that allows us to find a path of length $O(n)$ in Γ_G between any two arcs that are included in its orientation of a prime graph G.

The transitive orientation algorithm of [18] begins with a partition $\mathcal{P} = \{\{v\}, V - \{v\}\}$ of the vertices V of a prime graph G, where v is a selected *initial lone vertex*. In a process called *vertex partitioning*, it iteratively refines the partition using the following step, until $\mathcal{P}$ is the partition of V into one-element subsets:

- Select a vertex x as a *pivot*, and a partition class Y that does not contain x. Split Y into two classes, $Y_a = Y \cap N(x)$ of vertices that are *adjacent* to x and $Y_n = Y - N(x)$ of vertices that are *nonadjacent* to x. Let $\mathcal{P} := (\mathcal{P} - \{Y\}) \cup \{Y_a, Y_n\}$.

Figure 6 gives an example. Performing the first pivot on the initial lone vertex v splits $V - \{v\}$ into nonneighbors $\{c, f, g\}$ and neighbors $\{a, e, w\}$. Performing a pivot on w then splits the class $\{c, f, g\}$ into neighbors $\{c, f\}$ and nonneighbor $\{g\}$. Performing a pivot on $\{f\}$ then splits $\{a, e, w\}$ into nonneighbor $\{a\}$ and neighbors $\{e, w\}$, etc. Since G is prime, any partition class of size greater than one fails to be a module, so it is always possible to find a pivot that will split it. The vertex partitioning procedure halts when all partition classes are of size 1.

During the partitioning, a linear arrangement of the partition classes is maintained, so that when a set Y is split into two sets, Y_n and Y_a, these two sets occupy consecutive places at the former position of Y, with Y_a placed farther than Y_n from the partition class that contains the pivot. Initially, the lone vertex $\{v\}$ is placed first. For example, in Figure 6, when v splits $\{a, c, e, f, g, w\}$, it is in a class that *precedes* $\{a, c, e, f, g, w\}$, so the neighbors $\{a, e, w\}$ are placed *after* the nonneighbors, $\{c, f, g\}$. In the next step, when w then splits $\{c, f, g\}$, w is in a class that *follows* $\{c, f, g\}$, so

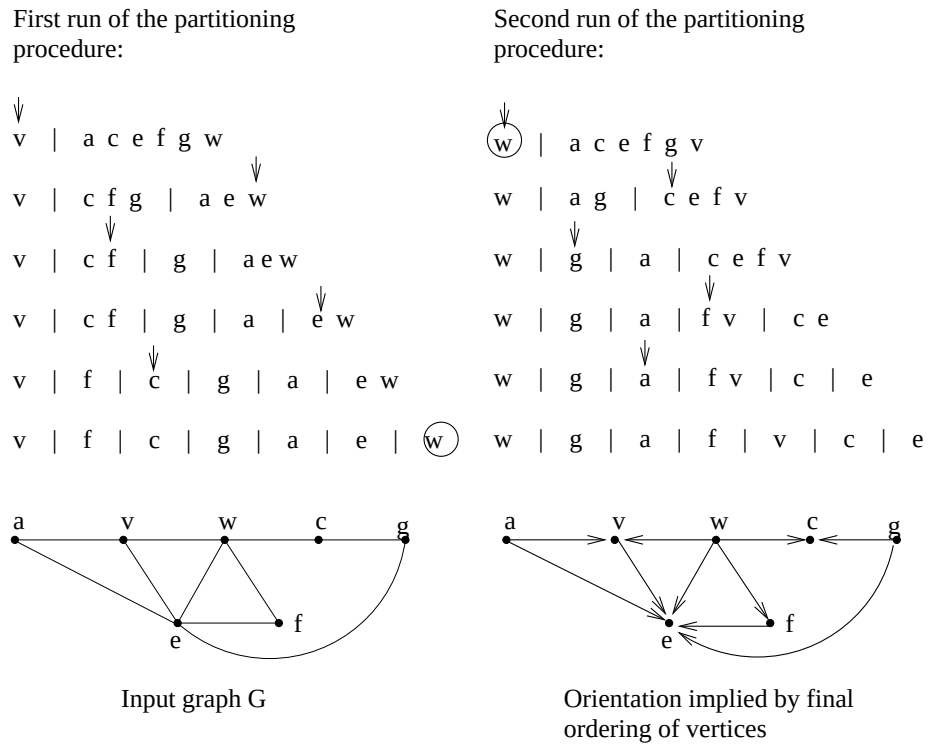


FIG. 6. *The transitive orientation algorithm of* [18] *performs two vertex partition refinements on a prime comparability graph in order to find a linear ordering of the vertices that gives a topological sort of a transitive orientation of the graph. The final ordering given by the second partition,* (w, g, a, f, v, e, c) *in this example, is the topological sort, which gives implicitly the transitive orientation of the graph. Arrows indicate pivot vertices that are used for the next refinement of the partition.*

the neighbors $\{c, f\}$ are placed before the nonneighbor $\{g\}$ in the ordering.

It is shown in [18] that if the initial lone vertex v is a source or sink in a transitive orientation of G, then the final ordering of vertices will be a topological sort of that transitive orientation. Moreover, it is shown that if v is an arbitrary vertex, then the rightmost vertex in the final ordering must be a source or sink in a transitive orientation of G. (The reasons will become apparent below.) Therefore, the procedure is run twice, once starting with arbitrary initial lone vertex v to identify a source/sink w, and once starting with w as the initial lone vertex to find a topological sort of the transitive orientation. Since the topological sort implies the orientations of the edges, this gives the transitive orientation if G is a comparability graph. In the illustration, the ordering (w, g, a, f, v, c, e) produced by the second run of the vertex partitioning gives the transitive orientation depicted in the graph at the bottom.

Let us now examine what happens when G is not a comparability graph. The procedure still produces an ordering of the vertices. Since G has no transitive orientation, this orientation must contain a pair (a, b) and (b, c) of directed edges such that (a, c) is not an edge, namely, an incompatible pair. An incompatible pair does not serve as a certificate that G is not a comparability graph, since it shows only that either G is not a comparability graph or the implementation of the transitive orientation algorithm has a bug.

We therefore seek a mechanism to turn an incompatible pair into a certificate

that G is not a comparability graph.

To accomplish this, we define a *parent* relation on the directed arcs of G that results from the vertex partitioning procedure. Suppose a set Y is split into Y_n and Y_a by a pivot vertex x. Let (y, z) be an arc from Y_n to Y_a. By the definition of Y_n and Y_a, xy is not an edge of G, and xz is, so $(y, z)\Gamma_G(x, z)$ and $(z, y)\Gamma_G(z, x)$. Let (x, z) be the *parent* of (y, z), and let (z, x) be the *parent* of (z, y).

LEMMA 6.13. *If arc a_1 is the parent of arc a_2, then $a_1\Gamma_G a_2$.*

Clearly, every arc has a unique parent except those that are incident to the initial lone vertex, which have no parents. At a given point in the partition refinement, let us say that a directed arc is *protected* if both of its endpoints are currently within a single partition class, and *exposed* if its endpoints are in two different partition classes. The parent relation is acyclic, since the parent is always exposed earlier than the child. All arcs are assigned a parent except those that are incident to the initial lone vertex, so the parent relation arising from one run of the vertex partitioning procedure defines a forest of rooted trees on the arcs of G, and the roots of these trees are the arcs incident to the initial lone vertex.

Let P_1 be the parent relation arising from the first run of the vertex partitioning procedure, which begins with partition $\{\{v\}, V - \{v\}\}$ and discovers a source/sink w. Let P_2 be the parent relation arising from the partition $\{\{w\}, V - \{w\}\}$. The arcs incident to v are the tree roots in P_1, and the arcs incident to w are the tree roots in P_2. Therefore, the only arcs that fail to have parents in both P_1 and P_2 are (v, w) and (w, v). Moreover, since w is in the rightmost class after every pivot during the first run of the partitioning procedure, every time it is in a class that is split by a pivot, it is in the class that contains neighbors of the pivot. Since it is a neighbor of the pivot, the parent of each edge incident to w that gets exposed by the pivot is also incident to w.

Therefore, all arcs incident to w lie in the two trees of P_1 that are rooted at (v, w) and (w, v). Let P_1' be the restriction of P_1 to arcs incident to w; that is, P_1' is two trees rooted at (v, w) and (w, v) that span the arcs incident to w.

It follows that $P_1' \cup P_2$ consists of exactly two trees $T_{(v,w)}$ and $T_{(w,v)}$ that are rooted at (v, w) and (w, v) and that span all directed arcs of G. Whenever a_1 and a_2 are arcs of G where a_1 is a parent of a_2, $a_1\Gamma_G a_2$. By Lemma 6.13, any transitive orientation of G that contains (v, w) must contain every arc in $T_{(v,w)}$ and any transitive orientation of G that contains (w, v) must contain every arc in $T_{(w,v)}$. The arcs spanned by the two trees are the two transitive orientations of G if G has a transitive orientation. Let us therefore call $T_{(v,w)}$ and $T_{(w,v)}$ the *orientation trees*.

Figure 7 depicts one of the two orientation trees produced by the two runs of the vertex partitioning algorithm in Figure 6. Each node is labeled with two vertices and represents the arc from the first vertex label to the second. (The other orientation tree is identical except that the directions of all of the edges are reversed.) The parent relation on arcs determined by the second run (P_2) is shown with solid edges; the parent relation on edges incident to w determined by the first run (P_1') are shown with dashed edges.

Given a parent arc a_1 and child arc a_2, it is easy to check in $O(1)$ time that $a_1\Gamma_G a_2$. Performing this on all parent-child pairs allows a user to confirm in $O(n + m)$ time that *if* G has a transitive orientation, then it must be the orientation consisting of the nodes of one of the two trees. It therefore serves as a certificate either that the precondition that G was a comparability graph was violated or that the orientation is transitive, without identifying which of these two cases occurred. In either case, it exonerates an implementation of the transitive orientation algorithm of providing a

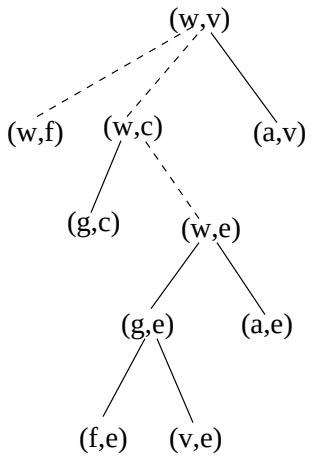


FIG. 7. *The orientation tree defined by the two runs of the vertex partitioning procedure shown in Figure* 6. *Solid edges depict the parent relation implied by the second run of the partitioning procedure and form a forest of trees rooted at arcs incident to* w. *Dashed edges give the parents of edges incident to* w *that are implied by the first run of the partitioning procedure. Together, these edges link this forest together into a tree rooted at* (w, v).

nontransitive orientation of a comparability graph.

In contrast to a simple transitive orientation, the orientation trees contain information that allows us to find a transposed path if G is not a comparability graph. In this case, each of the trees must contain two edges (a, b) and (b, c) that form an incompatible pair, which implies that ac is not an edge of G, and $(b, a)\Gamma_G(b, c)$.

Suppose without loss of generality that (a, b) and (b, c) occur within $T_{(v,w)}$. To document that G is not a comparability graph, we may find the least common ancestor (d, e) of (a, b) and (b, c) in $T_{(v,w)}$. Let $P_{(a,b)}$ be the path from (a, b) to (d, e) and $P_{(b,c)}$ be the path from (b, c) to (d, e) in $T_{(v,w)}$. $P_{(a,b)} \cup P_{(b,c)}$ defines a path in Γ_G from (a, b) to (b, c). Taking these together with $(b, a)\Gamma_G(b, c)$, we get a transposed path from (a, b) to (b, a)—a certificate that G is not a comparability graph.

For instance, suppose edge ae is removed from the graph of Figure 6. The resulting graph is no longer a comparability graph, but it is easy to see that the removal of this particular edge does not affect any of the steps of either of the two runs of the vertex partitioning procedure. Therefore, the transitive orientation algorithm still produces the topological sort (w, g, a, f, v, c, e) of an orientation that contains the incompatible pair $((a, v), (v, c))$. The least common ancestor of (a, v) and (v, e) in the orientation tree of Figure 7 is (w, v). The union of the paths from (a, v) and (v, c) to this least common ancestor forms the path $((v, e), (g, e), (w, e), (w, c), (w, v), (a, v))$ which, together with $((e, v)\Gamma_G(a, v))$, is a transposed path $((v, e), (g, e), (w, e), (w, c), (w, v), (a, v), (e, v))$ from (v, e) to (e, v). A skeptic can check each of the links of this path once they are pointed out and conclude that the graph is not, in fact, a comparability graph.

$T_{(v,w)}$ and $T_{(w,v)}$ have size $O(m)$; we now show how to construct them in $O(m)$ time. For this, we modify the vertex partitioning procedure to produce the data structure pictured on the right side of Figure 8. The construction is illustrated for the run of the vertex partitioning procedure, where w is the initial lone vertex; the data structure, where v is the initial lone vertex, is constructed in the same way. The data structure is a tree whose nodes represent subsets of V. The root of the tree is V, its children are $\{w\}$ and $V - \{w\}$ if w is the initial lone vertex, and the remaining

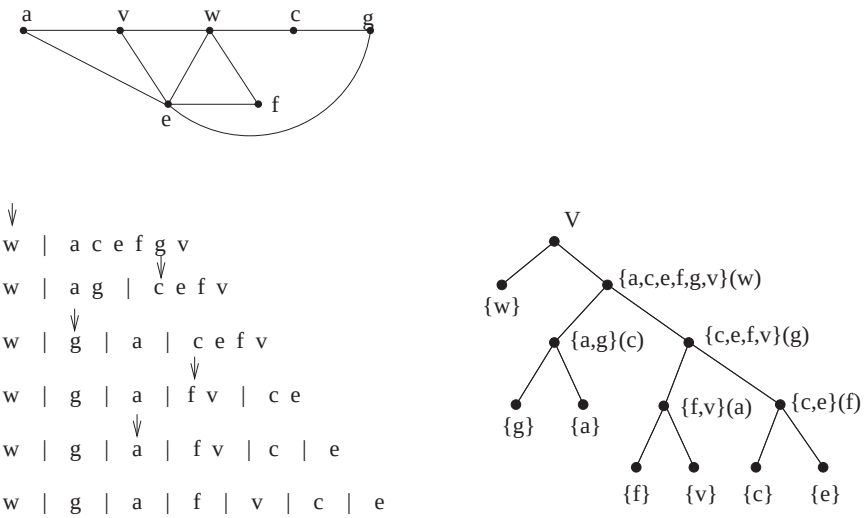


FIG. 8. *The Hasse diagram of a run of the vertex partitioning procedure. If a set Y is split into two sets Y_n and Y_a by a pivot, Y_n and Y_a are its children. In the figure, each set is labeled with its members, as well as the pivot that split it if it is an internal node. For the data structure, it is not necessary to label internal nodes with their members, so the data structure requires $O(n)$ space.*

nodes are the remaining sets that appear at some point during the vertex partitioning procedure. When a partition class Y is split into Y_n and Y_a by a pivot x, Y_n and Y_a are the children of Y. In other words, the tree is the Hasse diagram of the subset relation on the partition classes that appear at some point during the refinement. Let us refer to it as the *Hasse diagram* of the run of the partition refinement algorithm.

Each node is labeled with the identity of the pivot, in parentheses, that caused it to split into its two children during the refinement. For instance, when $\{c, e, f, v\}$ is split into $\{f, v\}$ and $\{c, e\}$ by pivot g, $\{f, v\}$ and $\{c, e\}$ become the children of $\{c, e, f, v\}$, and $\{c, e, f, v\}$ is labeled with the pivot (g) that split it.

To represent this tree with a data structure, we label each leaf with its sole member, and label each internal node only with the pivot that caused it to split, but not with a list of members. The members of the set X represented by an internal node can be found in $O(|X|)$ time by visiting its leaf descendants, which is just as efficient as labeling the node explicitly with the members of X. This allows each internal node to take $O(1)$ space, so the data structure for the Hasse diagram takes $O(n)$ space. The time to construct it does not affect the $O(n + m)$ time to run the vertex partitioning procedure, since it requires creating two children of size $O(1)$ each time a partition class is split by the procedure.

LEMMA 6.14. *Suppose (a, b) is an arc of G. Let Y be the least common ancestor of $\{a\}$ and $\{b\}$ in the Hasse diagram.*

- *If $Y = V$, then (a, b) has no parent.*
- *Otherwise, let c be the pivot that split Y into nonneighbors Y_n and neighbors Y_a of c. Then if $a \in Y_n$ and $b \in Y_a$, then (c, b) is the parent of (a, b), and if $b \in Y_a$ and $a \in Y_n$, then (a, c) is the parent of (a, b).*

Proof. The proof is immediate from the definition of the parent function. □

COROLLARY 6.15. *Given the Hasse diagram and a set A of arcs of G, it takes $O(|A| + n)$ time to find the parents of the members of A.*

Proof. Numbering the leaves from 1 to n in left-to-right order allows one to find, for any two vertices, which is earlier on the leaf order in $O(1)$ time. By the off-line least common ancestors algorithm of Harel and Tarjan [10], given k pairs of nodes in a rooted tree with $O(n)$ nodes, it takes $O(k+n)$ time to find the least common ancestor of each of the k pairs. By the rule for ordering partition classes, for an arc (a,b) with a least common ancestor split by pivot c, $a \in Y_n$ if a is in between b and c in this order, and $b \in Y_n$ otherwise. In the former case, (c,b) is the parent and, in the latter case, (a,c) is the parent. □

COROLLARY 6.16. *It takes $O(n+m)$ time to find the two orientation trees implied by the transitive orientation algorithm.*

Proof. It takes $O(n+m)$ time to get the parents of all arcs not incident to the lone initial vertex w in the parent relation defined by the second run of the partitioning procedure by Corollary 6.15, and it takes $O(n)$ time to get the parents of arcs incident to w, but not v in the first run. The union of these two sets of parent pointers is formed by the two orientation trees. □

Since the arcs of G are nodes of the orientation trees, these trees have $\Theta(m)$ nodes. However, we can now state the following.

COROLLARY 6.17. *The orientation trees have height $O(n)$.*

Proof. Recall that an arc is *exposed* during partitioning at the point when its endpoints are separated into two different partition classes. Each time a partition class splits, the number of partition classes increases by one, and this number is initially equal to one when the initial lone vertex is separated from V and equal to n when the partitioning procedure terminates. Therefore, classes are split at most $n-1$ times. When a split of a class exposes an arc, this means that the parent of the arc was exposed by an earlier split. Therefore, there is no chain longer than $n-1$ in the parent relation implied by one run of the partitioning procedure.

Each path from an arc of G to the arc (w,v) or (v,w) that is the root of the orientation tree follows zero or more parent pointers defined by the second run of the partitioning procedure to arrive at an arc incident to w, followed by zero or more parent pointers defined by the first run of the partitioning procedure, through arcs incident to w, to arrive at (w,v) or (v,w). The height of the tree is therefore at most $2n-2$. □

THEOREM 6.18. *Given an incompatible pair $((a,b),(b,c))$ in the orientation of a graph G produced by the transitive orientation algorithm, it takes $O(n+m)$ time to find an odd cycle of length $O(n)$ in G's incompatibility graph.*

Proof. It takes $O(n+m)$ time to find the orientation tree that contains (a,b) and (b,c) by Corollary 6.16. It takes $O(n)$ time to find the path from (a,b) to (b,c) in this tree by Corollary 6.17. By Lemma 6.13, this path, together with $(b,c)\Gamma_G(b,a)$, is a transposed path from (a,b) to (b,a). The constructive proof of Lemma 6.11 shows how to turn this into an odd cycle of the incompatibility graph of size $O(n)$. □

Proof of Theorem 6.2. The proof is immediate from Theorem 6.18 and Lemma 6.11.

Note that we do not claim an $O(n+m)$ certifying algorithm for recognizing comparability graphs, and no such bound is known for recognizing them, with or without a certificate. The bottleneck for recognition of comparability graphs is finding an incompatible pair in the orientation produced by the algorithm of [18]. It is noteworthy, however, that this gives a certifying algorithm for recognizing comparability graphs that is as fast as any currently known, and that produces a sublinear certificate of rejection.

LEMMA 6.19 (see [18]). *Given a graph G with n vertices and m edges, it takes*

$O(n+m)$ time to find an ordering of the vertices that is a topological sort of a transitive orientation of $\overline{G}$ if $\overline{G}$ is a comparability graph.

The algorithm works by symmetry in the roles of edges and nonedges. It performs the pivots exactly as it does for orienting G, but reverses the roles of the set Y_n of nonneighbors and the set Y_a of neighbors of the pivot. The only effect of this is that it reverses the relative order of Y_n and Y_a when we replace Y with these two sets in the ordering of partition classes. This trick is used in [18] to get linear-time recognition of permutation graphs.

However, one difficulty we now face in producing a certificate that $\overline{G}$ is not a comparability graph in this time bound is that the number of arcs in G, hence the number of nodes of its orientation trees, is $\Theta(n^2 - m)$, which does not conform to our desired $O(n + m)$ time bound. We cannot construct the orientation trees for $\overline{G}$ and stay within our time bound.

Fortunately, given an incompatible pair $((a, b), (b, c))$, the construction of Theorem 6.18 requires us only to find the paths from (a, b) and (b, c) to their least common ancestor in the orientation tree, not the whole orientation tree. These paths have length $O(n)$ by Corollary 6.17.

LEMMA 6.20. *Given an incompatible pair $((a, b), (b, c))$ in the orientation of $\overline{G}$ produced by the algorithm of Lemma* 6.19*, it takes $O(n)$ time to find an odd cycle of size $O(n)$ in the incompatibility graph of $\overline{G}$.*

Proof. Creation of the Hasse diagram is not affected by the modification of the partitioning algorithm for Lemma 6.19. To find the parent of an arc (a, b) of $\overline{G}$ in one run of the partitioning algorithm, mark the ancestors of a in the Hasse diagram. This takes $O(n)$ time. Search upward from b until a marked node is encountered. This is the least common ancestor A_1 of $\{a\}$ and $\{b\}$ in the Hasse diagram. Suppose by induction that the least common ancestor A_k in the Hasse diagram has been found for $\{x\}$ and $\{y\}$, where (x, y) is some ancestor of (a, b) in the parent relation. Let z be the pivot label of this ancestor. Search upward from z until a marked node of the Hasse diagram is encountered. This is the least common ancestor A_{k+1} in the Hasse diagram of the parent of (x, y) in T_1. Since this A_{k+1} is higher in the Hasse diagram than any other least common ancestor found so far, the search upward from z uses a different set of edges of the Hasse diagram from those used by previous upward searches. The cost of this search can be charged to the edges traversed during the search. The cost of finding all ancestors in the Hasse diagram is $O(n)$, and the length of the path is $O(n)$. □

Acknowledgments. The authors would like to thank the two anonymous referees for generous observations, questions, and comments that we have made ample use of in the paper.

REFERENCES

[1] M. BLUM AND S. KANNAN, *Designing programs that check their work,* in Proceedings of the 21st Symposium on the Theory of Computation, ACM, New York, 1989, pp. 86–97.

[2] K. S. BOOTH AND G. S. LUEKER, *Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms*, J. Comput. System Sci., 13 (1976), pp. 335–379.

[3] A. BRANDSTÄDT, V. B. LE, AND J. P. SPINRAD, *Graph Classes: A Survey*, SIAM Monogr. Discrete Math. Appl. 3, SIAM, Philadelphia, 1999.

[4] D. G. CORNEIL, S. OLARIU, AND L. STEWART, *The ultimate interval graph recognition algorithm?*, in Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1998, pp. 175–180.

[5] D. G. Corneil, Y. Perl, and L. K. Stewart, *A linear recognition algorithm for cographs*, SIAM J. Comput., 14 (1985), pp. 926–934.
[6] G. A Dirac, *On rigid circuit graphs*, Abh. Math. Sem. Univ. Hamburg, 25 (1961), pp. 71–76.
[7] T. Gallai, *Transitiv orientierbare Graphen*, Acta Math. Acad. Sci. Hungar., 18 (1967), pp. 25–66.
[8] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.
[9] M. Habib, R. M. McConnell, C. Paul, and L. Viennot, *Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition, and consecutive ones testing*, Theoret. Comput. Sci., 234 (2000), pp. 59–84.
[10] D. Harel and R. E. Tarjan, *Fast algorithms for finding nearest common ancestors*, SIAM J. Comput., 13 (1984), pp. 338–355.
[11] W. L. Hsu, *A simple test for interval graphs*, in Graph-theoretic Concepts in Computer Science, Lecture Notes in Comput. Sci. 657, Springer, Berlin, 1993, pp. 11–16.
[12] W. L. Hsu and R. M. McConnell, *PC trees and circular-ones arrangements*, Theoret. Comput. Sci., 296 (2003), pp. 99–116.
[13] N. Korte and R. H. Möhring, *An incremental linear-time algorithm for recognizing interval graphs*, SIAM J. Comput., 18 (1989), pp. 68–81.
[14] D. C. Kozen, *The Design and Analysis of Algorithms*, Springer, Berlin, 1991.
[15] C. Lekkerkerker and D. Boland, *Representation of finite graphs by a set of intervals on the real line*, Fund. Math., 51 (1962), pp. 45–64.
[16] A. Lubiw, *Doubly lexical orderings of matrices*, SIAM J. Comput., 16 (1987), pp. 854–879.
[17] R. M. McConnell and F. de Montgolfier, *On the Common Factors in a Set of Linear Orders*, Technical report CS-04-102, Colorado State University, Fort Collins, CO, 2004.
[18] R. M. McConnell and J. P. Spinrad, *Modular decomposition and transitive orientation*, Discrete Math., 201 (1999), pp. 189–241.
[19] K. Mehlhorn and S. Näher, *The LEDA Platform for Combinatorial and Geometric Computing*, Cambridge University Press, Cambridge, UK, 1999.
[20] K. Mehlhorn, S. Näher, T. Schilz, M. Seel, R. Seidel, and C. Uhrig, *Checking geometric programs or verification of geometric structures*, Comput. Geom., 12 (1999), pp. 85–103.
[21] K. Mehlhorn, S. Näher, and C. Uhrig, *The LEDA platform for combinatorial and geometric computing*, in Proceedings of the 24th International Colloquium on Automata, Languages, and Programming (ICALP '97), Lecture Notes in Comput. Sci. 1256, Springer-Verlag, Berlin, 1997, pp. 7–16.
[22] R. Paige and R. E. Tarjan, *Three partition refinement algorithms*, SIAM J. Comput., 16 (1987), pp. 973–989.
[23] A. Pnueli, A. Lempel, and S. Even, *Transitive orientation of graphs and identification of permutation graphs*, Canad. J. Math., 23 (1971), pp. 160–175.
[24] F. S. Roberts, *Graph Theory and Its Applications to Problems of Society*, SIAM, Philadelphia, 1978.
[25] D. J. Rose, R. E. Tarjan, and G. S. Lueker, *Algorithmic aspects of vertex elimination on graphs*, SIAM J. Comput., 5 (1976), pp. 266–283.
[26] J. P. Spinrad, *Doubly lexical ordering of dense* 0–1 *matrices*, Inform. Process. Lett., 45 (1993), pp. 229–235.
[27] R. E. Tarjan and M. Yannakakis, *Addendum: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs*, SIAM J. Comput., 14 (1985), pp. 254–255.
[28] H. Wasserman and M. Blum, *Software reliability via run-time result-checking*, J. ACM, 44 (1997), pp. 826–849.

SIAM J. COMPUT.
Vol. 36, No. 2, pp. 354–393

LOWER BOUNDS FOR ON-LINE GRAPH PROBLEMS WITH APPLICATION TO ON-LINE CIRCUIT AND OPTICAL ROUTING*

YAIR BARTAL†, AMOS FIAT‡, AND STEFANO LEONARDI§

Abstract. We present lower bounds on the competitive ratio of randomized algorithms for a wide class of on-line graph optimization problems, and we apply such results to on-line virtual circuit and optical routing problems. Lund and Yannakakis [*The approximation of maximum subgraph problems*, in Proceedings of the 20th International Colloquium on Automata, Languages and Programming, 1993, pp. 40–51] give inapproximability results for the problem of finding the largest vertex induced subgraph satisfying any nontrivial, hereditary property π—e.g., independent set, planar, acyclic, bipartite. We consider the on-line version of this family of problems, where some graph G is fixed and some subgraph H of G is presented on-line, vertex by vertex. The on-line algorithm must choose a subset of the vertices of H, choosing or rejecting a vertex when it is presented, whose vertex induced subgraph satisfies property π. Furthermore, we study the on-line version of graph coloring whose off-line version has also been shown to be inapproximable [C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, in Proceedings of the 25th ACM Symposium on Theory of Computing, 1993], on-line max edge-disjoint paths, and on-line path coloring problems. Irrespective of the time complexity, we show an $\Omega(n^\epsilon)$ lower bound on the competitive ratio of randomized on-line algorithms for any of these problems. As a consequence, we obtain an $\Omega(n^\epsilon)$ lower bound on the competitive ratio of randomized on-line algorithms for virtual circuit routing on general networks, in contrast to the known results for some specific networks. Similar lower bounds are obtained for on-line optical routing as well.

Key words. on-line computation, graph problems, network optimization, competitive analysis, randomized algorithms, lower bounds

AMS subject classifications. 68W20, 90B18, 05C85

DOI. 10.1137/S009753979833965X

1. Introduction.

1.1. On-line graph problems.

On-line graph optimization problems have been previously considered in several works: the problem of obtaining a maximal on-line matching in a bipartite graph is considered in [KVV90], the on-line coloring of a three colorable graph is studied in [V90], the problem of coloring an inductive graph is studied in [I90], and randomized lower bounds for on-line graph coloring are given in [HS92].

In all the previous papers, vertices and/or edges arrive on-line, and the on-line algorithm has to make some relevant decision when this happens. The adversary has total freedom to decide what graph will be presented, and in what order the edges/vertices will arrive.

In this paper, we consider a different model, where the graph presented by the

*Received by the editors June 1, 1998; accepted for publication (in revised form) February 5, 2006; published electronically June 23, 2006. A preliminary version of this work appeared in the Proceedings of the 28th Annual ACM Symposium on Theory of Computing, 1996.

http://www.siam.org/journals/sicomp/36-2/33965.html

†School of Computer Science, Hebrew University, Jerusalem (yairb@cs.huji.ac.il). This author's research was supported in part by a Rotschild Postdoctoral fellowship.

‡Department of Computer Science, Tel Aviv University, Tel Aviv (fiat@math.tau.ac.il). This author's research was supported in part by two grants from the Israel Academy of Sciences.

§Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza," Rome, Italy (leon@dis.uniroma1.it). This author's work was partly supported by EU ESPRIT Long Term Research Project ALCOM-IT under contract 20244, and by Italian Ministry of Scientific Research Project 40% "Algoritmi, Modelli di Calcolo e Strutture Informative."

adversary is a subgraph of some (larger) graph that is *known in advance* to the on-line algorithm. Upper bounds that hold for the models of [KVV90, V90, I90, HS92] clearly also hold for the model where the presented graph is a subgraph of some known graph. Lower bounds do not. Lower bounds for the unknown graph model can be trivially translated into our model simply by considering a much larger graph that contains all possible subgraphs. However, the large graph is now exponentially larger than the subgraph, which implies that the lower bound is exponentially smaller.

The motivation for this model is not only theoretical. It allows one to capture problems such as on-line routing in networks. We effectively translate routing problems onto graph problems on a graph related to the original network. It is reasonable to assume that the on-line routing algorithm does not know anything about future communication requests. It is not reasonable to assume that it does not know anything about the routing network itself.

We consider a large set of on-line graph optimization problems. Some of the problems are benefit problems (e.g., find the largest subgraph satisfying a property), and others are cost problems (e.g., color a graph with as few colors as possible).

The graph optimization problems that we consider are listed below. For each such problem we first give the off-line version and then give the on-line variant. The performance of an algorithm for an on-line problem is measured by its *competitive ratio* [ST85]. The competitive ratio for benefit problems is defined as the worst case ratio between the optimal off-line benefit and the on-line benefit. For cost problems the competitive ratio is defined as the worst case ratio between the on-line cost and the optimal off-line cost.

Off-line induced subgraph. Let π be a graph property that is *nontrivial* (i.e., true for infinitely many graphs and false for infinitely many graphs) and *hereditary* (i.e., if a graph G satisfies it, then also any vertex induced subgraph of G satisfies it). Given a graph $G = (V, E)$, find the subset of V of maximum cardinality whose vertex induced subgraph satisfies π.

Such problems include complete graph, independent set, k-colorable, planar, outerplanar, bipartite, complete bipartite, acyclic, degree-constrained, interval, circular-arc, circle graph, chordal, perfect, etc.

Lund and Yannakakis [LY93a] give inapproximability results for this family of problems. Note that in on-line computation, time complexity is not considered, and thus our results do not follow from [LY93a].

On-line induced subgraph. The graph $G = (V, E)$ is known to the on-line algorithm. Vertices $v \in V$ are presented in arbitrary order. The adversary may choose to stop the sequence at any time. The on-line algorithm has to decide whether to accept v or not. The induced graph of the accepted vertices must satisfy property π. The benefit of the on-line algorithm is the cardinality of the set of accepted vertices.

We define the *on-line independent set* problem as the on-line induced subgraph problem for the hereditary property $\pi =$ *independent set*.

Off-line graph coloring. Given a graph $G = (V, E)$, color its vertices with as few colors as possible so that no two adjacent nodes receive the same color.

Lund and Yannakakis [LY93b] also prove inapproximability results for minimum graph coloring.

On-line graph coloring. The graph $G = (V, E)$ is known to the algorithm. At every step, a new vertex $v \in V$ is presented to the on-line algorithm and assigned a color. The adversary may choose to stop the sequence at any time. The coloring must be valid. The cost of the on-line algorithm is given by the number of colors used.

The on-line graph coloring problem has been studied extensively, and determin-

istic and randomized algorithms have been proposed, but only in the setting where the graph is unknown [I90, V90, HS92].

Off-line edge-disjoint paths. The instance is a graph $G = (V, E)$ and a collection $\mathcal{C} = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ of k pairs of vertices. We say that a set of pairs $\{(s_{i_j}, t_{i_j})\}_{j=1}^{m}$ is *consistent* if there exist edge-disjoint paths $\{p_{i_j}\}_{j=1}^{m}$, where p_{i_ℓ} connects s_{i_ℓ} to t_{i_ℓ}. The problem is to find a maximum cardinality consistent subset of $\mathcal{C}$.

On-line edge-disjoint paths. The graph G is known to the algorithm, and at every step the adversary presents a pair of vertices to the on-line algorithm. The on-line algorithm may accept or reject this pair, and for every accepted pair must commit to a route connecting the two vertices, but the set of all accepted pairs must be consistent. The benefit of the on-line algorithm is here defined as the cardinality of the set of accepted pairs.

A variation of this problem is the edge-disjoint fixed paths problem, where an entire path, rather than just its endpoints, is given. The goal is to accept a maximal set of pairwise nonoverlapping paths.

Off-line path coloring. Given a graph $G = (V, E)$ and a collection $\mathcal{C} = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ of k pairs of vertices in G, what is the minimum number of consistent sets of pairs so that their union gives $\mathcal{C}$? This is equivalent to minimizing the number of colors required to color paths between the k endpoints (s_i, t_i) so that no edge is contained in two paths colored with the same color.

On-line path coloring. The graph $G = (V, E)$ is known to the on-line algorithm, and at every step the adversary presents a pair of vertices. The on-line algorithm chooses a color for the two vertices and commits to a route connecting the two vertices. The set of pairs accepted with the same color has to be consistent. The on-line cost is the number of colors used.

For all of the problems above we prove a lower bound of $\Omega(n^\epsilon)$ on the competitive ratio of any on-line algorithm, for some $\epsilon > 0$. This is fairly easy to show for deterministic algorithms; our main contribution is showing that this holds even for randomized on-line algorithms against an oblivious adversary (cf. [BBKTW90]).[1]

We complement these lower bounds results by presenting an algorithm for the on-line induced subgraph problem that achieves a competitive ratio of $O(\frac{n}{\sqrt{\alpha}})$, given a black-box $O(\frac{n}{\alpha})$-approximation algorithm for the induced subgraph problem, and an $O(\frac{n}{\sqrt{\alpha}})$-competitive algorithm for graph coloring if an $O(\frac{n}{\alpha})$-approximation algorithm for independent sets is available.

1.2. Virtual circuit routing problems. The benefit or throughput version of the virtual circuit routing problem is where communication requests require the assignment of a specified bandwidth on a path that connects a transmitter to a receiver, for a given duration and offering a given benefit. Each call can be either accepted and scheduled, observing the bandwidth constraints on all links, or rejected. Following [GG92], we call this problem the *call control* problem.

[1] In a recent paper Kaplan and Szegedy [KS98] pointed out an error in our original proof of Lemma 10 (presented as the proof of an analogous claim in [BFL96]) and gave a corrected version. A correct proof of Lemma 10, along slightly different lines from those in [KS98], is also presented in this paper.

In the conference version of this paper [BFL96] we showed that similar lower bounds also hold in the context of benefit problems for the on-line independent set and the on-line edge-disjoint paths problems when the use of *preemption* is allowed to the randomized algorithm. This means that the on-line algorithm is allowed to discard a previously chosen element (a vertex or a pair) but cannot later change its mind again. These results can be viewed in the on-line version of that report (see [BFL96]).

[GGKMY93] considered the call control problem when the algorithm is allowed to preempt previously accepted calls. They give deterministic preemptive algorithms with a competitive ratio of $O(\log n)$ for a line network, and show that no deterministic algorithm can achieve a better ratio. [BCK+95] studied a preemptive algorithm for a single link and a line topology when the benefit of a call is proportional to the bandwidth duration product. Lower bounds for randomized preemptive algorithms for a single link were given in [CI95]. The authors show an $\Omega\left(\sqrt{\frac{\log \mu}{\log \log \mu}}\right)$ lower bound, where μ is either the ratio between the maximum and the minimum duration or the ratio between the maximum and the minimum benefit of a call.

Nonpreemptive call control was first considered in [AAP93]. They gave throughput competitive algorithms, $O(\log n)$ competitive, for arbitrary networks (not requiring preemption), if every call request bandwidth was for no more than a logarithmic fraction of the minimal link capacity. They also prove an $\Omega(n)$ lower bound on the competitive ratio of deterministic on-line algorithms on general networks if calls may request arbitrary bandwidth.

[ABFR94] considered randomized nonpreemptive algorithms for call control, on tree networks, getting a competitive ratio of $O(\log n)$ when all edge capacities are 1, all bandwidth requests are 1, all durations are infinite, and all benefits are 1. [ABFR94] also showed that problem parameters such as variable bandwidth, call benefit, and call duration can be dealt with by a randomized algorithm at the expense of logarithmic factors in the competitive ratio.

The call control problem when all links have capacity 1, all calls require bandwidth 1, duration is infinite, and all call benefits are 1 is thus a central problem to be considered. This problem is equivalent to the edge-disjoint paths graph problem stated above. Thus, any lower bound on the competitive ratio for edge-disjoint paths is a lower bound for the call control problem. Similarly, an upper bound on the competitive ratio for edge-disjoint paths gives an upper bound on the more general problem using the techniques of [ABFR94].

[AGLR94] give a randomized call control algorithm for the mesh, with a competitive ratio of $O(\log^2 n)$ and an $O(\log D)$ competitive algorithm for trees, where D is the diameter of the tree. In a later paper, [KT95] gave an $O(\log n)$ competitive randomized call control algorithm for the mesh, and extended this result to more general "densely embedded nearly Euclidean" planar graphs. The issue of the variance of randomized on-line algorithms was considered in [LMPR98]. The authors propose randomized on-line competitive algorithms for trees and meshes that both achieve optimal competitive ratio and produce a solution concentrated with good probability around the expectation.

We show an $\Omega(n^{\epsilon})$ lower bound on the competitive ratio of nonpreemptive randomized on-line algorithms for general networks. We give specific networks where this occurs, including a network of degree $\Omega(\log n)$.

1.3. Routing problems in optical networks. In optical networks the bandwidth available on any link is split into many channels, each at a different wavelength. Each transmission is associated with exactly one wavelength that must be available on all links forming a path from the transmitter to the receiver.

The routing problem on optical networks varies considerably, depending on the network model considered. There are a number of models, differing in the nature of network switches.

The basic classification is into the following types:

- Nonreconfigurable networks—also called passive or *switchless* networks—

where a wavelength originating from a transmitting node will always follow a fixed pattern. There may be "switches" that mix and match incoming wavelengths to outgoing links, but these cannot be changed during the operation of the network.

- Reconfigurable networks, which use two different types of switches:
  - elementary switches, which can direct signals coming along one of the input lines to one or more output links, but which cannot differentiate between wavelengths when performing the switch;
  - generalized switches, which are capable of splitting the incoming stream based upon the wavelength.

In all models, the key restriction is that if more than one data item is transmitted along an edge on the same wavelength (color), then they are all unusable.

Optical routing was considered in [RU94, ABC$^+$94, BH93, P92]. Most of that work focuses on the question of how many wavelengths are required to accommodate a given communication traffic.

We consider a number of on-line optical routing problems on both *reconfigurable* optical networks, where all vertices are generalized switches, and *switchless* optical networks. We consider benefit problems, where calls can be accepted or rejected, analogous to the call control problem, and coloring problems, whose goal is to minimize the number of wavelengths (colors) used.

On-line routing in reconfigurable optical networks—benefit version. Calls are presented one-by-one, and calls can be either accepted or rejected. Every accepted call is assigned with a wavelength and can be routed using the generalized switches along a path to be chosen. All calls transmitted along the same wavelength cannot share an edge. If only one wavelength is available, this problem is identical to on-line edge-disjoint paths.

On-line routing in reconfigurable optical networks—coloring version. Calls are presented one-by-one, and all calls have to be accepted and assigned a wavelength and a path linking source to destination. The goal is to minimize the number of wavelengths used, in order to preserve the rule that no two calls on *the same wavelength* can share an edge. The problem is identical to on-line path coloring.

In the switchless cases described below, the routing decisions are very restricted. What does have to be decided is the wavelength to be assigned to any call so that calls do not interfere with one another. Switchless optical networks are modeled as directed acyclic graphs in order to prevent self-interference of transmissions.

Off-line routing in switchless optical networks—benefit version. Consider a directed graph $D = (V, E)$ and a collection $\mathcal{C} = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ of k pairs of vertices. Denote by $R(v)$ the set of vertices of V reachable from v in D via a directed path. We say that two pairs (s_i, t_i), (s_j, t_j) of terminal vertices are *noninterfering* if $t_j \notin R(s_i)$ and $t_i \notin R(s_j)$. The problem on a single wavelength consists of finding a maximum cardinality pairwise noninterfering subset of $\mathcal{C}$.

On-line routing in switchless optical networks—benefit version. The graph D is known to the algorithm. At every step a pair is communicated to the on-line algorithm, which can either accept or reject it; the set of accepted pairs should be pairwise noninterfering. The benefit of the on-line algorithm is the cardinality of the set of accepted calls.

Off-line routing in switchless optical networks—coloring version. Given a directed graph $D = (V, E)$ and a collection $\mathcal{C} = \{(s_i, t_i), \ldots, (s_k, t_k)\}$ of pairs of vertices, what is the minimum number of pairwise noninterfering sets of pairs so that their union

gives $\mathcal{C}$? This can be seen minimizing the number of colors necessary to color the pairs in $\mathcal{C}$, where two pairs with the same color must be noninterfering.

On-line routing in switchless optical networks—coloring version. The on-line algorithm knows some graph D. At each step a new pair of vertices is chosen by the adversary and assigned a color by the on-line algorithm. The on-line algorithm assigns a color so that the pair is noninterfering with all pairs previously assigned with the same color. The on-line cost is the number of colors used.

We show for all these on-line optical routing problems an $\Omega(n^{\epsilon})$ lower bound on the competitive ratio of randomized algorithms.

The results for reconfigurable networks are directly derived from the analogous results for on-line edge-disjoint paths and on-line path coloring.

The results for switchless networks are derived by showing that an independent set is a subproblem of routing in switchless optical networks (benefit version), and that graph coloring is a subproblem of routing in optical switchless networks (coloring version).

1.4. Structure of the paper. In section 2 we give formal definitions of competitive analysis and explain the use of Yao's lemma [Y77] for lower bounds for on-line randomized algorithms.

Section 3 deals with the on-line induced subgraph problem. In section 3.1 we present the lower bound for randomized algorithms for the on-line induced subgraph problem. Finally, section 3.2 gives upper bounds for the on-line induced subgraph problem. We then show in section 4 how the lower bounds for on-line independent sets can be transformed into a lower bound for on-line edge-disjoint paths (and thus for call control).

Section 5 deals with the on-line graph coloring problem. The lower bound for the problem is presented in section 5.1, and upper bounds are presented in section 5.2. The lower bound for on-line graph coloring is then transformed into a lower bound for on-line path coloring in section 6.

Section 7 deals with optical routing. The section contains all the lower bound results that follow from the results in the previous sections. Finally, in section 8 we present the lower bounds for the on-line edge-disjoint fixed paths problem.

2. Competitive analysis of randomized on-line algorithms. We consider problems that consist of a set of request sequences and a set of possible corresponding answers. Given a specific request sequence, an algorithm for the problem must provide an answer in response to each of the requests in the sequence. If the algorithm is on-line, it must base its answer for a particular request upon only the requests presented prior to the current request and the current request itself. A randomized algorithm may use coin tosses to decide upon its answers. An objective function is defined that maps request/answer sequences into the nonnegative reals.

Let A be an algorithm, and let σ be a request sequence. The value of the objective function of algorithm A associated with σ is denoted by $\mathrm{A}(\sigma)$.

Problems are divided into *cost* problems and *benefit* problems.

Cost problems. In the case of cost problems, $\mathrm{A}(\sigma)$ denotes the *cost* of algorithm A, and the goal of the algorithm is to minimize its cost. If A is randomized, the goal is to minimize its expected cost over its coin tosses, $\mathrm{E}[\mathrm{A}(\sigma)]$.

Let ON be an on-line algorithm (possibly randomized) and OPT be an optimal algorithm for some problem.

We say that ON is ρ-competitive [ST85] for a cost problem (against an oblivious

adversary (cf. [BBKTW90])) if there exists a constant a so that for every sequence σ,

$$\mathrm{E}[\mathrm{ON}(\sigma)] \leq \rho \cdot \mathrm{OPT}(\sigma) + a.$$

The value ρ is called the competitive ratio of the algorithm.

Benefit problems. In the case of benefit problems, $\mathrm{A}(\sigma)$ denotes the *benefit* of algorithm A, and the goal of the algorithm is to maximize its benefit. If A is randomized, the goal is to maximize its expected benefit over its coin tosses, $\mathrm{E}[\mathrm{A}(\sigma)]$.

Let ON be an on-line algorithm (possibly randomized) and OPT be an optimal algorithm for some problem.

We say that ON is ρ-competitive for a benefit problem (against an oblivious adversary) if there exists a constant a so that for every sequence σ,

$$\mathrm{OPT}(\sigma) \leq \rho \cdot (\mathrm{E}[\mathrm{ON}(\sigma)] + a).$$

The value ρ is called the competitive ratio of the algorithm.

2.1. Lower bounds using the minimax principle. Lower bounds for on-line randomized algorithms can be proved using Yao's lemma based on the *von Neumann minimax principle* (see [Y77]) applied to on-line algorithms. We need consider only the case where the sequences are finite and the number of deterministic algorithms is finite.

In [BE97] it is shown that in this case the following corollaries can be derived from Yao's lemma.

COROLLARY 1. *Given a cost problem, the following three statements are equivalent:*

1. *ρ is a lower bound on the competitive ratio of randomized on-line algorithms.*
2. *For every constant $a \geq 0$, there exists a probability distribution over request sequences σ such that $\mathrm{OPT}(\sigma) \geq a$ and for any deterministic on-line algorithm*
$$\mathrm{E}\left[\frac{\mathrm{ON}(\sigma)}{\mathrm{OPT}(\sigma)}\right] \geq \rho.$$
3. *For every constant $a \geq 0$, there exists a probability distribution over request sequences σ such that $\mathrm{OPT}(\sigma) \geq a$ and for any deterministic on-line algorithm*
$$\frac{\mathrm{E}[\mathrm{ON}(\sigma)]}{\mathrm{E}[\mathrm{OPT}(\sigma)]} \geq \rho.$$

COROLLARY 2. *Given a benefit problem, the following three statements are equivalent:*

1. *ρ is a lower bound on the competitive ratio of randomized on-line algorithms.*
2. *For every constant $a \geq 0$, there exists a probability distribution over request sequences σ such that $\mathrm{OPT}(\sigma) \geq a$ and for any deterministic on-line algorithm*
$$\mathrm{E}\left[\frac{\mathrm{ON}(\sigma)}{\mathrm{OPT}(\sigma)}\right] \leq \frac{1}{\rho}.$$
3. *For every constant $a \geq 0$, there exists a probability distribution over request sequences σ such that $\mathrm{OPT}(\sigma) \geq a$ and for any deterministic on-line algorithm*
$$\frac{\mathrm{E}[\mathrm{OPT}(\sigma)]}{\mathrm{E}[\mathrm{ON}(\sigma)]} \geq \rho.$$

Note that the expectation in the above corollaries is over the probability distribution on the input sequences.

In the remainder of the paper we sometime slightly abuse notation by using "input sequence from the probability distribution D" in place of the more accurate "input sequence having nonzero probability according to probability distribution D."

In this paper we give lower bounds based on statement 3 in the above corollaries.

3. The on-line induced subgraph problem. In this section we give lower and upper bounds on the competitive ratio of randomized algorithms for the on-line induced subgraph problem.

We start with some properties concerning induced subgraph problems, which will be used later in our proofs.

A graph property π is *hereditary* if, when satisfied for a given graph G, it is also satisfied for any vertex induced subgraph of G. Property π is *nontrivial* if it is satisfied for infinitely many graphs and not satisfied for infinitely many graphs.

Given a graph $G = (V, E)$ and a nontrivial hereditary property π, the induced subgraph problem consists of finding the maximum cardinality induced subgraph satisfying the property π. Any graph H that does not satisfy a hereditary property π is called a *forbidden graph* for π. Observe that any graph G that has H as an induced subgraph does not satisfy π.

For any hereditary property there exists a set of "minimal" forbidden graphs, i.e., forbidden graphs for which every induced proper subgraph satisfies the property. For instance, a K_5 (a clique of 5 vertices) is a minimal forbidden graph for planarity, and a K_2 is a minimal forbidden graph for an independent set.

Given a nontrivial hereditary property π, there is a complementary property π^c that is satisfied for a graph G if and only if π is satisfied for the complemented graph G^c (the graph that contains an edge if and only if G does not contain that edge). The complementary property is also a nontrivial hereditary property. Ramsey's theorem implies that any nontrivial hereditary property is satisfied by all independent sets or by all cliques [LY93a]. Thus, as observed in [LY93a], it is possible to restrict our attention to hereditary nontrivial graph properties that are satisfied for all independent sets.

The graph instances for our lower bounds are based on an application of the following variant of Lemma 9 in [LY93a].

LEMMA 3 (Lund and Yannakakis). *Given any graph H, there exist constants M and α such that for all $n > M$ there exists a graph G on n vertices such that any induced subgraph of G on at least $l = \alpha \log n$ vertices contains H as an induced subgraph.*

Observe that if the graph H of the lemma is a K_c, then G can be a K_n and $l = c$.

Let the graph H of Lemma 3 be a forbidden graph for a nontrivial, hereditary, property π. We base our lower bound constructions on a graph $G = (V, E)$ of n vertices, whose existence is guaranteed by Lemma 3 applied with the specific graph H.

On the basis of this graph G, we build the instance $\tilde{G}$ for the lower bound for the on-line induced subgraph problem.

Recall that in the on-line version of the induced subgraph problem we assume that the graph $\tilde{G}$, from which the adversary draws vertices, is known to the on-line algorithm.

The graph $\tilde{G}$ is built recursively. The vertex set of $\tilde{G}$ is V. In fact, $\tilde{G}$ is simply the graph G with some edges removed. We remove edges so that the "clever" adversary can choose vertices carefully while avoiding including the forbidden graph H as an

induced subgraph of the solution, whereas the on-line algorithm cannot do so well.

We next need to define sequences of requests to the vertices in $\tilde{G}$. Recall that at every step a new vertex is presented to the on-line algorithm, which must decide whether to choose to include it or choose to reject it.

The lower bounds are based on Yao's lemma (see section 2). We recursively define a probability distribution on request sequences in $\tilde{G}$ so that the ratio between the expected benefit of an optimal off-line algorithm and the expected benefit of any deterministic on-line algorithm is $\Omega(n^\epsilon)$.

Section 3.1 describes the graph construction, the definition of the probability distribution, and the analysis of the lower bound on the competitive ratio. In section 3.2 we give upper bounds for the on-line induced subgraph problem.

3.1. Lower bound for randomized algorithms for on-line induced subgraph. Let π be a hereditary nontrivial graph property. Let $G = (V, E)$ be a graph of n vertices such that every induced subgraph of at least $l = \alpha \log n$ vertices is a forbidden graph for property π. As mentioned before, the existence of such a graph for a large enough value of n is given by Lemma 3. For clarity of exposition, we assume that n is a power of 4.

We consider a sequence of $\log_4 n$ partitions of the vertices of $V = \{v_1, v_2, \ldots, v_n\}$. The level 0 partition,

$$P_0 = \{V_0^1 = \{v_1\}, V_0^2 = \{v_2\}, \ldots, V_0^n = \{v_n\}\},$$

consists of n disjoint singleton sets of vertices from V. The level i partition, $0 < i \leq \log_4 n$,

$$P_i = \{V_i^1, V_i^2, \ldots, V_i^{n/4^i}\},$$

is a partition of the vertices of V, where

$$V_i^j = \cup_{k=1}^4 V_{i-1}^{4(j-1)+k}.$$

As a shorthand notation we may use the definition $V_{i-1}^{4(j-1)+k} = V_{i-1,j}^k$, for $1 \leq i \leq \log_4 n$, $1 \leq j \leq n/4^i$, $1 \leq k \leq 4$. Note that $V_{\log_4 n}^1 = V$. Note too that the set $V_i^j = \{v_{(j-1)4^i+1}, \ldots, v_{j \cdot 4^i}\}$.

We now describe the construction of $\tilde{G}$ recursively. The vertex set of $\tilde{G}$ is V. The graph $\tilde{G}$ can be viewed as the graph G with some edges removed.

For all $0 \leq i \leq \log_4 n$ and all $V_i^j \in P_i$ we define graphs $\tilde{G}_i(V_i^j)$ with vertex set equal to V_i^j. The final graph is $\tilde{G} = \tilde{G}_{\log_4 n}(V)$. The graph $\tilde{G}_0(V_0^j)$ is simply the single vertex v_j and has no edges.

The graph $\tilde{G}_i(V_i^j)$, $i > 0$, $j = 1, \ldots, n/4^i$, is constructed from four graphs $\tilde{G}_{i-1,j}^k = \tilde{G}_{i-1}(V_{i-1,j}^k)$, for $k = 1, 2, 3, 4$.

The set of edges of $\tilde{G}_i(V_i^j)$ (see Figure 1) is recursively defined as the union of the set of edges in $\tilde{G}_{i-1,j}^1, \ldots, \tilde{G}_{i-1,j}^4$ and three new sets of edges:

- E^{12}—edges (u, w), $u \in V_{i-1,j}^1$, $w \in V_{i-1,j}^2$, if and only if $(u, w) \in E$;
- E^{13}—edges (u, w), $u \in V_{i-1,j}^1$, $w \in V_{i-1,j}^3$, if and only if $(u, w) \in E$;
- E^{24}—edges (u, w), $u \in V_{i-1,j}^2$, $w \in V_{i-1,j}^4$, if and only if $(u, w) \in E$.

Graph $\tilde{G}_i(V_i^j)$ has $n(i) = 4^i$ vertices.

Note that in general it is not true that the graphs $\tilde{G}_i(V_i^j)$, $V_i^j \in P_i$, are isomorphic. This will be true if $G = K_n$.

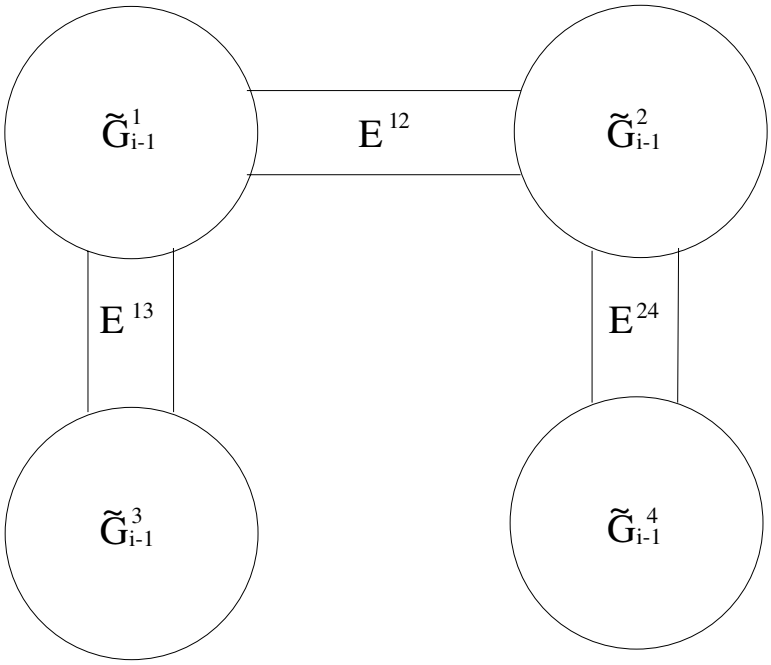


FIG. 1. *The construction of graph* $\tilde{G}_i(V_i^j)$.

We say that two graphs of level $i-1$ connected by edges are *adjacent*. (Observe that in the case that $G = K_n$ a vertex of a graph is connected to all the vertices of an adjacent graph).

We say that $H \subseteq V$ induces a subgraph of G, with respect to $\tilde{G}_i(V_i^j)$, of size m if the following condition holds: There exist m different vertices $\{u_1, u_2, \ldots, u_m\} \subseteq H \cap V_i^j$ such that, for all $1 \leq l, k \leq m$, if $(u_l, u_k) \in E$, then (u_l, u_k) is an edge of $\tilde{G}_i(V_i^j)$.

For a set $H \subseteq V$ and for any $0 \leq i \leq \log_4 n$, $1 \leq j \leq n/4^i$, we define

$$s(\tilde{G}_i(V_i^j), H) = \max\{m | H \text{ induces a subgraph of } G, \text{ with respect to } \tilde{G}_i(V_i^j), \text{ of size } m\};$$

i.e., $s(\tilde{G}_i(V_i^j), H)$ is the size of the largest vertex induced subgraph of $G(V_i^j)$ that is also a vertex induced subgraph of $\tilde{G}_i$ and whose vertices all belong to H. Note that H may contain other vertices not in V_i^j.

Assume that H induces a subgraph I_1 of G with respect to $\tilde{G}^1_{i-1,j}$, and that H also induces a subgraph I_2 of G with respect to $\tilde{G}^2_{i-1,j}$; then H induces a subgraph of G of size $|I_1| + |I_2|$ with respect to $\tilde{G}_i(V_i^j)$. This follows since $H \cap V_i^j \supseteq H \cap V^1_{i-1,j} \cup H \cap V^2_{i-1,j}$, and $\tilde{G}_i^j$ contains all the edges of $\tilde{G}^1_{i-1,j}$ and $\tilde{G}^2_{i-1,j}$ and all the edges between $V^1_{i-1,j}$ and $V^2_{i-1,j}$ that belong to E. A similar argument can be made for the subgraphs induced by H with respect to the pair of graphs $\tilde{G}^1_{i-1,j}$, $\tilde{G}^3_{i-1,j}$, respectively, and likewise for the pair $\tilde{G}^2_{i-1,j}$, $\tilde{G}^4_{i-1,j}$.

To summarize the argument above, the purpose of the edges E^{12}, E^{13}, and E^{24} is to ensure that for all $H \subseteq V$, $1 \le i \le \log_4 n$, and $1 \le j \le n/4^i$

$$\begin{aligned} s(\tilde{G}_i(V_i^j), H) \ge \max\{ & s(\tilde{G}^1_{i-1,j}, H) + s(\tilde{G}^2_{i-1,j}, H), \\ & s(\tilde{G}^3_{i-1,j}, H) + s(\tilde{G}^1_{i-1,j}, H), \\ & s(\tilde{G}^2_{i-1,j}, H) + s(\tilde{G}^4_{i-1,j}, H)\}. \end{aligned}$$

We will now define the probability distribution on the request sequences of vertices of $\tilde{G}$. We will prove for any on-line algorithm ON a lower bound on the ratio ρ between the expected optimal off-line benefit $\mathrm{E}[|OPT(\sigma)|]$ and the expected on-line benefit $\mathrm{E}[|ON(\sigma)|]$. We remark that for our distribution over sequences, the off-line benefit is independent of the choice of sequence.

The probability distribution over the sequences will be a uniform distribution over a set of sequences of vertices. The set of sequences is defined recursively as follows. The set of sequences for the graph $\tilde{G}_0(u)$, $u \in V$, consists of the single sequence $\langle u \rangle$. Let $S(V_i^j)$ denote the set of sequences over V_i^j, $0 \le i \le \log_4 n$, $V_i^j \in P_i$. Also, let $\|$ denote the concatenation operator on sequences.

Given $S(V^1_{i-1,j})$, $S(V^2_{i-1,j})$, $S(V^3_{i-1,j})$, and $S(V^4_{i-1,j})$, $1 \le i \le \log_4 n$, we define the set $S(V_i^j)$ as the result of the following process:

$$S_{12} = \{x \| y \mid x \in S(V^1_{i-1,j}), y \in S(V^2_{i-1,j})\},$$

$$S_{123} = \{x \| y | x \in S_{12}, y \in S(V^3_{i-1,j})\}, \tag{1}$$

$$S_{124} = \{x \| y | x \in S_{12}, y \in S(V^4_{i-1,j})\}, \tag{2}$$

$$S(V_i^j) = S_{123} \cup S_{124}.$$

Claim 4. *In any outcome of the above process (to generate $S(V_i^j)$), $|S_{123}| = |S_{124}|$.*

Proof. From step (1) above we have that $|S_{123}| = |S_{12}| \cdot |S(V^3_{i-1,j})|$, and from step (2) $|S_{124}| = |S_{12}| \cdot |S(V^4_{i-1,j})|$. For any two sets $V^k_{i-1,j}$, $V^\ell_{i-1,j}$, $1 \le k < \ell \le 4$, we can find a mapping $f(u) \mapsto w$, $u \in V^k_{i-1,j}$, $w \in V^\ell_{i-1,j}$, such that both sets and their recursive partitions into subsets are isomorphic under f. Specifically, $f(v_t) = v_{t+(\ell-k)4^{i-1}}$ will be such a mapping. Thus, $|S(V^3_{i-1,j})| = |S(V^4_{i-1,j})|$, and it now follows that $|S_{123}| = |S_{124}|$. ∎

Define the probability distribution $U(S)$, where S is a set of sequences to be the uniform distribution over S. Let $T \subset V$; the notation $\sigma|T$ denotes the sequence derived from σ by removing all elements in $V - T$. Let $T' \subseteq T \subseteq V$, where $S(T)$ and $S(T')$ are defined (i.e., $T = V_i^j$ and $T' = V_{i'}^{j'}$ for some i, j, i', j'). Define the restriction of $S(T)$ to T', i.e., $S(T|T') = \{\sigma|T' | \sigma \in S(T)\}$. $U(S(T|T'))$ is a distribution over $S(T')$.

Claim 5. *$U(S(V_i^j|V^k_{i-1,j})) = U(S(V^k_{i-1,j}))$ for $k = 1, \ldots, 4$.*

Proof. Every sequence in $S(V^k_{i-1,j})$ appears as a subsequence of the same number of sequences in $S(V_i^j)$. ∎

Let A be any algorithm (on-line or off-line), and let σ be a sequence of vertices from V. Let $A(\sigma)$ denote the set of vertices chosen by the algorithm. ($A(\sigma)$ induces a subgraph of $\tilde{G}(V)$ satisfying property π.)

Lemma 6. *For any $0 \le i \le \log_4 n$, $1 \le j \le n/4^i$, and any sequence $\sigma \in S(V_i^j)$, $|\mathrm{OPT}(\sigma)| \ge 2^i$.*

Proof. Given any sequence $\sigma \in S(V_i^j)$, we will prove by induction on i that the set of vertices in σ contains a subset of vertices $I(\sigma) \subseteq V_i^j$, $|I(\sigma)| = 2^i$, where $I(\sigma)$ is an independent set in $\tilde{G}_i(V_i^j)$. (And thus $I(\sigma)$ obeys property π—OPT *can* choose all vertices in $I(\sigma)$ giving $|\mathrm{OPT}(\sigma)| \geq 2^i$.)

The claim is clearly true for $i = 0$. Assume that the inductive hypothesis holds for $i-1$: Let τ be any sequence in $S(V_{i-1}^\ell)$, $1 \leq \ell \leq n/4^{i-1}$. The set of vertices in τ contain a subset of vertices $I(\tau) \subseteq V_{i-1}^\ell$, $|I(\tau)| \geq 2^{i-1}$, where $I(\tau)$ is an independent set in $\tilde{G}_{i-1}(V_{i-1}^\ell)$.

Without loss of generality, assume that $\sigma \in S_{123}$ from the construction. By construction (of S_{123}) there are disjoint subsequences of σ, $\tau_2 \in S(V_{i-1,j}^2)$ and $\tau_3 \in S(V_{i-1,j}^3)$. By the inductive hypothesis, there exist $I(\tau_2) \subseteq V_{i-1,j}^2$, $I(\tau_3) \subseteq V_{i-1,j}^3$, $|I(\tau_2)| = 2^{i-1}$, and $|I(\tau_3)| = 2^{i-1}$, where $I(\tau_2)$ is an independent set in $\tilde{G}_{i-1}(V_{i-1,j}^2)$, and $I(\tau_3)$ is an independent set in $\tilde{G}_{i-1}(V_{i-1,j}^3)$.

Letting $I(\sigma) = I(\tau_2) \cup I(\tau_3)$, we claim that $I(\sigma)$ is an independent set in $\tilde{G}_i(V_i^j)$; this follows because there are no edges in $\tilde{G}_i(V_i^j)$ between the sets $V_{i-1,j}^2$ and $V_{i-1,j}^3$.

It is also true that $|I(\sigma)| = |I(\tau_2)| + |I(\tau_3)| = 2^i$.

If $\sigma \in S_{124}$, then the same argument works, with $V_{i-1,j}^1$ taking the place of $V_{i-1,j}^2$ and $V_{i-1,j}^4$ taking the place of $V_{i-1,j}^3$. □

Let $E_{x \in_R D}[r(x)]$ denote the expectation of the random variable $r(x)$, where x is chosen from the distribution D.

Claim 7. *Given an online algorithm ON, a subset $T \subseteq V$, and sequences τ', τ'' of vertices from $V - T$, then for any sequence τ of vertices from T there is an online algorithm $ON^{\tau'}$ such that*

$$ON(\tau' \| \tau \| \tau'') \cap T = ON^{\tau'}(\tau).$$

Proof. $ON^{\tau'}$ emulates the execution of ON as follows. For any τ, $ON^{\tau'}$ accepts τ_ℓ if and only if ON would accept τ_ℓ after seeing the input sequence $\tau' \| \langle \tau_1, \tau_2, \ldots, \tau_\ell \rangle$. Note that any vertices accepted in τ' or τ'' are not in T, and therefore

$$ON(\tau' \| \tau \| \tau'') \cap T = ON^{\tau'}(\tau). \qquad \square$$

Example. Let ON be an online algorithm, and assume that $\sigma \in S_{123}$ from the construction. By construction (of S_{123}), σ is the concatenation of three disjoint subsequences: $\tau_1 \in S(V_{i-1,j}^1)$, $\tau_2 \in S(V_{i-1,j}^2)$, and $\tau_3 \in S(V_{i-1,j}^3)$. We have $ON(\tau_1) = ON(\sigma) \cap V_{i-1,j}^1$, $ON^{\tau_1}(\tau_2) = ON(\sigma) \cap V_{i-1,j}^2$, and $ON^{\tau_1 \| \tau_2}(\tau_3) = ON(\sigma) \cap V_{i-1,j}^3$.

Lemma 8. *For any deterministic online algorithm ON and a sequence of vertices from V_i^j, $0 \leq i \leq \log_4 n$, $1 \leq j \leq n/4^i$, the following hold:*

$$(3)\quad E_{\sigma \in_R U(S(V_i^j))}[|ON(\sigma) \cap V_{i-1,j}^1|] = E_{\tau_1 \in_R U(S(V_{i-1,j}^1))}[|ON(\tau_1)|],$$

$$(4)\quad E_{\sigma \in_R U(S(V_i^j))}[|ON(\sigma) \cap V_{i-1,j}^2|] = E_{\tau_1 \in_R U(S(V_{i-1,j}^1))} E_{\tau_2 \in_R U(S(V_{i-1,j}^2))}[|ON^{\tau_1}(\tau_2)|],$$

$$(5)\quad E_{\sigma \in_R U(S(V_i^j))}[|ON(\sigma) \cap V_{i-1,j}^3|] = \frac{1}{2} E_{\tau_1 \in_R U(S(V_{i-1,j}^1))} E_{\tau_2 \in_R U(S(V_{i-1,j}^2))} E_{\tau_3 \in_R U(S(V_{i-1,j}^3))}[|ON^{\tau_1 \| \tau_2}(\tau_3)|],$$

$$(6)\quad E_{\sigma \in_R U(S(V_i^j))}[|ON(\sigma) \cap V_{i-1,j}^4|] = \frac{1}{2} E_{\tau_1 \in_R U(S(V_{i-1,j}^1))} E_{\tau_2 \in_R U(S(V_{i-1,j}^2))} E_{\tau_4 \in_R U(S(V_{i-1,j}^4))}[|ON^{\tau_1 \| \tau_2}(\tau_4)|].$$

Proof. Let $\sigma = \tau_1\|\tau_2\|\tau'$, where $\tau_1 \in S(V^1_{i-1,j})$, $\tau_2 \in S(V^2_{i-1,j})$, and $\tau' \in S(V^3_{i-1,j}) \cup S(V^4_{i-1,j})$.

To prove (3) above we have

$$E_{\sigma\in_R U(S(V_i^j))}[|ON(\sigma)\cap V^1_{i-1,j}|] \tag{7}$$
$$= \frac{\sum_{\sigma\in S(V_i^j)}|ON(\sigma)\cap V^1_{i-1,j}|}{|S(V_i^j)|}$$
$$= \frac{\sum_{\tau_1\|\tau_2\|\tau'\in S(V_i^j)}|ON(\tau_1\|\tau_2\|\tau')\cap V^1_{i-1,j}|}{|S(V_i^j)|} \tag{8}$$
$$= \frac{\sum_{\tau_1\|\tau_2\|\tau'\in S(V_i^j)}|ON(\tau_1)|}{|S(V_i^j)|} \tag{9}$$
$$= \frac{|S(V^2_{i-1,j})|\cdot(|S(V^3_{i-1,j})|+|S(V^4_{i-1,j})|)\cdot\sum_{\tau_1\in S(V^1_{i-1,j})}|ON(\tau_1)|}{|S(V^1_{i-1,j})|\cdot|S(V^2_{i-1,j})|\cdot(|S(V^3_{i-1,j})|+|S(V^4_{i-1,j})|)} \tag{10}$$
$$= E_{\tau_1\in_R U((V^1_{i-1,j}))}|ON(\tau_1)|. \tag{11}$$

Equation (7) follows from the definition of the expectation. In (8) we substitute $\tau_1\|\tau_2\|\tau'$ for σ. As $\tau_2\cap V^1_{i-1,j} = \tau'\cap V^1_{i-1,j} = \emptyset$, we get that $ON(\tau_1\|\tau_2\|\tau')\cap V^1_{i-1,j} = ON(\tau_1)$, giving (9). Using that

$$|S(V_i^j)| = |S(V^1_{i-1,j})|\cdot|S(V^2_{i-1,j})|\cdot(|S(V^3_{i-1,j})|+|S(V^4_{i-1,j})|),$$

we derive (10) and (11).

To prove (4) above we have

$$E_{\sigma\in_R U(S(V_i^j))}[|ON(\sigma)\cap V^2_{i-1,j}|] \tag{12}$$
$$= \frac{\sum_{\sigma\in S(V_i^j)}|ON(\sigma)\cap V^2_{i-1,j}|}{|S(V_i^j)|}$$
$$= \frac{\sum_{\tau_1\|\tau_2\|\tau'\in S(V_i^j)}|ON(\tau_1\|\tau_2\|\tau')\cap V^2_{i-1,j}|}{|S(V_i^j)|} \tag{13}$$
$$= \frac{\sum_{\tau_1\|\tau_2\|\tau'\in S(V_i^j)}|ON^{\tau_1}(\tau_2)|}{|S(V_i^j)|} \tag{14}$$
$$= \frac{(|S(V^3_{i-1,j})|+|S(V^4_{i-1,j})|)\cdot\sum_{\tau_1\in S(V^1_{i-1,j}),\tau_2\in S(V^2_{i-1,j})}|ON^{\tau_1}(\tau_2)|}{|S(V^1_{i-1,j})|\cdot|S(V^2_{i-1,j})|\cdot(|S(V^3_{i-1,j})|+|S(V^4_{i-1,j})|)} \tag{15}$$
$$= E_{\tau_1\in_R U(S(V^1_{i-1,j}))}E_{\tau_2\in_R U(S(V^2_{i-1,j}))}|ON^{\tau_1}(\tau_2)|.$$

We derive (12) by definition of the expectation, (13) substitutes $\tau_1\|\tau_2\|\tau'$ for σ, and using Claim 7, we get (14). Next, we note that

$$\sum_{\tau_1\|\tau_2\|\tau'\in S(V_i^j)} f(\tau_1,\tau_2) = (|S(V^3_{i-1,j})|+|S(V^4_{i-1,j})|)\sum_{\tau_1\in V^1_{i-1,j},\tau_2\in V^2_{i-1,j}} f(\tau_1,\tau_2).$$

Our final simplification (15) follows from the definition of the expectation.

Next we prove (5). Equation (6) is analogous and can be derived similarly. We have

$$E_{\sigma\in_R U(S(V_i^j))}[|ON(\sigma)\cap V_{i-1,j}^3|] \tag{16}$$

$$= \frac{\sum_{\sigma\in S(V_i^j)}|ON(\sigma)\cap V_{i-1,j}^3|}{|S(V_i^j)|}$$

$$= \frac{\sum_{\tau_1\|\tau_2\|\tau'\in S(V_i^j)}|ON(\tau_1\|\tau_2\|\tau')\cap V_{i-1,j}^3|}{|S(V_i^j)|} \tag{17}$$

$$= \frac{\sum_{\tau_1\in S(V_{i-1,j}^1),\tau_2\in S(V_{i-1,j}^2),\tau_3\in S(V_{i-1,j}^3)}|ON^{\tau_1\|\tau_2}(\tau_3)|}{|S(V_{i-1,j}^1)|\cdot|S(V_{i-1,j}^2)|\cdot(|S(V_{i-1,j}^3)|+|S(V_{i-1,j}^4)|)} \tag{18}$$

$$= \frac{1}{2}\frac{\sum_{\tau_1\in S(V_{i-1,j}^1),\tau_2\in S(V_{i-1,j}^2),\tau_3\in S(V_{i-1,j}^3)}|ON^{\tau_1\|\tau_2}(\tau_3)|}{|S(V_{i-1,j}^1)|\cdot|S(V_{i-1,j}^2)|\cdot|S(V_{i-1,j}^3)|} \tag{19}$$

$$= \frac{1}{2}E_{\tau_1\in_R U(S(V_{i-1,j}^1))}E_{\tau_2\in_R U(S(V_{i-1,j}^2))}E_{\tau_3\in_R U(S(V_{i-1,j}^3))}[|ON^{\tau_1\|\tau_2}(\tau_3)|]. \tag{20}$$

We derive (16) by definition of the expectation, (17) substitutes $\tau_1\|\tau_2\|\tau'$ for σ, by noting that $ON(\tau_1\|\tau_2\|\tau')\cap V_{i-1,j}^3$ is empty when $\tau'\in S(V_{i-1,j}^4)$, and using Claim 7, we get (18). Equation (19) follows from Claim 4. Our final simplification (20) follows from the definition of the expectation. □

Given an on-line algorithm ON, define $s_i^j(\sigma, ON)$ to be $s(\tilde{G}_i(V_i^j), ON(\sigma))$, the size of the largest subgraph of G induced by $ON(\sigma)$ with respect to $\tilde{G}_i(V_i^j)$. Likewise define $s_{i-1,j}^k(\sigma, ON)$ to be $s(\tilde{G}_{i-1}(V_{i-1,j}^k), ON(\sigma))$, for $k = 1, 2, 3, 4$.

CLAIM 9. *For any deterministic on-line algorithm ON defined on sequences of vertices from V_i^j, $0 \le i \le \log_4 n$, $1 \le j \le n/4^i$, and for any $\tau_1 \in S(V_{i-1,j}^1), \tau_2 \in S(V_{i-1,j}^2), \tau_3 \in S(V_{i-1,j}^3), \tau_4 \in S(V_{i-1,j}^4)$ the following hold:*

$$s_{i-1,j}^2(\tau_2, ON^{\tau_1}) = s_{i-1,j}^2(\tau_1\|\tau_2, ON), \tag{21}$$

$$s_{i-1,j}^3(\tau_3, ON^{\tau_1\|\tau_2}) = s_{i-1,j}^3(\tau_1\|\tau_2\|\tau_3, ON), \tag{22}$$

$$s_{i-1,j}^4(\tau_4, ON^{\tau_1\|\tau_2}) = s_{i-1,j}^4(\tau_1\|\tau_2\|\tau_4, ON). \tag{23}$$

Proof. The proof follows from the definitions of ON^τ and s_i^j. □

LEMMA 10. *For any $0 \le i \le \log_4 n$, $1 \le j \le n/4^i$, the expected benefit of any deterministic on-line algorithm over the distribution $U(S(V_i^j))$ satisfies*

$$E_{\sigma\in_R U(S(V_i^j))}[|ON(\sigma)|] \le \left(\frac{3}{2}\right)^i E_{\sigma\in_R U(S(V_i^j))}[s_i^j(\sigma, ON)].$$

Proof. We prove the claim by induction on i. Consider an algorithm ON. The claim holds for $i = 0$, and any $1 \le j \le n$ (V_0^j is a single vertex). We assume by induction that the claim holds for all probability distributions $U(S(V_{i-1}^j))$, $1 \le j \le n/4^{i-1}$, which means in particular that it holds for the probability distributions $U(S(V_{i-1,j}^k))$, $1 \le j \le n/4^i$, $k = 1, 2, 3, 4$.

We prove the claim as follows:

$$
\begin{aligned}
&E_{\sigma\in_R U(S(V_i^j))}[|ON(\sigma)|]\\
&\quad= E_{\sigma\in_R U(S(V_i^j))}[|ON(\sigma)\cap V_{i-1,j}^1|]\\
&\qquad+E_{\sigma\in_R U(S(V_i^j))}[|ON(\sigma)\cap V_{i-1,j}^2|]\\
&\qquad+E_{\sigma\in_R U(S(V_i^j))}[|ON(\sigma)\cap V_{i-1,j}^3|]\\
&\qquad+E_{\sigma\in_R U(S(V_i^j))}[|ON(\sigma)\cap V_{i-1,j}^4|]\\
&\quad= E_{\tau_1\in_R U(S(V_{i-1,j}^1))}[|ON(\tau_1)|]\\
&\qquad+E_{\tau_1\in_R U(S(V_{i-1,j}^1))}E_{\tau_2\in_R U(S(V_{i-1,j}^2))}[|ON^{\tau_1}(\tau_2)|]\\
&\qquad+\frac{1}{2}E_{\tau_1\in_R U(S(V_{i-1,j}^1))}E_{\tau_2\in_R U(S(V_{i-1,j}^2))}E_{\tau_3\in_R U(S(V_{i-1,j}^3))}[|ON^{\tau_1\|\tau_2}(\tau_3)|]\\
&\qquad+\frac{1}{2}E_{\tau_1\in_R U(S(V_{i-1,j}^1))}E_{\tau_2\in_R U(S(V_{i-1,j}^2))}E_{\tau_4\in_R U(S(V_{i-1,j}^4))}[|ON^{\tau_1\|\tau_2}(\tau_4)|].
\end{aligned}
\tag{24}
$$

By the inductive hypothesis we get that (24) is at most

$$
\left(\frac{3}{2}\right)^{i-1}\left\{\begin{array}{l}
E_{\tau_1\in_R U(S(V_{i-1,j}^1))}[s_{i-1,j}^1(\tau_1,ON)]\\
\quad+E_{\tau_1\in_R U(S(V_{i-1,j}^1))}E_{\tau_2\in_R U(S(V_{i-1,j}^2))}[s_{i-1,j}^2(\tau_2,ON^{\tau_1})]\\
\quad+\frac{1}{2}E_{\tau_1\in_R U(S(V_{i-1,j}^1))}E_{\tau_2\in_R U(S(V_{i-1,j}^2))}E_{\tau_3\in_R U(S(V_{i-1,j}^3))}[s_{i-1,j}^3(\tau_3,ON^{\tau_1\|\tau_2})]\\
\quad+\frac{1}{2}E_{\tau_1\in_R U(S(V_{i-1,j}^1))}E_{\tau_2\in_R U(S(V_{i-1,j}^2))}E_{\tau_4\in_R U(S(V_{i-1,j}^4))}[s_{i-1,j}^4(\tau_4,ON^{\tau_1\|\tau_2})]
\end{array}\right\}.
\tag{25}
$$

We have

$$
\begin{aligned}
&E_{\tau_1\in_R U(S(V_{i-1,j}^1))}[s_{i-1,j}^1(\tau_1,ON)]\\
&\quad=\frac{1}{2}E_{\tau_1\in_R U(S(V_{i-1,j}^1))}E_{\tau_2\in_R U(S(V_{i-1,j}^2))}E_{\tau_3\in_R U(S(V_{i-1,j}^3))}[s_{i-1,j}^1(\tau_1,ON)]\\
&\qquad+\frac{1}{2}E_{\tau_1\in_R U(S(V_{i-1,j}^1))}E_{\tau_2\in_R U(S(V_{i-1,j}^2))}[s_{i-1,j}^1(\tau_1,ON)],\\
&E_{\tau_1\in_R U(S(V_{i-1,j}^1))}E_{\tau_2\in_R U(S(V_{i-1,j}^2))}[s_{i-1,j}^2(\tau_2,ON^{\tau_1})]\\
&\quad=\frac{1}{2}E_{\tau_1\in_R U(S(V_{i-1,j}^1))}E_{\tau_2\in_R U(S(V_{i-1,j}^2))}[s_{i-1,j}^2(\tau_2,ON^{\tau_1})]\\
&\qquad+\frac{1}{2}E_{\tau_1\in_R U(S(V_{i-1,j}^1))}E_{\tau_2\in_R U(S(V_{i-1,j}^2))}E_{\tau_4\in_R U(S(V_{i-1,j}^4))}[s_{i-1,j}^2(\tau_2,ON^{\tau_1})].
\end{aligned}
$$

Thus, we get that (25) is equal to

$$
\left(\frac{3}{2}\right)^{i-1}\frac{1}{2}\left\{\begin{array}{l}
E_{\tau_1\in_R U(S(V_{i-1,j}^1))}E_{\tau_2\in_R U(S(V_{i-1,j}^2))}E_{\tau_3\in_R U(S(V_{i-1,j}^3))}\\
\qquad\cdot[s_{i-1,j}^1(\tau_1,ON)+s_{i-1,j}^3(\tau_3,ON^{\tau_1\|\tau_2})]\\
+E_{\tau_1\in_R U(S(V_{i-1,j}^1))}E_{\tau_2\in_R U(S(V_{i-1,j}^2))}E_{\tau_4\in_R U(S(V_{i-1,j}^4))}\\
\qquad\cdot[s_{i-1,j}^2(\tau_2,ON^{\tau_1})+s_{i-1,j}^4(\tau_4,ON^{\tau_1\|\tau_2})]\\
+E_{\tau_1\in_R U(S(V_{i-1,j}^1))}E_{\tau_2\in_R U(S(V_{i-1,j}^2))}[s_{i-1,j}^1(\tau_1,ON)+s_{i-1,j}^2(\tau_2,ON^{\tau_1})]
\end{array}\right\}.
\tag{26}
$$

Using Claim 9, we get that (26) is equal to

$$\left(\left(\frac{3}{2}\right)^{i-1}\cdot\frac{1}{2}\right)\cdot$$

$$\left\{\begin{array}{l} E_{\tau_1\in_R U(S(V^1_{i-1,j}))}E_{\tau_2\in_R U(S(V^2_{i-1,j}))}E_{\tau_3\in_R U(S(V^3_{i-1,j}))}[s^1_{i-1,j}(\tau_1\|\tau_2\|\tau_3,ON) \\ \qquad\qquad + s^3_{i-1,j}(\tau_1\|\tau_2\|\tau_3,ON)] \\ + E_{\tau_1\in_R U(S(V^1_{i-1,j}))}E_{\tau_2\in_R U(S(V^2_{i-1,j}))}E_{\tau_4\in_R U(S(V^4_{i-1,j}))}[s^2_{i-1,j}(\tau_1\|\tau_2\|\tau_4,ON) \\ \qquad\qquad + s^4_{i-1,j}(\tau_1\|\tau_2\|\tau_4,ON)] \\ + E_{\tau_1\in_R U(S(V^1_{i-1,j}))}E_{\tau_2\in_R U(S(V^2_{i-1,j}))}[s^1_{i-1,j}(\tau_1\|\tau_2,ON)+s^2_{i-1,j}(\tau_1\|\tau_2,ON)] \end{array}\right\}.$$

We now have

$$\begin{aligned} &E_{\sigma\in_R U(S(V^j_i))}[s^j_i(\sigma,ON)] \\ &\geq \frac{1}{2}E_{\tau_1\in_R U(S(V^1_{i-1,j}))}E_{\tau_2\in_R U(S(V^2_{i-1,j}))}E_{\tau_3\in_R U(S(V^3_{i-1,j}))} \\ &\qquad\cdot[s^1_{i-1,j}(\tau_1\|\tau_2\|\tau_3,ON)+s^3_{i-1,j}(\tau_1\|\tau_2\|\tau_3,ON)] \\ &\quad+\frac{1}{2}E_{\tau_1\in_R U(S(V^1_{i-1,j}))}E_{\tau_2\in_R U(S(V^2_{i-1,j}))}E_{\tau_4\in_R U(S(V^4_{i-1,j}))} \\ &\qquad\cdot[s^2_{i-1,j}(\tau_1\|\tau_2\|\tau_4,ON)+s^4_{i-1,j}(\tau_1\|\tau_2\|\tau_4,ON)] \end{aligned} \tag{27}$$

and

$$\begin{aligned} &E_{\sigma\in_R U(S(V^j_i))}[s^j_i(\sigma,ON)] \\ &\geq E_{\tau_1\in_R U(S(V^1_{i-1,j}))}E_{\tau_2\in_R U(S(V^2_{i-1,j}))}[s^1_{i-1,j}(\tau_1\|\tau_2,ON)+s^2_{i-1,j}(\tau_1\|\tau_2,ON)]. \end{aligned} \tag{28}$$

The lemma follows from (27) and (28). □

We then conclude with the following result.

THEOREM 11. *Any randomized algorithm for the on-line induced subgraph problem has competitive ratio* $\Omega(n^{1-\log_4 3-o(1)})$. *For a hereditary property having a* K_c *as forbidden graph the competitive ratio is* $\Omega(n^{1-\log_4 3})$.

Proof. By Lemmas 6 and 10 we have the following. For any $0 \leq i \leq \log_4 n$, $1 \leq j \leq n/4^i$, and any sequence $\sigma \in S(V^j_i)$, $|\mathrm{OPT}(\sigma)| \geq 2^i$. For any $0 \leq i \leq \log_4 n$, $1 \leq j \leq n/4^i$, the expected benefit of any deterministic on-line algorithm over the distribution $U(S(V^j_i))$ satisfies

$$E_{\sigma\in_R U(S(V^j_i))}[|ON(\sigma)|] \leq \left(\frac{3}{2}\right)^i E_{\sigma\in_R U(S(V^j_i))}[s^j_i(\sigma,ON)].$$

The ratio $\rho(i,j)$ between the expected optimal benefit and the expected on-line benefit for the probability distribution on the input sequences $U(S(V^j_i))$, $1 \leq i \leq \log_4 n$, $1 \leq j \leq n/4^i$, is

$$\begin{aligned} \rho(i,j) &= \frac{E_{\sigma\in_R U(S(V^j_i))}[|\mathrm{OPT}(\sigma)|]}{E_{\sigma\in_R U(S(V^j_i))}[|\mathrm{ON}(\sigma)|]} && \geq \frac{2^i}{\left(\frac{3}{2}\right)^i \mathrm{E}_{\sigma\in_R U(S(V^j_i))}[s^j_i(\sigma,ON)]} \\ &\geq \frac{4^i}{3^i E_{\sigma\in_R U(S(V^j_i))}[s^j_i(\sigma,ON)]} && = \frac{n(i)}{3^{\log_4 n(i)} E_{\sigma\in_R U(S(V^j_i))}[s^j_i(\sigma,ON)]} \\ &= \frac{n(i)}{n(i)^{\log_4 3} E_{\sigma\in_R U(S(V^j_i))}[s^j_i(\sigma,ON)]} && = \frac{n(i)^{1-\log_4 3}}{E_{\sigma\in_R U(S(V^j_i))}[s^j_i(\sigma,ON)]}. \end{aligned}$$

From Lemma 3 it follows that $E_{\sigma \in_R U(S(V_i^j))}[s_i^j(\sigma, ON)] < l = \alpha \log n$. Therefore, for $i = \log_4 n$, we obtain the $\Omega(n^{1-\log_4 3-o(1)})$ lower bound on the competitive ratio. If the forbidden graph is a K_c and G is a K_n, then $l = c$ and we get an $\Omega(n^{1-\log_4 3})$ lower bound. □

3.2. Upper bounds for the on-line induced subgraph problem. In this section we give upper bounds for the on-line induced subgraph problem. The upper bounds are based on algorithms that compute an approximate solution for the off-line maximum induced subgraph problem.

We present a randomized algorithm based on the "classify and randomly select" method of [ABFR94].

Let $G = (V, E)$ be the graph that we assume is known to the on-line algorithm. The algorithm requires a preprocessing phase in which a collection of k solutions $I_1, I_2, \ldots, I_k$ of the induced subgraph problem for the graph G are computed, as follows.

Let $G_1 = G$, and $G_{i+1} = G_i - I_i$ be the result of the operation that removes from G_i all vertices in I_i and all the incident edges.

I_i is the set of vertices forming the solution of the induced subgraph problem returned by applying the $\frac{n}{\alpha}$-approximation algorithm to the graph G_i. The process of computing solutions is continued until $|I_{k+1}| \leq \sqrt{\alpha}$. Let $\overline{I} = V - \{I_1 \cup I_2 \cup \cdots \cup I_k\}$ be the set of vertices of G_{k+1}.

Finally, the algorithm performs the following random choice.

Toss a fair coin:

1. If heads, choose at random with equal probability one of the sets $I_1, I_2, \ldots, I_k$, and then accept only presented vertices that belong to that set.
2. If tails, then greedily accept vertices from $\overline{I}$.

THEOREM 12. *Suppose that there is some $\frac{n}{\alpha}$-approximation algorithm for the induced subgraph problem and that there exists an $O(\frac{n}{\sqrt{\alpha}})$-competitive randomized algorithm for the on-line version of the problem. If the original algorithm is polynomial, so is the on-line algorithm.*

Proof. First observe that k is at most $\frac{n}{\sqrt{\alpha}}$. Next, observe that the optimal solution over the set $\overline{I}$ is at most $\frac{n}{\sqrt{\alpha}}$, as the solution produced by the $\frac{n}{\alpha}$-approximation algorithm applied to the graph induced by $\overline{I}$ has size not bigger than $\sqrt{\alpha}$.

Let $\mathrm{OPT}(I)$ be the size of the optimal solution in the subgraph of I induced by the vertices in I presented. Then

$$\mathrm{OPT} \leq \sum_{i=1}^{k} \mathrm{OPT}(I_i) + \mathrm{OPT}(\overline{I}).$$

The expected number of vertices accepted by the on-line algorithm is

$$\begin{aligned} \mathrm{ON} &\geq \frac{1}{2}\left(\frac{1}{k}\sum_{i=1}^{k} \mathrm{OPT}(I_i) + 1\right) \\ &\geq \frac{\sqrt{\alpha}}{2n}\left(\sum_{i=1}^{k} \mathrm{OPT}(I_i) + \mathrm{OPT}(\overline{I})\right) \geq \frac{\sqrt{\alpha}}{2n}\mathrm{OPT}. \end{aligned}$$ □

The current best known polynomial time algorithms have an approximation ratio of $O(\frac{n}{\log n})$ for the induced subgraph problem [Hal94], and $O(\frac{n}{(\log n)^2})$ for the indepen-

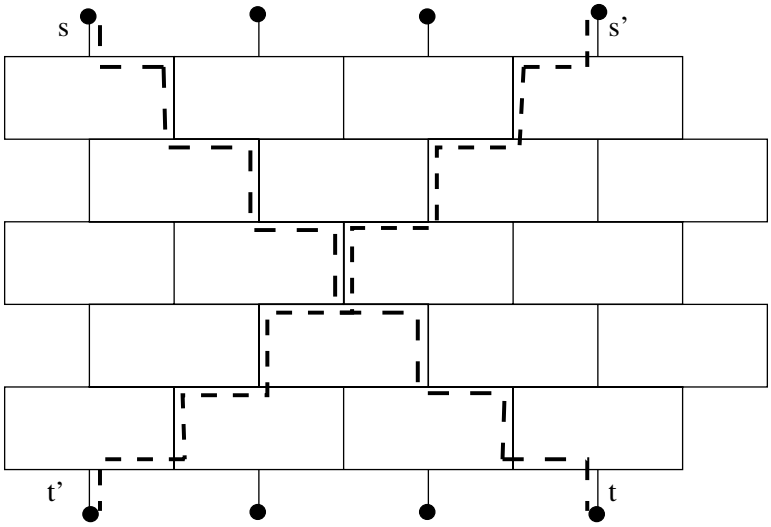


FIG. 2. *The brick wall network.*

dent set [BH92]. In particular, using the optimal (nonpolynomial time) algorithm for the problem (e.g., $\alpha = n$), we obtain the following result.

COROLLARY 13. *There exists an $O(\sqrt{n})$-competitive randomized algorithm for the on-line induced subgraph problem.*

4. On-line edge-disjoint paths. In this section we present $\Omega(n^\epsilon)$ lower bounds on the competitive ratio of nonpreemptive randomized algorithms for on-line edge-disjoint paths. As a consequence (see section 1.2), we obtain an $\Omega(n^\epsilon)$ lower bound for on-line call control. For this reason, in the following we refer to pairs of vertices as calls.

The lower bounds are derived from the lower bounds for the on-line induced subgraph problem presented in section 3 when the hereditary property is an independent set. Recall that the forbidden graph of Lemma 3 for an independent set can be a K_2, and thus the proofs of section 3 can be carried out with the graph G of n vertices being a K_n.

We say two calls are *consistent* if they can be connected with two edge-disjoint paths; otherwise they are *inconsistent*.

The lower bounds for on-line edge-disjoint paths we derive are obtained by embedding the graph $\tilde{G}$ and all input sequences used in the lower bounds proof for the on-line independent set within a network W with the following two properties:

1. Every vertex u in $\tilde{G}$ corresponds to one call $(s(u), t(u))$ in W.
2. Two vertices u, v in $\tilde{G}$ are not adjacent if and only if the two corresponding calls $(s(u), t(u))$, $(s(v), t(v))$ in W are consistent.

The network constructions we will present are based on the *brick wall* network. A brick wall means a portion of the hexagonal lattice as indicated in Figure 2. A brick wall of height h and width w consists of h horizontal paths with w edges between consecutive pairs of them.

We restrict our attention to calls from top vertices to bottom vertices of the brick wall. Top and bottom vertices of the network are numbered from left to right. We can easily observe in Figure 2 the following property.

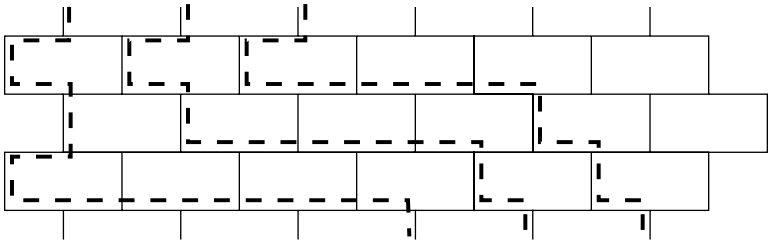


FIG. 3. *Three consistent calls in a brick wall of height* 3.

LEMMA 14. *Let $c = (s, t)$ and $c' = (s', t')$ be two top-to-bottom calls in W with $s < s'$. The calls c and c' are consistent if and only if $t < t'$.*

The following lemma gives a bound on the maximum number of top-to-bottom calls that can be routed in a brick wall network.

LEMMA 15. *Let h be the height of the brick wall, and let w be its width. Any set of up to $\ell = \min(h, w)$ pairwise consistent top-to-bottom calls can be routed in a brick wall network of height h and width w.*

Proof. Consider a set of pairwise consistent top-to-bottom calls. A feasible routing of the set of calls is as follows. Consider the calls from left to right. Assign at each call the leftmost available path that does not intersect a path assigned to any previous considered call. Figure 3 gives an example for a set of three calls in a brick wall of height 3. □

In section 4.1 we present a lower bound on the competitive ratio of nonpreemptive algorithms for on-line edge-disjoint paths, obtained by embedding the on-line independent set lower bounds of section 3 in a network of $O(n^{\frac{3}{2}})$ vertices. This result is then improved using a more complicated embedding in a network of $O(n^{\log_4 6})$ vertices.

4.1. A lower bound for on-line edge-disjoint paths. In this section we present the embedding in a brick wall W of the graph $\tilde{G}$ used in the lower bound for nonpreemptive randomized algorithms for on-line independent sets presented in section 3.

Let n be the number of vertices in $\tilde{G}$. The number of vertices at both top and bottom rows of W (indicated with dots in Figure 2) is n. The largest independent set in the graph $\tilde{G}$ of n vertices of the randomized lower bound for an on-line independent set has size $\sqrt{n}$. By setting the height of W to $\sqrt{n}$, we can ensure, by Lemma 15, that the optimal solution can be consistently routed. Thus, the number of vertices of the network W is $N = O(n^{\frac{3}{2}})$.

We associate every vertex $u \in \tilde{G}$ with a top-to-bottom call $(s(u), t(u))$ in W. The mapping will be such that two vertices are adjacent in $\tilde{G}$ if and only if the two calls are inconsistent. Each of the top and bottom vertices will appear in exactly one call.

Figure 4 shows how such embedding can be done at the ith level. For each of the four graphs of level $i-1$, $\tilde{G}^1_{i-1}$, $\tilde{G}^2_{i-1}$, $\tilde{G}^3_{i-1}$, and $\tilde{G}^4_{i-1}$ that make up $\tilde{G}_i$, a slice of the brick wall is reserved. The mapping of vertices to calls observes the same rules as for the embedding in the nonpreemptive lower bound of the previous subsection.

Observe in Figure 4 that the slices associated with adjacent graphs of Figure 1 intersect, while the slices associated with nonadjacent graphs are disjoint.

Since W has $N = n^{\frac{3}{2}}$ vertices, from Theorem 11 we get the following theorem.

THEOREM 16. *Any nonpreemptive randomized algorithm for the on-line edge-disjoint paths problem (and hence for call control) on general networks of N vertices has competitive ratio $\Omega(N^{\frac{2}{3}(1-\log_4 3)})$.*

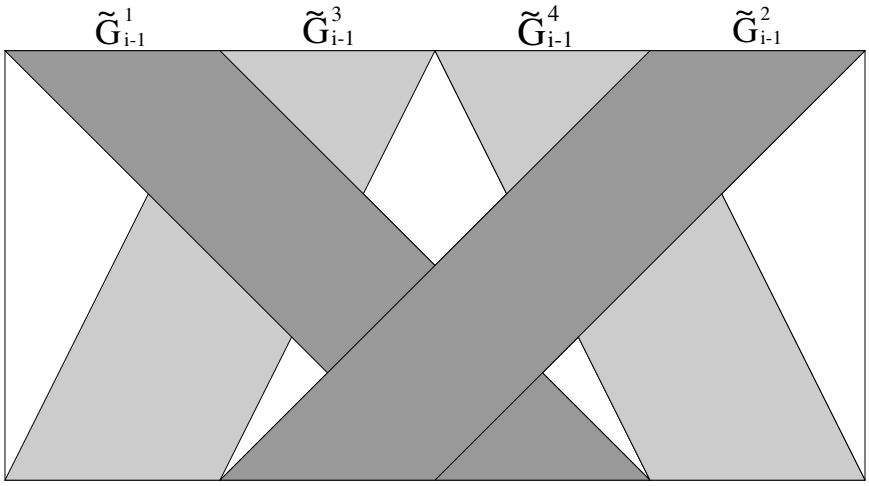


FIG. 4. *The mapping of the sequence in the graph $\tilde{G}_i$ on the brick wall.*

In the next section we will present a better lower bound for preemptive randomized algorithms.

4.2. Better results for on-line edge-disjoint paths. In this section we present an improved randomized lower bound for nonpreemptive on-line edge-disjoint paths.

As for the lower bounds for on-line edge-disjoint paths described in the previous subsections, the lower bound is obtained by embedding the lower bound for an on-line independent set (section 3) on a particular network. The transformation associates a call with every vertex in the instance of the on-line independent set lower bound.

Let $n(i)$ denote the number of vertices in the i-level graph of the on-line independent set lower bound. We describe a network construction with $N(i) = O(n(i)^{\log_4 6})$ nodes.

The construction is built recursively. The network is associated with a set of calls, only some of which can be routed consistently. The construction of a certain i-level network is composed of $(i-1)$-level networks.

The network H_i has 4^i input nodes and 3^i output nodes.

H_0 is the network having a single vertex, representing both one input and one output of the network.

The network H_i (Figure 5) is recursively composed of four instances of $(i-1)$-level networks, H^1_{i-1}, H^2_{i-1}, H^3_{i-1}, H^4_{i-1}, and one brick-wall network of width $4 \cdot 3^{i-1}$ and height 2^i.

The input of H_i is formed by the union of the sets of inputs for the four $(i-1)$-level networks.

The outputs of $H^1_{i-1}, H^2_{i-1}, H^3_{i-1}$, and H^4_{i-1} are directed into the appropriate four slices of the brick wall network, *while maintaining the respective order of the outputs within a slice.* The slices are chosen to intersect in such a way that the calls associated with H^1_{i-1} intersect with those associated with H^2_{i-1} and H^3_{i-1}, and the calls associated with H^2_{i-1} intersect with H^4_{i-1}.

We now describe in detail the transformation from the instance of the independent set lower bound to the network described above.

Let $\tilde{G}_i$ denote the i-level graph of the independent set lower bound. Recall that the graph $\tilde{G}_i$ is recursively composed of four $(i-1)$-level graphs, $\tilde{G}^1_{i-1}, \tilde{G}^2_{i-1}, \tilde{G}^3_{i-1}, \tilde{G}^4_{i-1}$.

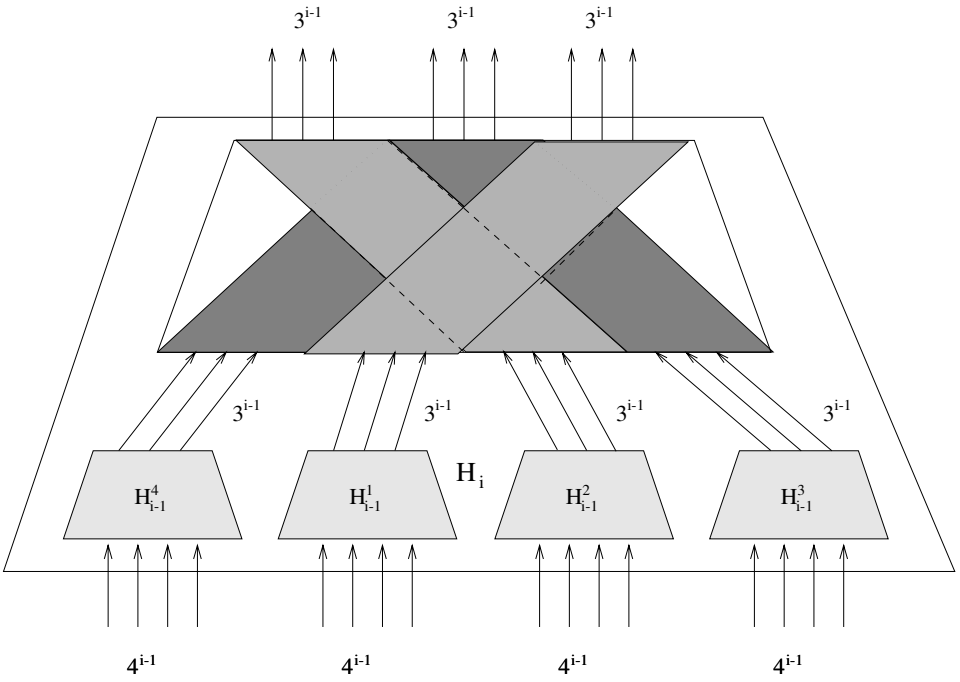


FIG. 5. *The construction of the network H_i.*

With each vertex v of $\tilde{G}_i$ we associate a corresponding call in the network H_i recursively.

If $v \in \tilde{G}^j_{i-1}$, then v is recursively associated with a call in H^j_{i-1} $(1 \le j \le 4)$.

For $i = 0$, the single vertex of $\tilde{G}_0$ is associated with the degenerate call (u, u), where u is the single vertex of H_0.

Let $v \in \tilde{G}^j_{i-1}$. Consider the associated call $c_{i-1}(v) = (s_{i-1}(v), t_{i-1}(v))$ in H^j_{i-1}. We define the associated call in H_i as the call $c_i(v) = (s_i(v), t_i(v))$, where $s_i(v) = s_{i-1}(v)$ and $t_i(v)$ is the output in the slice of the brick wall contained in H_i corresponding to the input on which $t_{i-1}(v)$ is directed.

LEMMA 17.

1. *Two nodes v_1, v_2 of $\tilde{G}_i$ are adjacent if and only if $c_i(v_1), c_i(v_2)$ are inconsistent in H_i.*
2. *Given an independent set T in $\tilde{G}_i$, all calls in the set $C(T) = \{c_i(v); v \in T\}$ can be routed consistently in H_i.*

Proof. The proof is by induction on i. For $i = 0$ the claims are trivial.

Assume that the claim holds for $i-1$. First we prove the first part of the lemma. Let us consider first the case in which two nodes v_1 and v_2 are in the same $(i-1)$-level graph. The corresponding calls $c_{i-1}(v_1)$ and $c_{i-1}(v_2)$ in the associated $(i-1)$-level network enter the brick wall of H_i in the same slice, and the order between $t_{i-1}(v_1)$ and $t_{i-1}(v_2)$ is maintained between $t_i(v_1)$ and $t_i(v_2)$. Therefore, $c_i(v_1)$ and $c_i(v_2)$ are consistent if and only if $c_{i-1}(v_1)$ and $c_{i-1}(v_2)$ are consistent, and thus the claim holds for the inductive hypothesis.

We are left to consider pairs of nodes in different $(i-1)$-level graphs.

Let v_1 and v_2 be nodes from two different $(i-1)$-level graphs, with the exception of $\tilde{G}^3_{i-1}$ and $\tilde{G}^4_{i-1}$ which are never presented together in the on-line independent set lower bound. Therefore $c_{i-1}(v_1)$ and $c_{i-1}(v_2)$ enter the brick wall of network H_i in

two different slices. It follows that the two calls $c_i(v_1)$ and $c_i(v_2)$ are inconsistent if and only if the two corresponding slices are intersecting and thus if and only if the two graphs are nonadjacent. This ends the proof of the first part of the lemma.

Now we turn to the second part of the lemma. We will prove the claim by induction on the level i of the network construction. For level i the independent set T is of size at most 2^i. For $i = 0$ there is only one call, and the claim is obvious.

Consider $i > 0$. The vertices of T can be partitioned into two subsets T_1 and T_2, belonging to two nonadjacent level $(i-1)$ graphs (each of size at most 2^{i-1}). Notice that the two nonadjacent graphs are either $\tilde{G}^1_{i-1}$ and $\tilde{G}^4_{i-1}$ or $\tilde{G}^2_{i-1}$ and $\tilde{G}^3_{i-1}$.

Therefore, without loss of generality, we can assume that T_1 belongs to $\tilde{G}^1_{i-1}$ and T_2 belongs to $\tilde{G}^4_{i-1}$, and thus for every vertex $v \in T_1$, $c_{i-1}(v)$ belongs to H^1_{i-1}, and for every vertex $v \in T_2$, $c_{i-1}(v)$ belongs to H^4_{i-1}.

By induction, the set of calls $\{c_{i-1}(v); v \in T_1\}$ can be routed consistently through H^1_{i-1}, and the set of calls $\{c_{i-1}(v); v \in T_2\}$ can be routed consistently through H^4_{i-1}.

The output vertices of these calls in H^1_{i-1} and H^4_{i-1} are all distinct, and so are the input vertices in the brick wall of H_i into which these calls are directed. Since the slices of the brick wall associated with H^1_{i-1} and H^4_{i-1} are nonintersecting, and since for each slice the order of the corresponding outputs in the brick wall of H_i is maintained, we have that every pair of calls in $C(T)$ can be routed consistently through the brick wall of H_i.

Since the number of such calls is at most 2^i, and thus at most the height of the brick wall of H_i, all calls in $C(T)$ can be routed consistently.

The case in which T_1 belongs to $\tilde{G}^2_{i-1}$ and T_2 belongs to $\tilde{G}^3_{i-1}$ can be similarly proved. □

The next lemma gives a bound on the number of vertices in an i-level network.

CLAIM 18. *The number of vertices/edges in H_i is $N(i) = O(6^i)$.*

Proof. Let α be some constant such that a brick wall of height h and width w has at most αhw vertices/edges. We show by induction on i that $N(i) \le 4\alpha \cdot 6^i$. For $i = 0$ the claim is clearly satisfied. Assume that $N(i-1) \le \alpha \cdot 6^{i-1}$. Since H_i is composed of four H_{i-1} networks and a brick wall of width $4 \cdot 3^{i-1}$ and height 2^i, we obtain

$$N(i) \le 4\alpha \cdot 3^{i-1} \cdot 2^i + 4 \cdot 4\alpha \cdot 6^{i-1} \le 4\alpha \cdot 6^i. \qquad \square$$

Using the network construction described, we derive from Theorem 11 the following result.

THEOREM 19. *Any randomized nonpreemptive algorithm for on-line edge-disjoint paths (and thus for call control) on general networks of N vertices has competitive ratio $\Omega(N^{\frac{1-\log_4 3}{\log_4 6}})$.*

4.3. Lower bounds for networks of high degree. The brick wall network has degree 3. However, we prove the following theorem, which extends any lower bound for on-line edge-disjoint paths to networks of minimum degree δ.

THEOREM 20. *Given an $\Omega(f(N))$ lower bound on the competitive ratio of nonpreemptive algorithms for on-line edge-disjoint paths (and thus for on-line call control) on networks of N vertices, there exists an $\Omega(f(\frac{N}{\delta}))$ lower bound for the problem on networks of minimum degree δ.*

Proof. We transform the network H of N vertices on which the lower bound is achieved into a network H' of minimum degree δ as follows. To every vertex corresponds a clique of size δ in the new network, and to every edge of H corresponds a single edge between two arbitrary vertices in the two associated cliques.

Let us consider the input sequence σ for the $\Omega(f(n))$ lower bound in H. We transform σ into an input sequence σ' for H' by replacing every call in σ formed by two vertices with a call formed by two arbitrary vertices in the two associated cliques. Under this transformation, any solution for σ' in H' corresponds to a solution for σ in H with equal size. Since H' has more vertices than H by a factor δ, the $\Omega(f(N))$ lower bound in the network H with input sequence σ yields an $\Omega(f(\frac{N}{\delta}))$ lower bound in H' with input sequence σ'. □

5. The on-line graph coloring problem. In this section we will consider the on-line graph coloring problem. We present a lower bound of $\Omega(n^\epsilon)$ on the competitive ratio of randomized on-line algorithms. We recall that in our model the algorithm knows the graph $G = (V, E)$ from which the instance of the problem is drawn.

Vertices of V are presented over time. Each time a new vertex is communicated, the algorithm must answer with a color for the vertex. The goal is to color the set of vertices presented with as few colors as possible, so that any two adjacent vertices have different colors.

In section 5.1 we present an $\Omega(n^\epsilon)$ lower bound for the on-line graph coloring problem, and in section 5.2 we give upper bounds for the problem.

5.1. Lower bounds for on-line graph coloring. The structure of the graph G from which the input sequence is chosen is recursive. Given a set of vertices $V = \{v_0, v_1, \ldots, v_{n-1}\}$, where n is a power of 5, we define subsets $V_i^j \subset V$, $|V_i^j| = 5^i$,

$$V_i^j = \{v_{j5^i}, v_{j5^i+1}, \ldots, v_{(j+1)5^i-1}\},$$

for all $0 \le i \le \log_5 n$, $0 \le j \le n/5^i - 1$. We also define the graphs $G_i^j = (V_i^j, E_i^j)$, $0 \le i \le \log_5 n$, $0 \le j \le n/5^i - 1$, where E_i^j is the set of edges in $E_{i-1}^{5j} \cup E_{i-1}^{5j+1} \cup \cdots \cup E_{i-1}^{5j+4}$, to which we add all edges between V_{i-1}^{5j+k} and $V_{i-1}^{5j+((k+1) \bmod 5)}$, $0 \le k \le 4$. The recursive construction of G_i^j is given pictorially in Figure 6.

Note that $V_{\log_5 n}^0 = V$, and we set $G = G_{\log_5 n}^0$ for all $0 \le i \le \log_5 n$ and $0 \le j \le n/5^i - 1$; G_i^j is a vertex induced subgraph of G.

As in section 3.1 we define a probability distribution on the request sequences of G. For any on-line vertex coloring algorithm ON we give a lower bound on the ratio between the number of different colors required by the online algorithm and the optimal number of colors (which is constant for all sequences with nonzero probability in the distribution).

The probability distribution over the sequences will be a uniform distribution over a set of sequences of vertices. The set of sequences is defined recursively as follows. The set of sequences for the graph G_0^j, $0 \le j \le n-1$, consists of the single sequence $\langle v_j \rangle$. Let $S(V_i^j)$ denote the set of sequences over V_i^j, $0 \le i \le \log_5 n$, $0 \le j \le n/5^i - 1$. Also, let $\|$ denote the concatenation operator on sequences.

Given $S(V_{i-1}^{5j})$, $S(V_{i-1}^{5j+1})$, $S(V_{i-1}^{5j+2})$, $S(V_{i-1}^{5j+3})$, and $S(V_{i-1}^{5j+4})$, $1 \le i \le \log_5 n$, $0 \le j \le n/5^i - 1$, we define the set $S(V_i^j)$ as the result of the following process:

Set

$$S_{023} = \{x\|y\|z | x \in S(V_{i-1}^{5j}), y \in S(V_{i-1}^{5j+2}), z \in S(V_{i-1}^{5j+3})\},$$

$$S_{0231} = \{x\|y | x \in S_{023}, y \in S(V_{i-1}^{5j+1})\}, \tag{29}$$

$$S_{0234} = \{x\|y | x \in S_{023}, y \in S(V_{i-1}^{5j+4})\}, \tag{30}$$

$$S(V_i^j) = S_{0231} \cup S_{0234}.$$

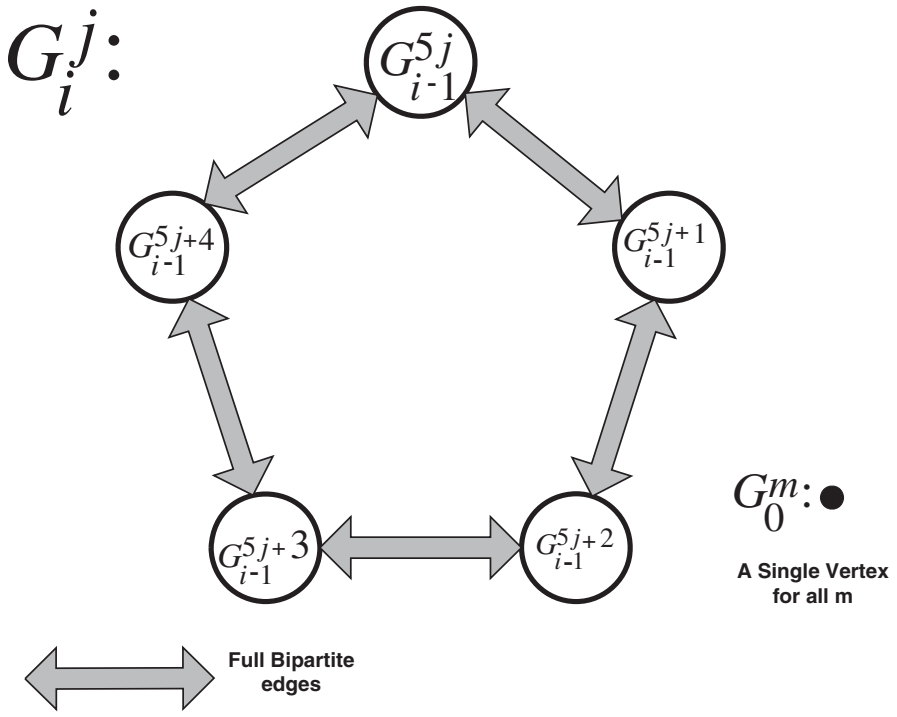


FIG. 6. *The recursive definition of* G_i.

CLAIM 21. *In any outcome of the above process (to generate* $S(V_i^j)$*),* $|S_{0231}| = |S_{0234}|$.

Proof. From step (29) above we have that $|S_{0231}| = |S_{023}| \cdot |S(V_{i-1}^{5j+1})|$, and by step (30) above we have $|S_{0234}| = |S_{023}| \cdot |S(V_{i-1}^{5j+4})|$. For any two sets V_{i-1}^{5j+k}, $V_{i-1}^{5j+\ell}$, $0 \le k < \ell \le 4$, we can find a mapping $f(u) \mapsto w$, $u \in V_{i-1}^{5j+k}$, $w \in V_{i-1}^{5j+\ell}$, such that both sets and their recursive partition into subsets are isomorphic under f. Specifically, $f(v_t) = v_{t+(\ell-k)5^{i-1}}$ will be such a mapping. Thus, $|S(V_{i-1}^{5j+1})| = |S(V_{i-1}^{5j+4})|$, and it now follows that $|S_{0231}| = |S_{0234}|$. □

CLAIM 22. $U(S(V_i^j | V_{i-1}^{5j+k})) = U(S(V_{i-1}^{5j+k}))$ *for* $k = 0, \ldots, 4$.

Proof. Every sequence in $S(V_{i-1}^{5j+k})$ appears as a subsequence of the same number of sequences in $S(V_i^j)$. □

Given a sequence σ of vertices from V, let $T(\sigma)$ denote the set of vertices in σ. For any algorithm A (online or offline) and any sequence of vertices σ from V, let $A^C(\sigma) : T(\sigma) \mapsto Z^+$ denote the function mapping vertices of $T(\sigma)$ onto colors.[2] Let $A^\#(\sigma)$ denote the number of different colors used by A to color the vertices of $T(\sigma)$:

$$A^\#(\sigma) = |\{A^C(\sigma)(u) : u \in T(\sigma)\}|.$$

Given a sequence σ of vertices from V and a set $V' \subset V$, we let $A(\sigma, V')$ denote

[2]For online algorithms, the function $A^C(\sigma)$ cannot depend on yet unseen parts of σ; i.e., if $\sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_{|\sigma|} \rangle$, $A^C(\sigma)(\sigma_i)$ cannot depend on the suffix $\langle \sigma_{i+1}, \ldots, \sigma_{|\sigma|} \rangle$.

the set of colors assigned by $A^C(\sigma)$ to the vertices of $V' \cap T(\sigma)$,

$$A(\sigma, V') = \{A^C(\sigma)(u) : u \in V' \cap T(\sigma)\}.$$

We also define

$$A^{\#}(\sigma, V') = |A(\sigma, V')|.$$

For example, $A^{\#}(\sigma, V) = A^{\#}(\sigma, T(\sigma)) = A^{\#}(\sigma)$.

LEMMA 23. *For any $0 \le i \le \log_5 n$, $0 \le j \le n/5^i - 1$, and any sequence $\sigma \in S(V_i^j)$, $\mathrm{OPT}^{\#}(\sigma) \le 2^i$.*

Proof. Given any sequence $\sigma \in S(V_i^j)$, we will prove by induction on i that the number of colors used to color the vertices of σ is no more than 2^i. For $i = 0$, σ is a single vertex and can be colored with one color.

For $i \ge 1$, $\sigma \in V_i^j$, either $\sigma \in S_{0231}$ or $\sigma \in S_{0234}$, as defined in (29) and (30). If $\sigma \in S_{0231}$, then we can use the same set of 2^{i-1} colors to color both the vertices in V_{i-1}^{5j} and V_{i-1}^{5j+2}; likewise, we can use the same set of 2^{i-1} colors to color the vertices in V_{i-1}^{5j+1} and in V_{i-1}^{5j+3}. Thus, if $\sigma \in S_{0231}$, then we can color the vertices of V_i^j with 2^i colors. A similar argument works for the case where $\sigma \in S_{0234}$. ∎

Let $E_{x \in_R D}[r(x)]$ denote the expectation of the random variable $r(x)$, where x is chosen from the distribution D.

Let $T \subset V$, the notation $\sigma|T$ denotes the sequence derived from σ by removing all elements in $V - T$.

CLAIM 24. *Given an online algorithm ON, for any sequences of vertices τ, τ', and τ'' from V, such that $T(\tau)$, $T(\tau')$, and $T(\tau'')$ are pairwise disjoint, there is an online algorithm $ON^{\tau'}$ such that*

$$\begin{aligned} ON^C(\tau' \| \tau \| \tau'')(u) &= {ON^{\tau'}}^C(\tau)(u) \quad \textit{for all} \ \ u \in T(\tau), \\ ON(\tau' \| \tau \| \tau'', T(\tau)) &= ON^{\tau'}(\tau, T(\tau)), \\ ON^{\#}(\tau' \| \tau \| \tau'', T(\tau)) &= {ON^{\tau'}}^{\#}(\tau, T(\tau)). \end{aligned}$$

Proof. Let $\tau = \tau_1, \tau_2, \ldots, \tau_{|\tau|}$. $ON^{\tau'}$ emulates the execution of ON as follows: $ON^{\tau'}$ colors vertex τ_ℓ, $1 \le \ell \le |\tau|$, with the same color that ON would have given τ_ℓ after coloring $\tau' \| \langle \tau_1, \tau_2, \ldots, \tau_{\ell-1} \rangle$. Thus,

$${ON^{\tau'}}^C(\tau)(\tau_\ell) = ON^C(\tau' \| \tau)(\tau_\ell). \qquad ∎$$

LEMMA 25. *For any deterministic online algorithm ON defined on sequences of vertices from V_i^j, $0 \le i \le \log_5 n$, $0 \le j \le n/5^i - 1$, the following hold:*

(31)
$$E_{\sigma \in_R U(S(V_i^j))}[ON^{\#}(\sigma, V_{i-1}^{5j})] = E_{\tau_0 \in_R U(S(V_{i-1}^{5j}))}[ON^{\#}(\tau_0)],$$

(32)
$$E_{\sigma \in_R U(S(V_i^j))}[ON^{\#}(\sigma, V_{i-1}^{5j+2})] = E_{\tau_0 \in_R U(S(V_{i-1}^{5j}))} E_{\tau_2 \in_R U(S(V_{i-1}^{5j+2}))}[{ON^{\tau_0}}^{\#}(\tau_2)],$$

(33)
$$\begin{aligned} &E_{\sigma \in_R U(S(V_i^j))}[ON^{\#}(\sigma, V_{i-1}^{5j+3})] \\ &= E_{\tau_0 \in_R U(S(V_{i-1}^{5j}))} E_{\tau_2 \in_R U(S(V_{i-1}^{5j+2}))} E_{\tau_3 \in_R U(S(V_{i-1}^{5j+3}))}[{ON^{\tau_0 \| \tau_2}}^{\#}(\tau_3)], \end{aligned}$$

(34)
$$E_{\sigma\in_R U(S(V_i^j))}[ON^{\#}(\sigma, V_{i-1}^{5j+1})]$$
$$= \frac{1}{2} E_{\tau_0\in_R U(S(V_{i-1}^{5j}))} E_{\tau_2\in_R U(S(V_{i-1}^{5j+2}))} E_{\tau_3\in_R U(S(V_{i-1}^{5j+3}))} E_{\tau_1\in_R U(S(V_{i-1}^{5j+1}))} [ON^{\tau_0\|\tau_2\|\tau_3\#}(\tau_1)],$$
(35)
$$E_{\sigma\in_R U(S(V_i^j))}[ON^{\#}(\sigma, V_{i-1}^{5j+4})]$$
$$= \frac{1}{2} E_{\tau_0\in_R U(S(V_{i-1}^{5j}))} E_{\tau_2\in_R U(S(V_{i-1}^{5j+2}))} E_{\tau_3\in_R U(S(V_{i-1}^{5j+3}))} E_{\tau_4\in_R U(S(V_{i-1}^{5j+4}))} [ON^{\tau_0\|\tau_2\|\tau_3\#}(\tau_4)].$$

Proof. The proof follows from Claim 24 and follows arguments similar to those made in Lemma 8. □

CLAIM 26. *For any function f such that*

$$f : \{\tau_0 \times \tau_2 \times \tau_3 | \tau_k \in S(V_{i-1}^{5j+k}), k = 0, 2, 3\} \mapsto Z^+$$

we have

$$\begin{aligned} &E_{\sigma\in S_{0231}}[f(\tau_0, \tau_2, \tau_3)] \\ &\quad = E_{\sigma\in S_{0234}}[f(\tau_0, \tau_2, \tau_3)] \\ &\quad = E_{\tau_0\|\tau_2\|\tau_3\|\tau'\in U(S(V_i^j)), \tau_k\in S(V_{i-1}^{5j+k}), k=0,2,3}[f(\tau_0, \tau_2, \tau_3)]. \end{aligned}$$

For any function f such that

$$f : \{\tau' | \tau' \in S(V_{i-1}^{5j+1})\} \mapsto Z^+$$

we have

$$E_{\sigma\in S_{0231}}[f(\tau')] = 2\ E_{\tau_0\|\tau_2\|\tau_3\|\tau'\in U(S(V_i^j)), \tau_k\in S(V_{i-1}^{5j+k}), k=0,2,3}[f(\tau')].$$

For any function f such that

$$f : \{\tau' | \tau' \in S(V_{i-1}^{5j+4})\} \mapsto Z^+$$

we have

$$E_{\sigma\in S_{0234}}[f(\tau')] = 2\ E_{\tau_0\|\tau_2\|\tau_3\|\tau'\in U(S(V_i^j)), \tau_k\in S(V_{i-1}^{5j+k}), k=0,2,3}[f(\tau')].$$

Proof. We have

$$\begin{aligned} E_{\sigma\in S_{0231}}[f(\tau_0, \tau_2, \tau_3)] &= \frac{\sum_{\tau_0\|\tau_2\|\tau_3\|\tau_1\in S_{0231}} f(\tau_0, \tau_2, \tau_3)}{|S_{0231}|} \\ &= \frac{|S(V_{i-1}^{5j+1})| \cdot \sum_{\tau_0\|\tau_2\|\tau_3\in S_{023}} f(\tau_0, \tau_2, \tau_3)}{|S_{023}| \cdot |S(V_{i-1}^{5j+1})|} \\ &= \frac{\sum_{\tau_0\|\tau_2\|\tau_3\in S_{023}} f(\tau_0, \tau_2, \tau_3)}{|S_{023}|} \\ &= \frac{|S(V_{i-1}^{5j+1}) \cup S(V_{i-1}^{5j+4})|}{|S(V_{i-1}^{5j+1}) \cup S(V_{i-1}^{5j+4})|} \cdot \frac{\sum_{\tau_0\|\tau_2\|\tau_3\in S_{023}} f(\tau_0, \tau_2, \tau_3)}{|S_{023}|} \\ &= \frac{\sum_{\tau_0\|\tau_2\|\tau_3\|\tau'\in S(V_i^j)} f(\tau_0, \tau_2, \tau_3)}{|S(V_i^j)|} \\ &= E_{\tau_0\|\tau_2\|\tau_3\|\tau'\in S(V_i^j)}[f(\tau_0, \tau_2, \tau_3)]. \end{aligned}$$

Similar arguments prove the other claims. ∎

LEMMA 27. *For any $1 \le i \le \log_5 n$, $0 \le j \le n/5^i - 1$, and any deterministic online algorithm ON defined over sequences from V_i^j,*

$$E_{\sigma \in_R U(S(V_i^j))}[ON^{\#}(\sigma)] \ge \left(\frac{5}{2}\right)^i.$$

Proof. We prove the claim by induction on i. The claim holds for $i = 0$ and any $1 \le j \le n$ (V_0^j is a single vertex). We assume by induction the claim holds for all probability distributions $U(S(V_{i-1}^{\ell}))$, $0 \le \ell \le n/5^{i-1} - 1$, which means in particular that it holds for the probability distributions $U(S(V_{i-1}^{5j+k}))$, $0 \le j \le n/5^i - 1$, $0 \le k \le 4$.

From Claim 21 we have that $|S_{0231}| = |S_{0234}|$ and so

$$E_{\sigma \in S(V_i^j)}[ON^{\#}(\sigma)] = (1/2)E_{\sigma \in S_{0231}}[ON^{\#}(\sigma)] + (1/2)E_{\sigma \in S_{0234}}[ON^{\#}(\sigma)]. \tag{36}$$

The set of colors used to color V_i^j is the union of the sets of colors used to color V_i^{5j+k}, $0 \le k \le 4$. For $\sigma \in S(V_i^j)$, where $\sigma \in S_{0231}$ as described in the construction of $S(V_i^j)$, we have $\sigma = \tau_0 \| \tau_2 \| \tau_3 \| \tau_1$, where $\tau_k \in S(V_{i-1}^{5j+k})$, $k = 0, 1, 2, 3$. For $\sigma \in S(V_i^j)$, where $\sigma \in S_{0234}$ as described in the construction of $S(V_i^j)$, we have $\sigma = \tau_0 \| \tau_2 \| \tau_3 \| \tau_4$, where $\tau_k \in S(V_{i-1}^{5j+k})$, $k = 0, 2, 3, 4$.

From (31), (32), (33), (34), and (35) we can now derive the following:

(37)
$$E_{\sigma \in S_{0231}}[ON^{\#}(\sigma)] \ge E_{\sigma \in S_{0231}} \left[\begin{array}{l} ON^{\#}(\sigma, T(\tau_0)) + ON^{\#}(\sigma, T(\tau_2)) \\ + ON^{\#}(\sigma, T(\tau_3)) + ON^{\#}(\sigma, T(\tau_1)) \\ - |ON(\sigma, T(\tau_0)) \cap (ON(\sigma, T(\tau_2)) \cup ON(\sigma, T(\tau_3)))| \\ - |ON(\sigma, T(\tau_1)) \cap ON(\sigma, T(\tau_3))| \end{array} \right],$$

(38)
$$E_{\sigma \in S_{0234}}[ON^{\#}(\sigma)] \ge E_{\sigma \in S_{0234}} \left[\begin{array}{l} ON^{\#}(\sigma, T(\tau_0)) + ON^{\#}(\sigma, T(\tau_2)) \\ + ON^{\#}(\sigma, T(\tau_3)) + ON^{\#}(\sigma, T(\tau_4)) \\ - |ON(\sigma, T(\tau_0)) \cap (ON(\sigma, T(\tau_2)) \cup ON(\sigma, T(\tau_3)))| \\ - |ON(\sigma, T(\tau_4)) \cap ON(\sigma, T(\tau_2))| \end{array} \right].$$

Now,

$$\begin{aligned} |ON(\sigma, T(\tau_0)) \cap (ON(\sigma, T(\tau_2)) \cup ON(\sigma, T(\tau_3)))| = & \; |ON(\sigma, T(\tau_0)) \cap ON(\sigma, T(\tau_2))| \\ & + |ON(\sigma, T(\tau_0)) \cap ON(\sigma, T(\tau_3))| \\ & - |ON(\sigma, T(\tau_2)) \cap ON(\sigma, T(\tau_3))| \\ = & \; |ON(\sigma, T(\tau_0)) \cap ON(\sigma, T(\tau_2))| \\ & + |ON(\sigma, T(\tau_0)) \cap ON(\sigma, T(\tau_3))|, \end{aligned} \tag{39}$$

because the colors used for V_{i-1}^{5j+2} and V_{i-1}^{5j+3} must be distinct.

By substituting (37) and (38) into (36) and making use of the equalities in (39), we obtain

$$
\begin{aligned}
(40)\quad E_{\sigma\in S(V_i^j)}[ON^{\#}(\sigma)] \geq\ & (1/2)E_{\sigma\in S_{0231}}\begin{bmatrix} ON^{\#}(\sigma,T(\tau_0))+ON^{\#}(\sigma,T(\tau_2)) \\ +ON^{\#}(\sigma,T(\tau_3))+ON^{\#}(\sigma,T(\tau_1)) \end{bmatrix} \\
& +(1/2)E_{\sigma\in S_{0234}}\left[ON^{\#}(\sigma,T(\tau_4))\right] \\
& +(1/2)E_{\sigma\in S_{0234}}\begin{bmatrix} ON^{\#}(\sigma,T(\tau_0)) \\ -\,|ON(\sigma,T(\tau_0))\cap ON(\sigma,T(\tau_2))| \\ -\,|ON(\sigma,T(\tau_0))\cap ON(\sigma,T(\tau_3))| \end{bmatrix} \\
& +(1/2)E_{\sigma\in S_{0234}}\begin{bmatrix} ON^{\#}(\sigma,T(\tau_2)) \\ -\,|ON(\sigma,T(\tau_4))\cap ON(\sigma,T(\tau_2))| \end{bmatrix} \\
& -(1/2)E_{\sigma\in S_{0231}}\left[|ON(\sigma,T(\tau_0))\cap ON(\sigma,T(\tau_2))|\right] \\
& +(1/2)E_{\sigma\in S_{0234}}\left[ON^{\#}(\sigma,T(\tau_3))\right] \\
& -(1/2)E_{\sigma\in S_{0231}}\begin{bmatrix} |ON(\sigma,T(\tau_0))\cap ON(\sigma,T(\tau_3))| \\ +\,|ON(\sigma,T(\tau_1))\cap ON(\sigma,T(\tau_3))| \end{bmatrix}.
\end{aligned}
$$

The following equations are derived from Claim 26 and by observing that the colors assigned to vertices V_{i-1}^{5j+k} must be distinct from the colors assigned to vertices $V_{i-1}^{5j+(k+1)\bmod 5}$, $k=0,\ldots,4$:

$$
(41)\qquad E_{\sigma\in S_{0234}}\left[\begin{array}{l} ON^{\#}(\sigma,T(\tau_0)) \\ -\,|ON(\sigma,T(\tau_0))\cap ON(\sigma,T(\tau_2))| \\ -\,|ON(\sigma,T(\tau_0))\cap ON(\sigma,T(\tau_3))| \end{array}\right] \geq 0,
$$

$$
\begin{aligned}
(42)\qquad & E_{\sigma\in S_{0234}}\left[\begin{array}{l} ON^{\#}(\sigma,T(\tau_2)) \\ -\,|ON(\sigma,T(\tau_4))\cap ON(\sigma,T(\tau_2))| \end{array}\right] \\
& -E_{\sigma\in S_{0231}}\left[\ |ON(\sigma,T(\tau_0))\cap ON(\sigma,T(\tau_2))|\ \right] \\
& =E_{\sigma\in S_{0234}}\left[\begin{array}{l} ON^{\#}(\sigma,T(\tau_2)) \\ -\,|ON(\sigma,T(\tau_4))\cap ON(\sigma,T(\tau_2))| \\ -\,|ON(\sigma,T(\tau_0))\cap ON(\sigma,T(\tau_2))| \end{array}\right] \geq 0,
\end{aligned}
$$

$$
\begin{aligned}
(43)\qquad & E_{\sigma\in S_{0234}}\left[\ ON^{\#}(\sigma,T(\tau_3))\ \right] \\
& -E_{\sigma\in S_{0231}}\left[\begin{array}{l} |ON(\sigma,T(\tau_0))\cap ON(\sigma,T(\tau_3))| \\ +\,|ON(\sigma,T(\tau_1))\cap ON(\sigma,T(\tau_3))| \end{array}\right] \\
& =E_{\sigma\in S_{0234}}\left[\begin{array}{l} ON^{\#}(\sigma,T(\tau_3)) \\ -\,|ON(\sigma,T(\tau_0))\cap ON(\sigma,T(\tau_3))| \\ -\,|ON(\sigma,T(\tau_1))\cap ON(\sigma,T(\tau_3))| \end{array}\right] \geq 0.
\end{aligned}
$$

Using (41), (42), (43) and Claim 26, we derive the following from (40):

$$
\begin{aligned}
(44)\quad E_{\sigma\in S(V_i^j)}[ON^{\#}(\sigma)] \geq\ & (1/2)E_{\sigma\in S_{0231}}\left[\begin{array}{l} ON^{\#}(\sigma,T(\tau_0))+ON^{\#}(\sigma,T(\tau_2)) \\ +ON^{\#}(\sigma,T(\tau_3))+ON^{\#}(\sigma,T(\tau_1)) \end{array}\right] \\
& +(1/2)E_{\sigma\in S_{0234}}\left[\ ON^{\#}(\sigma,T(\tau_4))\ \right] \\
=\ & (1/2)E_{\sigma\in S(V_i^j)}\left[\begin{array}{l} ON^{\#}(\sigma,T(\tau_0))+ON^{\#}(\sigma,T(\tau_2)) \\ +ON^{\#}(\sigma,T(\tau_3)) \end{array}\right] \\
& +E_{\sigma\in S(V_i^j)}\left[\ ON^{\#}(\sigma,T(\tau_1))+ON^{\#}(\sigma,T(\tau_4))\ \right].
\end{aligned}
$$

Now, observe that for every $\sigma = \tau_0 \| \tau_2 \| \tau_3 \| \tau_1 \in U(S(V_i^j))$, $\tau_k \in U(S(V_{i-1}^{5j+k}))$, $k =, 0, 2, 3, 1$, we have

$$T(\sigma) \cap V_{i-1}^{5j+k} = T(\tau_k),$$

and for every $\sigma = \tau_0 \| \tau_2 \| \tau_3 \| \tau_4 \in U(S(V_i^j))$, $\tau_k \in U(S(V_{i-1}^{5j+k}))$, $k =, 0, 2, 3, 4$,

$$T(\sigma) \cap V_{i-1}^{5j+k} = T(\tau_k).$$

The two relations above allow us to apply Lemma 25 to (44), to obtain

(45)

$$\begin{aligned} & E_{\sigma \in S(V_i^j)}[ON^{\#}(\sigma)] \\ & \geq \left(\frac{1}{2}\right) \begin{bmatrix} E_{\tau_0 \in_R U(S(V_{i-1}^{5j+k}))}[ON^{\#}(\tau_0)] \\ + E_{\tau_0 \in_R U(S(V_{i-1}^{5j+k}))} E_{\tau_2 \in_R U(S(V_{i-1}^{5j+2}))}[ON^{\tau_0 \#}(\tau_2)] \\ + E_{\tau_0 \in_R U(S(V_{i-1}^{5j+k}))} E_{\tau_2 \in_R U(S(V_{i-1}^{5j+2}))} E_{\tau_3 \in_R U(S(V_{i-1}^{5j+3}))}[ON^{\tau_0 \| \tau_2 \#}(\tau_3)] \\ + E_{\tau_0 \in_R U(S(V_{i-1}^{5j+k}))} E_{\tau_2 \in_R U(S(V_{i-1}^{5j+2}))} E_{\tau_3 \in_R U(S(V_{i-1}^{5j+3}))} E_{\tau_1 \in_R U(S(V_{i-1}^{5j+1}))} \\ \quad \cdot [ON^{\tau_0 \| \tau_2 \| \tau_3 \#}(\tau_1)] \\ + E_{\tau_0 \in_R U(S(V_{i-1}^{5j+k}))} E_{\tau_2 \in_R U(S(V_{i-1}^{5j+2}))} E_{\tau_3 \in_R U(S(V_{i-1}^{5j+3}))} E_{\tau_4 \in_R U(S(V_{i-1}^{5j+4}))} \\ \quad \cdot [ON^{\tau_0 \| \tau_2 \| \tau_3 \#}(\tau_4)] \end{bmatrix}. \end{aligned}$$

By the inductive hypothesis we get that (45) is at least

$$\begin{aligned} E_{\sigma \in S(V_i^j)}[ON^{\#}(\sigma)] &\geq \frac{5}{2} \left(\frac{5}{2}\right)^{i-1} \\ &= \left(\frac{5}{2}\right)^{i}, \end{aligned}$$

thus proving the lemma. □

We then conclude with the following result.

THEOREM 28. *Any randomized algorithm for the on-line vertex coloring problem has a competitive ratio of* $\Omega(n^{1-\log_5 4})$.

Proof. By Lemmas 23 and 27 we have the following. For any $0 \leq i \leq \log_5 n$, $0 \leq j \leq n/5^i - 1$, and any sequence $\sigma \in S(V_i^j)$, $\mathrm{OPT}^{\#}(\sigma) = 2^i$. For any $0 \leq i \leq \log_5 n$ and $0 \leq j \leq n/5^i - 1$, the number of colors required by any deterministic on-line algorithm over the distribution $U(S(V_i^j))$ satisfies

$$E_{\sigma \in_R U(S(V_i^j))}[ON^{\#}(\sigma)] \geq \left(\frac{5}{2}\right)^{i}.$$

The ratio $\rho(i, j)$ between the expected on-line number of colors and the expected optimal number of colors and for the probability distribution on the input sequences $U(S(V_i^j))$, $0 \leq i \leq \log_5 n$, $0 \leq j \leq n/5^i - 1$, is

$$\begin{aligned} \rho(i, j) &= \frac{E_{\sigma \in_R U(S(V_i^j))}[\mathrm{ON}^{\#}(\sigma)]}{E_{\sigma \in_R U(S(V_i^j))}[\mathrm{OPT}^{\#}(\sigma)]} \\ &\geq \frac{\left(\frac{5}{2}\right)^{i}}{2^i} \qquad (46) \\ &= \left(\frac{5}{4}\right)^{i}. \end{aligned}$$

For $i = \log_5 n$ we obtain that for any online algorithm the ratio $\rho(\log_5 n, 0)$ for sequences drawn from the probability distribution $U(S(V^0_{\log_5 n}))$ is

$$\begin{aligned}\rho(\log_5 n, 0) &\geq \frac{n}{4^{\log_5 n}} \\ &= n^{1-\log_5 4}.\end{aligned}$$

Using Yao's lemma (Lemma 1) yields the claimed result. □

5.2. Upper bounds for on-line graph coloring. Using techniques similar to those in section 3.2, we show that off-line algorithms for maximum independent sets can also be used to obtain deterministic on-line graph coloring algorithms.

Let $G = (V, E)$ be the graph that we assume is known to the on-line algorithm. The algorithm requires a preprocessing phase similar to the algorithm presented in section 3.2 for an on-line induced subgraph. A collection of k independent sets $I_1, I_2, \ldots, I_k$ of the graph G are computed as follows:

Let $G_1 = G$, and $G_{i+1} = G_i - I_i$ be the result of the operation that removes from G_i all the vertices of the independent set I_i and all the incident edges.

In particular, I_i is the set of vertices forming the solution of the independent set problem returned by applying the $\frac{n}{\alpha}$-approximate algorithm to the graph G_i. The process of computing solutions is continued until $|I_{k+1}| \leq \sqrt{\alpha}$. Let $\overline{I} = V - \{I_1 \cup I_2 \cup \cdots \cup I_k\}$ be the set of vertices of G_{k+1}.

The algorithm colors every presented vertex v as follows:

1. If there exist $i \leq k$ such that $v \in I_i$, assign color i to vertex v.
2. Otherwise, if $v \in \overline{I}$, assign the first color $j > k$ not used for any previously presented vertex $u \in \overline{I}$, $(u, v) \in E$.

THEOREM 29. *Given an $\frac{n}{\alpha}$-approximation (deterministic) algorithm for a maximum independent set, there exists a (deterministic) $O(\frac{n}{\sqrt{\alpha}})$-competitive algorithm for on-line minimum graph coloring. If the original algorithm is polynomial and deterministic, so is the on-line algorithm.*

Proof. First note that k is at most $\frac{n}{\sqrt{\alpha}}$. Next, observe that the maximum size of an independent set in $\overline{I}$ is at most $\frac{n}{\sqrt{\alpha}}$, since the size of the solution of the $\frac{n}{\alpha}$-approximate algorithm applied to the graph induced by $\overline{I}$ is less than $\sqrt{\alpha}$. Let $\overline{I}'$ be the subset of $\overline{I}$ presented in the sequence.

If the sequence is not empty, the number of colors of the optimal solution is at least the size of $\overline{I}'$ divided by the size of a maximum independent set in $\overline{I}'$. Therefore

$$\mathrm{OPT} \geq \max\left\{1, \frac{\sqrt{\alpha}}{n}|\overline{I}'|\right\}.$$

The number of colors used by the on-line algorithm is bounded by

$$\mathrm{ON} \leq k + |\overline{I}'| \leq 2\frac{n}{\sqrt{\alpha}}\mathrm{OPT},$$

thus proving the theorem. □

Using the optimal (nonpolynomial time) algorithm for a maximum independent set (e.g., $\alpha = n$) we obtain the following.

COROLLARY 30. *There exists a deterministic $O(\sqrt{n})$-competitive algorithm for the on-line graph coloring problem.*

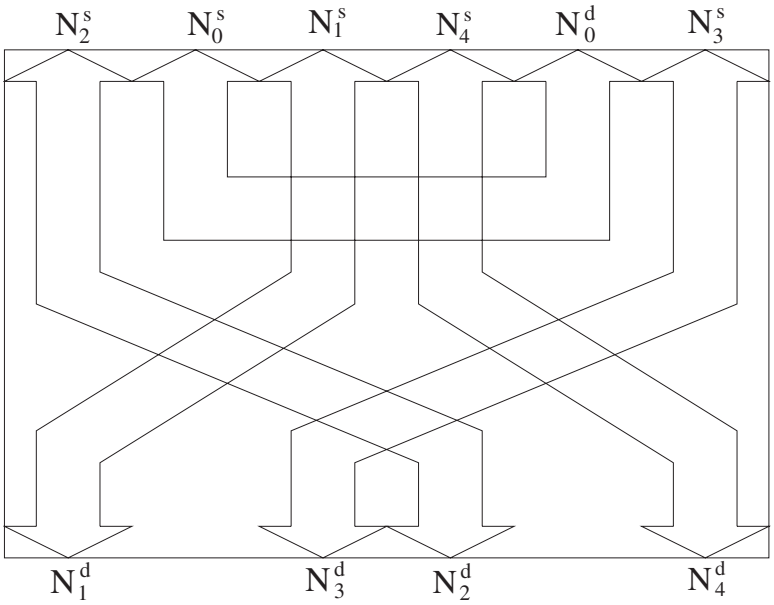


FIG. 7. *The mapping on the brick wall of a sequence of graph* G_i^j.

6. Lower bound for on-line path coloring. In this section we derive an $\Omega(n^\epsilon)$ competitive randomized lower bound for on-line path coloring. In the on-line path coloring problem, a pair of vertices of the network is presented at every step. The algorithm must reply with a color before the pairs following in the sequence are known. The coloring must obey the constraint that any collection of pairs assigned the same color can be connected with edge-disjoint paths.

The lower bound for on-line path coloring is obtained by transforming every sequence of vertices in the graph G (see Figure 6) used for the graph coloring lower bound into a sequence of pairs in the brick wall W shown in Figure 7. We say that two pairs are *consistent* if they can be connected through edge-disjoint paths; otherwise they are *inconsistent*. To derive the result it is sufficient to show that the embedding satisfies the two following properties:

1. Any two adjacent vertices of G are mapped to two inconsistent pairs.
2. Every independent set of G is mapped to a collection of mutually consistent pairs.

Consider a graph G_i^j, $i = 1, \ldots, \log_5 n$, $j = 0, \ldots, n/5^i - 1$, as defined in section 5. G_i^j is formed of five graphs G_{i-1}^{5j+k}, $k = 0, \ldots, 4$. The embedding, described in Figure 7, associates a top-to-bottom call in the brick wall of level i to each vertex of graph G_{i-1}^{5j+k}, $k = 1, \ldots, 4$, and a top-to-top call to every vertex of graph G_{i-1}^{5j}. More precisely, the vertices of G^i are recursively associated with calls from vertices N_i^s to vertices N_i^d. Property 1 then immediately follows, since every two adjacent vertices are mapped to inconsistent pairs.

It is also rather simple to see that property 2 holds if the brick wall of level i has height 2^i and width 6^i. Every sequence of $S(V_i^j)$ contains an independent set of size at most 2^i. The independent set is formed of vertices from at most two nonadjacent sets of vertices of G_i^j, e.g., V_{i-1}^{5j+1} and V_{i-1}^{5j+3} or V_{i-1}^{5j} and V_{i-1}^{5j+2}. If all vertices of the independent set are mapped to top-to-bottom calls, as in the case when they belong to

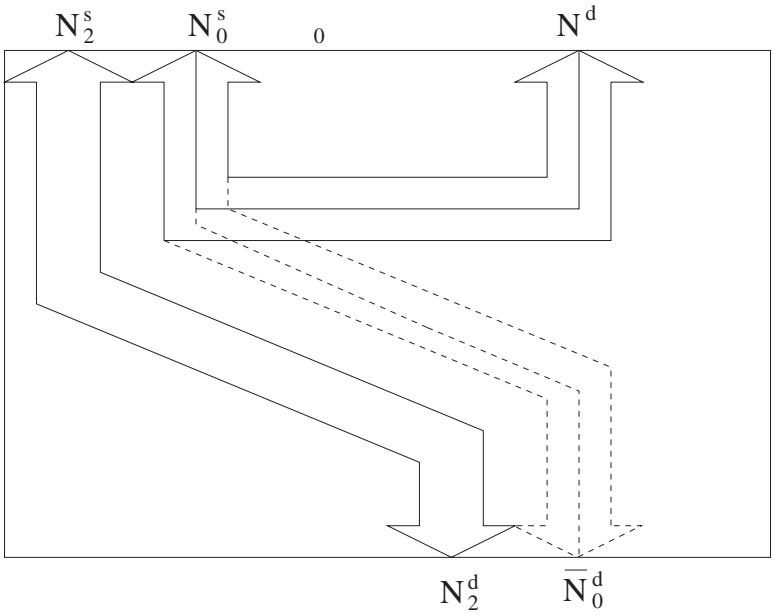


FIG. 8. *The routing of top-to-top calls.*

V_{i-1}^{5j+1} and V_{i-1}^{5j+3}, by Lemma 15, the associated pairs are mutually consistent in a brick wall of height 2^i. Alternatively, consider the case of an independent set containing vertices of V_{i-1}^{5j} mapped to top-to-top calls, and vertices mapped to top-to-bottom calls, for instance, of set V_{i-1}^{5j+2}.

For the sake of the analysis, assume for the moment that every vertex of V_{i-1}^{5j} is mapped, as shown in Figure 8, to two top-to-bottom calls, rather than one top-to-top call. The first call is from a vertex of N_0^s to a vertex of $\overline{N}_0^d$, while the second call is from a vertex of $\overline{N}_0^s$ to a vertex of N_0^d. (Observe that the vertices of $\overline{N}_0^d$ and $\overline{N}_0^s$ are not endpoints of other calls of the sequence.) By Lemma 15, a collection of calls from N_0^s to $\overline{N}_0^d$ and from N_2^s to N_2^d associated with an independent set G of size at most 2^i can be consistently routed.

Consider also the edge-disjoint paths connecting pairs, associated with vertices of V_{i-1}^{5j} in the independent set, formed of a vertex in $\overline{N}_0^s$ and a vertex in N_0^d . A set of edge-disjoint paths connecting the top-to-top pairs associated with vertices of V_{i-1}^{5j} is then obtained as follows: For every vertex of the independent set, let x be the intersection of the two paths connecting the associated pairs from N_0^s to $\overline{N}_0^d$ and from $\overline{N}_0^s$ to N_0^d. The path from N_0^s to N_0^d is then formed, as shown in Figure 8, by the segment between the vertex of N_0^s and x, followed by the segment between x and the vertex of N_0^d. Every such path is disjoint from any other path from N_0^s to N_0^d and from N_2^s to N_2^d connecting pairs associated with vertices of the independent set. Property 2 is then proved.

Let n be the number of vertices of graph G. Every input sequence for the graph $G = G_{\log_5 n}^0$ is therefore mapped in a brick wall W of size $N = 6^i \times 2^i$. We then conclude with the following theorem.

THEOREM 31. *Any randomized algorithm for on-line path coloring on general*

networks of N vertices has competitive ratio $\Omega(N^{\frac{\log 5}{\log 12}\log_5 4})$.

Proof. The theorem follows from Theorem 28 and by observing that any input sequence of vertices of the graph G of n vertices can be mapped into a brick wall of $N = n^{\frac{\log 12}{\log 5}}$ vertices. □

7. Lower bounds for optical networks. As pointed out in the introduction, a lower bound for on-line edge-disjoint paths is also a lower bound for on-line routing in reconfigurable networks (benefit version), and a lower bound for on-line path coloring is also a lower bound for on-line routing in reconfigurable networks (coloring version). Therefore, from Theorems 19 and 31, we derive the following two corollaries.

COROLLARY 32. *Any nonpreemptive randomized algorithm for on-line routing in reconfigurable optical networks (benefit version) on general networks of N vertices has competitive ratio* $\Omega(N^{\frac{1-\log_4 3}{\log_4 6}})$.

COROLLARY 33. *Any randomized algorithm for on-line routing in reconfigurable optical networks (coloring version) on general networks of N vertices has competitive ratio* $\Omega(N^{\frac{2}{3}(1-\log_4 3)})$.

In what follows we present a simple result (Lemma 36) that allows us to reduce maximum independent sets to routing in switchless optical networks (benefit version) and minimum graph coloring to routing in switchless optical networks (coloring version). This reduction is from a graph of n vertices to a network of $O(n)$ vertices. Therefore, from Theorems 11 and 28 we respectively derive the following two corollaries.

COROLLARY 34. *Any nonpreemptive randomized algorithm for on-line routing in switchless optical networks (benefit version) on general networks of N vertices has competitive ratio* $\Omega(N^{1-\log_4 3})$.

COROLLARY 35. *Any randomized algorithm for on-line routing in switchless optical networks (coloring version) on general networks of N vertices has competitive ratio* $\Omega(N^{1-\log_4 3})$.

The required reductions are derived from the following lemma.

LEMMA 36. *For any undirected graph $G = (V, E)$ there exist a directed network $D' = (V', E')$ and a one-to-one mapping from vertices v in G to calls $(s(v), t(v))$ in D' so that two vertices $v_i, v_j \in V$ are adjacent if and only if the two corresponding calls $(s(v_i), t(v_i))$, $(s(v_j), t(v_j))$ are conflicting.*

Proof. Let n be the number of vertices of $V = \{v_1, \ldots, v_n\}$. D' is a bipartite graph whose set of vertices V' is formed by the union of two sets of n vertices, $S = \{s_1, \ldots, s_n\}$ and $T = \{t_1, \ldots, t_n\}$. The one-to-one mapping is defined by $s(v_i) = s_i$ and $t(v_i) = t_i$, $i = 1, \ldots, n$. The set of directed edges of D' is $E' = \{(s(v), t(v)) | v \in V\} \cup \{(s(v_i), t(v_j)) | (v_i, v_j) \in E\}$.

For any two vertices $v_i, v_j \in V$ consider the two calls $(s(v_i), t(v_i))$ and $(s(v_j), t(v_j))$ in D'. If v_i and v_j are adjacent in E, then the bipartite graph D' contains the two edges $(s(v_i), t(v_j))$ and $(s(v_j), t(v_i))$. Therefore, $t(v_j) \in R(s(v_i))$ and $t(v_i) \in R(s(v_j))$, implying that the two calls are interfering. For the other direction, if v_i and v_j are nonadjacent, then $t(v_j) \notin R(s(v_i))$ and $t(v_j) \notin R(s(v_i))$. Since the graph D' is bipartite, with all edges directed from S to T, it follows that $u_2 \notin R(v_1)$ and $u_1 \notin R(v_2)$, implying that the two calls are noninterfering. □

8. Lower bounds for on-line edge-disjoint fixed paths. In this section we present tight lower bounds for on-line edge-disjoint fixed paths, an $\Omega(\sqrt{n})$ lower bound for *meshes* and an $\Omega(n)$ lower bound for *general networks*. We use the terminology of call control.

We make use of Yao's lemma (see section 2) and establish a lower bound for any deterministic algorithm on a given probability distribution. First we describe the probability distribution we use in proving the lower bounds.

The generic call c_i of an input sequence is specified by a path p_i connecting endpoints s_i and t_i. Two calls c_i and c_j from the input sequence are said to be *inconsistent* if p_i and p_j intersect on at least one edge ($p_i \cap p_j \neq \emptyset$), and *consistent* otherwise ($p_i \cap p_j = \emptyset$).

Every input sequence having nonzero probability is formed of $2k$ calls presented in k steps according to the following probability distribution:

1. At step j, $j = 1, \ldots, k$, two inconsistent calls c_j^1 and c_j^2 are presented.
2. Toss a fair coin.
   - If heads, all the calls presented later will be consistent with c_j^1 and inconsistent with c_j^2.
   - If tails, all the calls presented later will be consistent with c_j^2 and inconsistent with c_j^1.

LEMMA 37. *Any nonpreemptive randomized algorithm for on-line edge-disjoint fixed paths has competitive ratio $\Omega(k)$ on the probability distribution of k steps described above.*

Proof. There is a maximal set of k mutually consistent calls. This set is formed by selecting for every step of the sequence the call that is consistent with all the calls presented in the future. The claim then follows by proving an $O(1)$ upper bound on the expected on-line benefit.

Let $p(i)$ be the probability that an on-line algorithm has accepted i calls, $i \geq 1$, at some point along the execution of the algorithm. We prove by induction on i that $p(i) \leq \frac{1}{2^{i-1}}$. For the base of the induction, $p(1) \leq 1$. Assume $p(i-1) \leq \frac{1}{2^{i-2}}$, $i \geq 2$. Let A_i, $i \geq 2$, be the event "The algorithm is able to accept one additional call after having already accepted $i-1$ calls." Let c be the $(i-1)$th call accepted by the algorithm. The probability of event A_i is $p(A_i) \leq 1/2$ since, with probability $1/2$, all the calls presented after c are inconsistent with c itself. We then have $p(i) \leq p(A_i)p(i-1) \leq \frac{1}{2^{i-1}}$. Let $p(=i)$ be the probability that the on-line algorithm has accepted exactly i calls along its execution. We clearly have $p(=i) \leq p(i)$. Therefore, the expected on-line benefit is bounded by $E[\mathrm{ON}] \leq \sum_{i=1}^{k} ip(=i) \leq \sum_{i=1}^{k} i\frac{1}{2^{i-1}} = O(1)$. ∎

We will present in the following two sections the embedding of the probability distribution described in this section on a mesh topology and a general network. The obvious goal is to have k as large as possible with respect to the size of the network. In the two following subsections we will implement the probability distribution on specific networks to provide lower bounds on mesh topologies and general networks.

8.1. Lower bound on meshes. Consider a two-dimensional $\sqrt{n} \times \sqrt{n}$ mesh. Let the set of nodes be $\{(u,v) | 1 \leq u \leq \sqrt{n}, 1 \leq v \leq \sqrt{n}\}$. Every vertex is denoted with a pair; the first item is the row index, and the second item is the column index.

For our construction we use a network (see Figure 9, where the row index increases from top to bottom, the column index from left to right) derived from a two-dimensional mesh via the two following operations:

1. Remove all vertices (u,v) with $v > u$, and the adjacent edges.
2. Insert the edges $\{((u,v),(u-1,v+1)) | 3 \leq u \leq \sqrt{n}, 1 \leq v \leq \sqrt{n}-1\}$.

Such network can always be embedded in a mesh of size $2\sqrt{n} \times 2\sqrt{n}$.

We prove that any instance drawn from the probability distribution for the $\Omega(k)$ lower bound of the previous section can be implemented on this network with $k =$

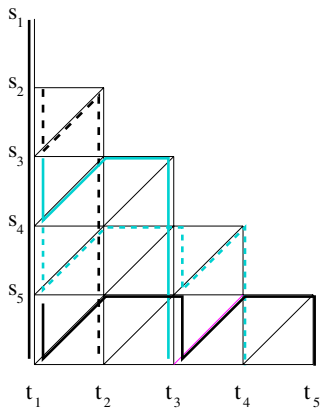


FIG. 9. *The network for the lower bound on meshes.*

$\frac{\sqrt{n}-1}{2}$. This is done by presenting a sequence of $\sqrt{n}-1$ calls; the path of each call is chosen in order to make it consistent or inconsistent with any other previous call in the sequence, as required from the outcome of the coin tosses.

Call c_i in the sequence is defined by a pair of vertices (s_i, t_i) and by a path p_i that connects s_i to t_i. Let $C_i = \{j : j < i, p_i \cap p_j = \emptyset\}$ be the set of previous calls in the input sequence from the probability distribution that are consistent with c_i, and let $I_i = \{j : j < i, p_i \cap p_j \neq \emptyset\}$ be the set of previous calls that are inconsistent with c_i.

Call c_i of the sequence, $i = 1, \ldots, \sqrt{k}-1$, is defined as follows:

- $s_i = (i, 1)$;
- $t_i = (\sqrt{n}, i)$;
- p_i is formed by the following sets of edges:
  1. $E_i^1 = \{((i,j),(i,j+1)) | j \in C_i\}$;
  2. $E_i^2 = \{((i,j),(i+1,j)) \text{ and } ((i+1,j),(i,j+1)) | j \in I_i\}$;
  3. $E_i^3 = \{((j,i),(j+1,i)) | i \leq j \leq \sqrt{n}-1\}$.

Observe that the set of edges forming p_i, i.e., $E_i^1 \cup E_i^2 \cup E_i^3$, is actually a path from s_i to t_i. For $j = 1, \ldots, i-1$, vertex (i,j) is connected to vertex $(i, j+1)$, either directly if $j \in C_i$ or through the two-edge path formed by $((i,j),(i+1,j))$ and $((i+1,j),(i,j+1))$ if $j \in I_i$. For $j = i, \ldots, \sqrt{n}-1$, vertex (j,i) is directly connected to vertex $(j+1,i)$. Moreover, the first vertex of the path is $s_i = (i,1)$, and the last vertex is $(\sqrt{n}, i)$.

Figure 9 shows a network with a sequence of five calls and the relative paths, where c_1 and c_3 are inconsistent and c_2 and c_4 consistent with respect to following calls.

LEMMA 38. *Let c_j and c_i, $j < i$, be two calls in the sequence. Calls p_i and p_j do not intersect if and only if $j \in C_i$.*

Proof. First we show that if $j \in I_i$, then c_i and c_j intersect in one edge, namely $((i,j),(i+1,j))$. In fact, this edge always belongs to p_j, while it belongs to p_i only if c_j is inconsistent with c_i.

Next, we show that if $j \in C_i$, then p_i and p_j do not intersect in any edge. For any h, all edges in $E_h^1 \cup E_h^2$ are either between two vertices in row h or connect a vertex in row h to a vertex in row $h+1$. Hence, the set $E_i^1 \cup E_i^2$ does not intersect with the set $E_j^1 \cup E_j^2$.

Since all edges in p_j have endpoints of column at most j, and all edges in E_i^3 have endpoints of column $i > j$, the two sets do not intersect.

It remains to show that the edges in E_j^3 do not intersect with the edges in $E_i^1 \cup E_i^2$. This follows since every edge of E_j^3 has both endpoints on column j, whereas since $j \in C_i$, no edge in $E_i^1 \cup E_i^2$ has both endpoints on column j. □

We then conclude with the following theorem.

THEOREM 39. *Any nonpreemptive randomized algorithm for on-line edge-disjoint with fixed paths on meshes of n nodes has competitive ratio* $\Omega(\sqrt{n})$.

8.2. Lower bound on general networks. We prove an $\Omega(n)$ lower bound on the competitive ratio of randomized algorithms for the on-line edge-disjoint fixed paths problem in general networks by showing that the probability distribution at the basis of Lemma 37 can be constructed for a sequence of $\Omega(n)$ steps, in a complete graph with $O(n)$ vertices.

Let n be a *prime number*, and let $V \cup V'$ with $V = \{v(0), \ldots, v(n-1)\}$ and $V' = \{v'(0), \ldots, v'(n-1)\}$ be the set of vertices of a complete graph G.

We define

$$n_h^l = (h \cdot l) \bmod n,$$

with $h = 1, \ldots, \lfloor \frac{n-1}{2} \rfloor$ and $l = 1, \ldots, n$.

The two following lemmas follow from simple algebraic properties of prime numbers.

LEMMA 40. *Let* (h, l) *and* (h', l') *be such that* $h, h' \in \{1, \ldots, \frac{n-1}{2}\}$, $h \neq h'$, *and* $l, l' \in \{1, \ldots, n-1\}$. *Then,* $\{n_h^l, n_h^{l+1}\} \neq \{n_{h'}^{l'}, n_{h'}^{l'+1}\}$.

LEMMA 41. *For any integer* $h = 1, \ldots, \frac{n-1}{2}$ *it holds that*

$$\{n_h^l | l = 0, \ldots, n-1\} = \{0, 1, \ldots, n-1\}.$$

Let c_i be the ith call of the sequence, let $C_i = \{j : j < i, p_i \cap p_j = \emptyset\}$ be the set of previous calls that are consistent with c_i, and let $I_i = \{j : j < i, p_i \cap p_j \neq \emptyset\}$ be the set of previous calls that are inconsistent with c_i.

With every pair of calls c_i, c_j, $i = 1, \ldots, j-1$, $j = 1, \ldots, k$, two values s_{ij} and s'_{ij} are associated, $s_{ij} \in \{0, \ldots, n-2\}$ and $s'_{ij} \in \{0. \ldots, n-1\}$.

The idea behind the construction presented in the remainder of this section is the following. The generic path p_i associated with call c_i crosses all vertices of set V in the sequence $v(n_i^0), \ldots, v(n_i^{n-1})$. (By Lemma 41 such a sequence spans the set V.) Consider two calls c_i and c_j, $i < j$, with associated values s_{ij} and s'_{ij}. By Lemma 41, there exist two integer values q and r such that $n_i^q = n_j^r = s_{ij}$. In our construction, path p_i always contains edges $(v(s_{ij}), v'(s'_{ij}))$ and $(v'(s'_{ij}), v(n_i^{q+1}))$. If $c_i \in C_j$, then path p_j must not share any edge with p_i. In this case vertex $v(s_{ij})$ is followed by $v(n_j^{r+1})$ in p_j. Otherwise, if $c_i \in I_j$, then path p_j contains edges $(v(s_{ij}), v'(s'_{ij}))$ and $(v'(s'_{ij}), v(n_j^{r+1}))$, the first of which is common with p_i.

To ensure that two consistent calls do not share any edge we use Lemma 40, which states that two integers between 0 and $n-1$ are adjacent only in one sequence; furthermore, the association of values s_{ij} and s'_{ij} with any pair of calls c_i, c_j is to guarantee that any edge between a vertex in V and a vertex in V', included in some path p_i and dedicated to the possible intersection with path p_j, does not appear in any other path.

Values s_{ij} and s'_{ij} are assigned to any pair of calls c_i, c_j, $i < j$, in order of increasing j, and for a fixed j in order of increasing i. Let $n_i^q = n_j^r = s_{ij}$.

Consider a pair x, y, $x < y \leq j$, for which the values s_{xy} and s'_{xy} have been assigned prior to the assignment for i, j. Let l and m be integers such that $n_x^l = n_y^m = s_{xy}$. The conditions on the assignment for c_i, c_j are as follows:

1. Assign value s_{ij} different from s_{xy} with $y = i$, $y = j$, or $x = i$.
2. Assign value s'_{ij} different from s'_{xy} if $\{s_{xy}, n_x^{l+1}, n_y^{m+1}\} \cap \{s_{ij}, n_i^{q+1}, n_j^{r+1}\} \neq \emptyset$ for any previously assigned pair xy.

LEMMA 42. *There exists a sequence of $k = \Omega(n)$ calls such that for every pair c_i, c_j, $i < j$, values s_{ij} and s'_{ij} can be assigned obeying conditions* 1 *and* 2.

Proof. We prove that there exists a value $k = \Omega(n)$ such that, in the worst case, at least one possibility to assign values s_{ij} and s'_{ij} is left for any pair of calls c_i, c_j with $i < j \leq k$.

Denote by x, y any pair considered before i, j in the designed order. Then either $x < y < j$ or $y = j$ and $x < i$.

Following condition 1, we can bound the number of pairs such that $y = i$, $y = j$, or $x = i$ by $(i-1) + (j-1) + (j-i) \leq 2(k-2)$. Therefore, if $n > 2(k-2)$, at least one possibility is left for assigning s_{ij}.

Following condition 2, we would like to bound the number of pairs c_x, c_y for which one of the three values in $\{s_{ij}, n_i^{q+1}, n_j^{r+1}\}$ appears in $\{s_{xy}, n_x^{l+1}, n_y^{m+1}\}$.

For fixed $h < k$, each of the values in $\{s_{ij}, n_i^{q+1}, n_j^{r+1}\}$ is equal to n_h^d for some integer d. By condition 1 there could be either one y for which $s_{hy} = n_h^d$ or one x such that $s_{xh} = n_h^d$. Similarly, there could be either one y for which $s_{hy} = n_h^{d-1}$ or one x such that $s_{xh} = n_h^{d-1}$. Thus we get that h can be involved in at most six pairs for which the above restriction holds.

Hence, if $n > 6(k-1)$, there is still a possibility left for assigning s'_{ij}.

The above observations show that by setting $k = n/6$, values s_{ij} and s'_{ij} can be assigned to each pair of calls c_i, c_j. ∎

In the following we formally define call c_i, $i = 1, \ldots, k$. The endpoints s_i, t_i of call c_i are

- $s_i = v(n_i^0)$,
- $t_i = v(n_i^{n-1})$.

Path p_i associated with call c_i contains the following set of edges:

1. $(v(n_i^l), v(n_i^{l+1}))$, for any $l \in \{1, \ldots, n-1\}$ such that
   (a) For all $j < i$, $n_i^l \neq s_{ji}$ and for all $j > i$, $n_i^l \neq s_{ij}$.
   (b) There exists $j < i$ such that $n_i^l = s_{ji}$ and $j \in C_i$.
2. $(v(n_i^l), v'(s'_{ij}))$ and $(v'(s'_{ij}), v(n_i^{l+1}))$, for any $l \in \{1, \ldots, n-1\}$ such that there exists $j > i$ for which $s_{ij} = n_i^l$.
3. $(v(n_i^l), v'(s'_{ji}))$ and $(v'(s'_{ji}), v(n_i^{l+1}))$, for any $l \in \{1, \ldots, n-1\}$ such that there exists $j < i$ for which $s_{ji} = n_i^l$ and $j \in I_i$.

LEMMA 43. *Let c_i, c_j, with $i < j$ be two calls in the sequence. p_i and p_j do not intersect if and only if $i \in C_j$.*

Proof. First we prove that, for any $i \in I_j$, the paths p_i and p_j intersect. Then the edge $(v(s_{ij}), v'(s'_{ij}))$ is in p_i by rule 2 and in p_j by rule 3.

We now turn to proving that if $c_i \in C_j$, then the two paths p_i and p_j do not intersect. Lemma 40 ensures that the edges between two vertices in V (rule 1) cannot be shared by two paths.

Edges in p_i connecting vertices of V to vertices of V' are either (by rule 2) of the form $(v(s_{ih}), v'(s'_{ih}))$ or $(v'(s'_{ih}), v(n_i^{d+1}))$ for some $h > i$ and d such that $s_{ih} = n_i^d$, or (by rule 3) $(v(s_{hi}), v'(s'_{hi}))$ or $(v'(s'_{hi}), v(n_i^{d+1}))$ for some $h < i$ and d such that

$s_{hi} = n_i^d$ (by rule 3). Thus, one endpoint of each such edge is a vertex $v'(s'_{xy})$, and the other one has index in $\{s_{xy}, n_x^{l+1}, n_y^{m+1}\}$ for a pair in which one of x, y is equal to i and $n_x^l = n_y^m = s_{xy}$. A similar statement holds for path p_j. By condition 2 of the assignment procedure for s'_{ij}, such edges cannot appear in more than one path, and hence the claim follows. □

We conclude with the following theorem.

THEOREM 44. *Any nonpreemptive randomized algorithm for on-line edge-disjoint fixed paths in a network of n nodes has competitive ratio* $\Omega(n)$.

Acknowledgments. The authors would like to thank Yossi Azar, Anna Karlin, and Alberto Marchetti-Spaccamela. We also thank Allan Borodin, Ran El-Yaniv, Haim Kaplan, Mario Szegedy, and two anonymous referees for very helpful comments and for helping us avoiding errors.

REFERENCES

[AAFLR96] B. AWERBUCH, Y. AZAR, A. FIAT, S. LEONARDI, AND A. ROSÉN, *On-line competitive algorithms for call admission in optical networks*, in Proceedings of the 4th Annual European Symposium on Algorithms, Barcelona, Spain, 1996, Lecture Notes in Comput. Sci. 1136, Springer, New York, pp. 431–444.

[AAFPW92] J. ASPNES, Y. AZAR, A. FIAT, S. PLOTKIN, AND O. WAARTS, *On-line load balancing with applications to machine scheduling and virtual circuit routing*, in Proceedings of the 33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, PA, 1992, IEEE Computer Society Press, Piscataway, NJ, 1992, pp. 164–173.

[AAP93] B. AWERBUCH, Y. AZAR, AND S. PLOTKIN, *Throughput competitive on-line routing*, in Proceedings of the 34th Annual Symposium on Foundations of Computer Science, Palo Alto, CA, 1993, IEEE Computer Society Press, Piscataway, NJ, 1993, pp. 32–40.

[AAPW94] B. AWERBUCH, Y. AZAR, S. PLOTKIN, AND O. WAARTS, *Competitive routing of virtual circuits with unknown duration*, in Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, 1994, SIAM, Philadelphia, 1994, pp. 321–327.

[ABC+94] A. AGGARWAL, A. BAR-NOY, D. COPPERSMITH, R. RAMASWAMI, B. SCHIEBER, AND M. SUDAN, *Efficient routing and scheduling algorithms for optical networks*, in Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, Arlington, VA 1994, SIAM, Philadelphia, 1994, pp. 412–423.

[ABFR94] B. AWERBUCH, Y. BARTAL, A. FIAT, AND A. ROSÉN, *Competitive non-preemptive call control*, in Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, Arlington, VA 1994, SIAM, Philadelphia, 1994, pp. 312–320.

[AGLR94] B. AWERBUCH, R. GAWLICK, F. T. LEIGHTON, AND Y. RABANI, *On-line admission control and circuit routing for high performance computing and communication*, in Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, 1994, IEEE Computer Society Press, Piscataway, NJ, 1994, pp. 412–423.

[AKPPW93] Y. AZAR, B. KALYANASUNDARAM, S. PLOTKIN, K. PRUHS, AND O. WAARTS, *Online load balancing of temporary tasks*, in Proceedings of the 3rd Workshop on Algorithms and Data Structures, Montreal, QC, 1993, Lecture Notes in Comput. Sci. 709, Springer, New York, 1993, pp. 119–130.

[BBKTW90] S. BEN-DAVID, A. BORODIN, R. M. KARP, G. TARDOS, AND A. WIDGERSON, *On the power of randomization in on-line algorithms*, in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, Baltimore, MD, 1990, pp. 379–386.

[BCK+95] A. BAR-NOY, R. CANETTI, S. KUTTEN, Y. MANSOUR, AND B. SCHIEBER, *Bandwidth allocation with preemption*, in Proceedings of the 27th Annual ACM Symposium on Theory of Computing, Las Vegas, 1995, ACM, New York, pp. 616–625.

[BE98] A. BORODIN AND R. EL-YANIV, *On-Line Computation and Competitive Analysis*, Cambridge University Press, Cambridge, UK, 1998.

[BE97] A. BORODIN AND R. EL-YANIV, *On randomization in on-line computation*, Inform. Comput., 150 (1999), pp. 244–267.

[BFL96] Y. Bartal, A. Fiat, and S. Leonardi, *Lower bounds for on-line graph problems with application to on-line circuit and optical routing*, in Proceedings of the 28th ACM Symposium on Theory of Computing, Philadelphia, 1996, ACM, New York, pp. 531–540; also available online at http://www.dis.uniroma1.it/∼leon/bfl.ps.

[BH92] R. B. Boppana and M. M. Halldorson, *Approximating maximum independent set by excluding subgraphs*, BIT, 32 (1992), pp. 180–196.

[BH93] R. A. Barry and P. A. Humblet, *On the number of wavelengths and switches in all optical networks*, IEEE Trans. Communication, 42 (1994), pp. 583–591.

[BL97] Y. Bartal and S. Leonardi, *On-line routing in all-optical networks*, in Proceedings of the 24th International Colloqium on Automata, Languages and Programming, Bologna, Italy, 1997, Lecture Notes in Comput. Sci. 1256, Springer, New York, 1997, pp. 516–526.

[CI95] R. Canetti and S. Irani, *On the power of preemption in randomized scheduling*, in Proceedings of the 27th Annual ACM Symposium on Theory of Computing, Las Vegas, 1995, ACM, New York, pp. 606–615.

[GG92] J. Garay and I. S. Gopal, *Call preemption in communications networks*, in Proceedings of INFOCOM 92, Florence, Italy, 1992, IEEE Computer Society Press, Piscataway, NJ, 1992, pp. 2332–2342.

[GGKMY93] J. Garay, I. S. Gopal, S. Kutten, Y. Mansour, and M. Yung, *Efficient on-line call control algorithms*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing, 1993, pp. 285–293.

[Hal94] M. M. Halldórsson, *Approximations of weighted independent set and hereditary subset problems*, J. Graph Algorithms Appl., 4 (2000), pp. 1–16.

[HS92] M. M. Halldórsson and M. Szegedy, *Lower bounds for on-line graph coloring*, in Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms, Orlando, FL, 1992, SIAM, Philadelphia, 1992, pp. 211–216.

[I90] S. Irani, *Coloring inductive graphs on-line*, in Proceedings of the 31st Annual Symposium on Foundations of Computer Science, St. Louis, MO, 1990, IEEE Computer Society Press, Piscataway, NJ, 1990, pp. 470–479.

[KS98] H. Kaplan and M. Szegedy, *On-line complexity of monotone set systems*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, 1999, SIAM, Philadelphia, 1999, pp. 507–516.

[KS95] V. Kapoulas and P. Spirakis, *Randomized competitive algorithms for admission control in general networks*, in Proceedings of the 14th Annual ACM Symposium on Principles of Distributed Computing, Ottawa, ON, 1995, ACM, New York, 1995, p. 253.

[KT95] J. Kleinberg and E. Tardos, *Disjoint paths in densely embedded graphs*, in Proceedings of the 36th IEEE Annual Symposium on Foundations of Computer Science, Milwaukee, WI, 1995, IEEE Press, Piscataway, NJ, pp. 52–61.

[KVV90] R. M. Karp, U. V. Vazirani, and V. V. Vazirani, *An optimal algorithm for on-line bipartite matching*, in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, Baltimore, MD, 1990, ACM, New York, 1990, pp. 352–358.

[LMPR98] S. Leonardi, A. Marchetti-Spaccamela, A. Presciutti, and A. Rosén, *On-line randomized call control revisited*, in Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, 1998, SIAM, Philadelphia, 1998, pp. 323–332.

[LY93a] C. Lund and M. Yannakakis, *The approximation of maximum subgraph problems*, in Proceedings of the 20th International Colloquium on Automata, Languages and Programming, Lund, Sweden, 1993, Lecture Notes in Comput. Sci. 700, Springer, New York, 1993, pp. 40–51.

[LY93b] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, in Proceedings of the 25th ACM Symposium on Theory of Computing, 1993, ACM, New York, pp. 40–51.

[P92] R. K. Pankay, *Architectures for Linear Light-Wave Networks*, Ph.D. thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, 1992.

[RU94] P. Raghavan and U. Upfal, *Efficient routing in all-optical networks*, in Proceedings of the 26th Annual Symposium on Theory of Computing, Montréal, 1994, ACM, New York, pp. 133–143.

[ST85] D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, Comm. ACM, 28 (1985), pp. 202–208.

[V90] S. Vishwanathan, *Randomized on-line graph coloring*, in Proceedings of the 31st IEEE Annual Symposium on Foundations of Computer Science, St. Louis, MO, 1990, IEEE Computer Society Press, Piscataway, NJ, 1990, pp. 464–469.

[Y77] A. C. Yao, *Probabilistic computations: Towards a unified measure of complexity*, in Proceedings of the 18th Annual Symposium on Foundations of Computer Science, Providence, RI, 1977, IEEE Computer Society Press, Piscataway, NJ, 1977, pp. 222–227.

SIAM J. COMPUT.
Vol. 36, No. 2, pp. 394–410

EFFICIENT BUNDLE SORTING*

YOSSI MATIAS†, ERAN SEGAL‡, AND JEFFREY SCOTT VITTER§

Abstract. Many data sets to be sorted consist of a limited number of distinct keys. Sorting such data sets can be thought of as bundling together identical keys and having the bundles placed in order; we therefore denote this as *bundle sorting*. We describe an efficient algorithm for bundle sorting in external memory, which requires at most $c(N/B)\log_{M/B} k$ disk accesses, where N is the number of keys, M is the size of internal memory, k is the number of distinct keys, B is the transfer block size, and $2 < c < 4$. For moderately sized k, this bound circumvents the $\Theta((N/B)\log_{M/B}(N/B))$ I/O lower bound known for general sorting. We show that our algorithm is optimal by proving a matching lower bound for bundle sorting. The improved running time of bundle sorting over general sorting can be significant in practice, as demonstrated by experimentation. An important feature of the new algorithm is that it is executed "in-place," requiring no additional disk space.

Key words. sorting, external memory, bundle sorting, algorithms

AMS subject classification. 68W01

DOI. 10.1137/S0097539704446554

1. Introduction.

Sorting is a frequent operation in many applications. It is used not only to produce sorted output, but also in many sort-based algorithms such as grouping with aggregation, duplicate removal, and sort-merge join, as well as set operations including union, intersect, and except [Gra93, IBM95]. In this paper, we identify a common external memory sorting problem, present an algorithm to solve it while circumventing the lower bound for general sorting for this problem, prove a matching lower bound for our algorithm, and demonstrate the improved performance through experiments.

External merge sort is the most commonly used algorithm for large-scale sorting. It has a run formation phase, which produces sorted runs, and a merge phase, which merges the runs into sorted output. Its running time, as in most external memory algorithms, is dominated by the number of input/outputs (I/Os) performed, which is $O((N/B)\log_{M/B}(N/B))$, where N is the number of keys, M is the size of internal memory, and B is the transfer block size. It was shown in [AV88] (see also [Vit99]) that there is a matching lower bound within a constant factor.

The number of passes over the sequence performed by sorting algorithms is $\lceil \log_{M/B}(N/B) \rceil$ in the worst case. When the available memory is large enough compared to the size of the sequence, the sorting can be performed in one or two passes over the sequence (see [ADADC+97] and references therein). However, there are many

*Received by the editors November 22, 2004; accepted for publication (in revised form) February 24, 2006; published electronically June 23, 2006. A preliminary version of this paper was presented at the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms [MSV00].

http://www.siam.org/journals/sicomp/36-2/44655.html

†School of Computer Science, Tel-Aviv University, Tel-Aviv 69978 Israel (matias@cs.tau.ac.il). This author's work was supported in part by an Alon Fellowship, by the Israel Science Foundation founded by the Academy of Sciences and Humanities, and by the Israeli Ministry of Science.

‡Department of Computer Science, Stanford University, Stanford, CA 94305 (eran@cs.stanford.edu). Much of this author's work was done while the author was at Tel-Aviv University.

§Department of Computer Science, Purdue University, West Lafayette, IN 47907-2066 (jsv@purdue.edu). Much of this author's work was done while the author was on sabbatical at I.N.R.I.A. in Sophia Antipolis, France, and was supported in part by Army Research Office MURI grant DAAH04–96–1–0013 and DAAD19–01–1–0725, and by the National Science Foundation research grant CCR–9522047.

settings in which the available memory is moderate, at best. For instance, in multithreading and multiuser environments, an application, process, or thread which may execute a sorting program might be allocated only a small fraction of the machine memory. Such settings may be relevant to anything from low-end servers to high-end decision support systems. For moderate size memory, $\log_{M/B}(N/B)$ may become large enough to imply a significant number of passes over the data. As an example, consider the setting $N = 256$ GB, $B = 128$ kB, and $M = 16$ MB. Then we have $\log_{M/B}(N/B) = 3$, and the number of I/Os per disk block required by merge sort is at least 6. For smaller memory allocations, the I/O costs will be even greater.

Our contributions. Data sets that are to be sorted often consist of keys taken from a bounded universe. This fact is well exploited in main memory algorithms such as counting sort and radix sort, which are substantially more efficient than general sort. In this paper we consider the extent to which a limit, k, on the number of distinct keys can be exploited to obtain more effective sorting algorithms in external memory on massive data sets, where the attention is primarily given to the number of I/Os. Sorting such data sets can be thought of as bundling together identical keys and having the bundles placed in order; we therefore denote this as *bundle sorting*. It is similar to partial sorting, which was identified by Knuth [Knu73] as an important problem. While many algorithms are given for partial sorting in main memory, to the best of our knowledge, there exist no efficient algorithms for solving the problem in external memory. As we shall see, bundle sorting can be substantially more efficient than general sorting.

A key feature of bundle sorting is that the number of I/Os performed per disk block depends solely on the number k of distinct keys. Hence, in sorting applications in which the number of distinct keys is constant, the number of I/Os performed per disk block remains constant for any data set size. In contrast, merge sort or other general sorting algorithms will perform more I/Os per disk block as the size of the data set increases. In settings in which the size of the data set is large this can be significant. In the example given earlier, six I/Os per data block are needed to sort in the worst case. For some constant $k < 100$, bundle sorting performs only two I/Os per disk block, and for some constant $k < 10000$, only four I/Os per disk block, regardless of the size of the data set.

The algorithm that we present requires at most $3 \log_{M/B} k$ passes over the sequence. It performs the sorting in-place, meaning that the input data set can be permuted as needed without using any additional working space in external memory. When the number k of distinct keys is less than N/B, our bundle sorting algorithm circumvents the lower bound for general sorting. The lower bound for general sorting is derived by a lower bound for permuting the input sequence, which is an easier problem than general sorting. In contrast to general sorting, bundle sorting is not harder than permuting; rather than requiring that a particular key be moved to a specific location, it is required that the key be moved to a location within a specified range, which belongs to its bundle. This so-called bundle-permutation consists of a set of permutations, and implementing bundle-permutation can be done more efficiently than implementing a particular permutation.

For cases in which $k \ll N/B$, the improvement in the running time of bundle sorting over general sorting algorithms can be significant in practical sorting settings, as supported by our experimentation done on U.S. Census data and on synthetic data. In fact, the number of passes over the sequence executed by our algorithm does not depend at all on the size of the sequence, in contrast to general sorting algorithms.

To complement the algorithmic component, we prove a matching lower bound

for bundle sorting. In particular, we show that the number of I/Os required in the worst case to sort N keys consisting of k distinct key values is $\Omega((N/B)\log_{M/B} k)$. This lower bound is realized by proving lower bounds on two problems that are both easier than bundle sorting, and the combination of the lower bounds gives the desired result. The first special case is bundle-permutation, and the second is a type of matrix transposition. Bundle-permutation is the special case of bundle sorting in which we know the distribution of key values beforehand, and thus it is easier than bundle sorting for much the same reason that permuting is easier than general sorting. The other special case of bundle sorting is a type of matrix transposition, in which we transpose a $k \times N/k$ matrix, but the final order of the elements in each row is not important. This problem is a special case of bundle sorting of N keys consisting of exactly N/k records for each of k different keys and is thus easier than bundle sorting. Interestingly, these two problems, when combined, capture the difficulty of bundle sorting.

Our bundle sorting algorithm is based on a simple observation: If the available memory, M, is at least kB, then we can sort the data in three passes over the sequence, as follows. In the first pass, we count the size of each bundle. After this pass we know the range of blocks in which each bundle will reside upon termination of the bundle sorting. The first block from each such range is loaded to main memory. The loaded blocks are scanned concurrently, while swapping keys so that each block is filled only with keys belonging to its bundle. Whenever a block is fully scanned (i.e., it contains only keys belonging to its bundle), it is written back to disk, and the next block in its range is loaded. In this phase, each block is loaded exactly once (except for at most k blocks in which the ranges begin), and the total number of accesses over the input sequence in the entire algorithm is hence 3. Whenever memory is insubstantial to hold the k blocks in memory, we group bundles together into M/B superbundles, implementing the algorithm to sort the superbundles to M/B subsequences and reiterate within each subsequence, incurring a total of $\log_{M/B} k$ iterations over the sequence to complete the bundle sorting.

There are many applications and settings in which bundle sorting may be applied, resulting in a significant speed-up in performance. For instance, any application that requires partial sorting or partitioning of a data set into value independent buckets can take advantage of bundle sorting since the number of buckets (k in bundle sorting) is small, thus making bundle sorting very appealing. Another example would be *accelerating sort join computation for suitable data sets*: Consider a join operation between two large relations, each having a moderate number of distinct keys; then our bundle sorting algorithm can be used in a sort join computation, with performance improvement over the use of general sort algorithm.

Finally, we consider a more performance-sensitive model that, rather than just counting the number of I/Os as a measurement for performance, differentiates between a sequential I/O and a random I/O and assigns a reduced cost for sequential I/Os. We study the tradeoffs that occur when we apply bundle sorting in this model, and show a simple adaptation of bundle sorting that results in an optimal performance. In this sense, we also present a slightly different algorithm for bundle sorting, which is more suitable for sequential I/Os.

The rest of the paper is organized as follows. In section 2 we explore related work. In section 3 we describe the external memory model in which we will analyze our algorithm and prove the lower bound. Section 4 presents our algorithm for bundle sorting along with the performance analysis. In section 5 we prove the lower bound for external bundle sorting. In section 6 we consider a more performance-sensitive

model, which takes into account a reduced cost for sequential I/Os and shows the modifications in our bundle sorting algorithm required to achieve an optimal algorithm in that model. Section 7 describes the experiments we conducted, and section 8 is our conclusions.

2. Related work. External memory sorting is an extensively researched area. Many efficient in-memory sorting algorithms have been adapted for sorting in external memory, such as merge sort, and much of the recent research in external memory sorting has been dedicated to improving the run time performance. Over the years, numerous authors have reported the performance of their sorting algorithms and implementations (cf. [Aga96, BBW86, BGK90]). We note a recent paper [ADADC+97] that shows external sorting of 6 GB of data in under one minute on a network of workstations. For the problem of bundle sorting where $k < N/B$ we note that our algorithm will reduce the number of I/Os that all these algorithms perform and hence can be utilized in benchmarks. We also consider a more performance-sensitive model of external memory, in which rather than just counting the I/Os for determining the performance, there is a reduced cost for sequential I/Os compared to random access I/Os. We study the tradeoffs there, and show the adaptation in our bundle sorting algorithm to arrive at an optimal algorithm in that model. We also note that another recent paper [ZL98] shows in detail how to improve the merge phase of the external merge sort algorithm, a phase that is completely avoided by using our in-place algorithm.

In the general framework of external memory algorithms, Aggarwal and Vitter showed a lower bound of $\Omega((N/B)\log_{M/B}(N/B))$ on the number of I/Os needed in the worst case for sorting [AV88, Vit99]. In contrast, since our algorithm relies on the number k of distinct keys for its performance, we are able to circumvent this lower bound when $k \ll N/B$. Moreover, we prove a matching lower bound for bundle sorting, which shows that our algorithm is optimal.

Finally, sorting is used not only to produce sorted output, but also in many sort-based algorithms such as grouping with aggregation, duplicate removal, and sort-merge join, as well as set operations including union, intersect, and except [Gra93, IBM95]. In many of these cases the number of distinct keys is relatively small, and hence bundle sorting can be used for improved performance. We identify important applications for bundle sorting, but note that since sorting is such a common procedure, there are probably many more applications for bundle sorting that we did not consider.

3. External memory model. In our main bundle sorting algorithm and in the lower bound that we prove, we use the external memory model from Aggarwal and Vitter [AV88] (see also [Vit99]). The model is as follows. We assume that there is a single central processing unit, and we model secondary storage as a generalized random-access magnetic disk. (For completeness, the model is also extended to the case in which the disk has some parallel capabilities.) The parameters are

$$\begin{aligned} N &= \#\text{ records to sort,} \\ M &= \#\text{ records that can fit into internal memory,} \\ B &= \#\text{ records transferred in a single block,} \\ D &= \#\text{ blocks that can be transferred concurrently,} \end{aligned}$$

where $1 \le B \le M/2$, $M < N$, and $1 \le D \le \lfloor M/B \rfloor$. For brevity we consider only the case of $D = 1$, which corresponds to a single conventional disk.

The parameters N, M, and B are referred to as the *file size*, *memory size*, and *transfer block size*, respectively. Each block transfer is allowed to access any contiguous group of B records on the disk. We will consider the case where $D = 1$, meaning that there is no disk parallelism. Performance in this model is measured by the number of I/O accesses performed where the cost of all I/Os is identical. In section 6 we consider a more performance-sensitive model in which we differentiate between costs of sequential and random-access I/Os and assign a reduced cost for sequential I/Os.

4. External bundle sorting algorithm. In this section we present our bundle sorting algorithm, which sorts in-place a sequence that resides on disk and contains k distinct keys. We start by defining the bundle sorting problem:

Input: *A sequence of keys* $\{a_1, a_2, \ldots, a_n\}$ *from an ordered universe* U *of size* k.

Output: *A permutation* $\{a'_1, a'_2, \ldots, a'_n\}$ *of the input sequence such that* $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

In our algorithm, it will be easy, and with negligible overhead, to compute and use an order-preserving mapping from U to $\{1, \ldots, k\}$; we discuss the implementation details of this function in section 4.2. This enables us to consider the problem at hand as an integer sorting problem in which the keys are taken from $\{1, \ldots, k\}$. Hence, we assume that $U = \{1, \ldots, k\}$.

We use the external memory model from section 3, where performance is determined by the number of I/Os performed. Our goal is to minimize the number of disk I/Os. In section 6 we consider a more performance-sensitive model in which rather than simply counting I/Os as a measurement of performance, we differentiate between a sequential I/O and a random I/O and assign a reduced cost to sequential I/Os. We show the necessary modifications to the bundle sorting presented in this section required to achieve an optimum in that model.

4.1. $\{1, \ldots, k\}$ integer sorting. We start by presenting "one-pass sorting"—a procedure that sorts a sequence into $\mu = \lfloor M/B \rfloor$ distinct keys. It will be used by our bundle sorting algorithm to perform one iteration that sorts a chunk of data blocks into μ ranges of keys.

The general idea is this: Initially we perform one pass on the sequence, loading one block of size B at a time, in which we count the number of appearances of each of the μ distinct keys in the sequence. Next, we keep in memory μ blocks and a pointer for each block, where each block is of size B. Using the count pass, we initialize the μ blocks, where the ith block is loaded from the exact location in the sequence where keys of type i will start residing in the sorted sequence. We set each block pointer to point to the first key in its block. When the algorithm runs, the ith block pointer is advanced as long as it encounters keys of type i. When a block pointer is "stuck" on a key of type j, it waits for the jth block pointer until it too is stuck (this will happen since a block pointer yields only to keys of its block), in which case a swap is performed and at least one of the two block pointers may continue to advance. When any of the μ block pointers reaches the end of its block, we write that block back to disk to the exact location from which it was loaded, and load the next contiguous block from disk into memory (and of course set its block pointer again to the first key in the block). We finish with each of the μ blocks upon crossing the boundaries of the next adjacent block. The algorithm terminates when all blocks are done with. The following is a pseudocode of the algorithm. See also Figure 1.

ONE-PASS SORTING ALGORITHM.

procedure one-pass-sort $(sequence, k, M, B)$

for $i = 0$ to $\lfloor M/B \rfloor$

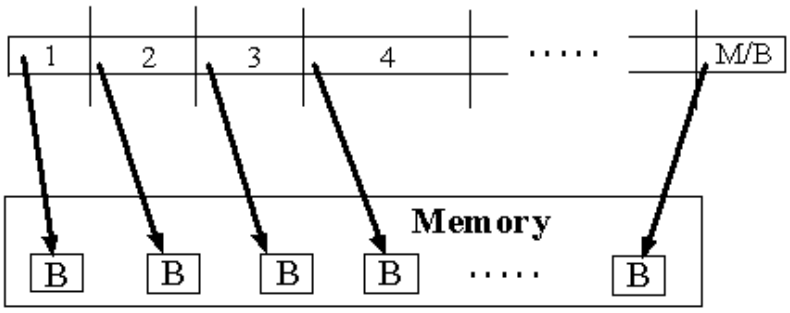


FIG. 1. *Initialization of the M/B blocks in the One-pass sorting algorithm. After the counting pass, we know where the sorted blocks reside, and load blocks from these locations. Swaps are performed in memory. When any of the blocks is full, we write it to disk to the location from which it was loaded, and load the next block from disk.*

```
        current_block = ith block of sequence
        for j = 0 to B of current_block
            Count[sequence[i · B + j]]++
    Count[k + 1] = M
    for i = 1 to k
        μ[i] = block at position Count[i] from sequence
        μ_block_pointer[i] = 0
        μ_global_pointer[i] = Count[i]
    blocks_done = 0
    while blocks_done < k
        for i = 1 to k
            if μ_global_pointer[i] < Count[i + 1]
                while μ_block_pointer[i] < B and μ[i][μ_block_pointer[i]] = i
                    μ_block_pointer[i]++
                    μ_global_pointer[i]++
                stuck[i] = (μ_block_pointer[i] < B and μ_global_pointer[i] < Count[i + 1])
                stuck_value = μ[i][μ_block_pointer[i]]
                if stuck[i] and stuck[stuck_value] then
                    swap μ[i][μ_block_pointer[i]] with
                    μ[stuck_value][μ_block_pointer[stuck_value]]
                if μ_block_pointer[i] = B
                    Write μ[i] as a block to the location from which it was loaded
                    μ[i] = block at position μ_global_pointer[i] from sequence
                    μ_block_pointer[i] = 0
                if μ_global_pointer[i] = Count[i + 1]
                    blocks_done++;
```

LEMMA 4.1. *Let S be a sequence of N keys from $\{1, \ldots, \mu\}$, let B be the transfer block size, and let M be the available memory such that $M \geq \mu B$. Then the sequence can be sorted in-place using the procedure "one-pass sorting" with a total of $\lceil 3N/B + 2M/B \rceil$ I/Os.*

Proof. We first show that the algorithm indeed sorts the input sequence. The algorithm allocates one pointer in memory for each of the μ distinct keys, and the ith

such pointer writes only contiguous blocks of records whose keys consist solely of the ith key. Thus, to show that the sequence is sorted by "one-pass sorting," it suffices to show that the algorithm terminates and that, upon termination, the ith pointer writes its blocks in a physical location that precedes the blocks written by any j pointer for $j > i$. The ordering between the pointers is ensured by setting the contiguous block of the ith pointer to write to the exact location where keys of its type should reside in the sorted sequence. This location is derived from the first pass in which we count the number of appearances of each of the μ distinct keys. Termination is guaranteed, since at each step at least one of the pointers encounters keys of its type, or a swap will be performed and at least one of the pointers can proceed. Note that such a swap will always be possible, since if the ith pointer is "stuck" on a key of type j, then the jth pointer will necessarily get stuck at some step. Since at each step one of the keys is written and there are N keys, the algorithm will terminate.

For computing the number of I/Os, note that the first counting pass reads each block once and thus requires $\lceil N/B \rceil$ I/Os. All the μ pointers combined read and write each block once, adding another $\lceil 2N/B \rceil$ I/Os. Finally, if the number of appearances of each distinct key is not an exact multiple of B, then every pair of consecutive pointers may overlap by one block at the boundaries, thus requiring an additional $\lceil 2M/B \rceil$ I/Os. □

We now present the complete integer sorting algorithm. We assume that the sequence contains keys in the range $1, \ldots, k$, where k is the number of distinct keys. In section 4.2 we discuss the adaptation needed if the k distinct keys are not from this integer range. We use the above one-pass sorting procedure. The general idea is this: We initially perform one sorting iteration in which we sort the sequence into $k' = \lfloor M/B \rfloor$ keys. We select a mapping function f such that for all $1 \le i \le k$ we have $f(i) = \lceil ik'/k \rceil$, and we apply f to every key when the key is examined. This ensures that we are actually in the range of $1, \ldots, k'$. Moreover, it will create sorted buckets on disk such that the number of distinct keys in each of the buckets is roughly k/k'. We repeat this procedure recursively for each of the sorted blocks obtained in this iteration until the whole sequence is sorted. Each sorting iteration is done by calling the procedure for one-pass sorting. We give a pseudocode of the algorithm below, followed by an analysis of its performance.

THE INTEGER SORTING ALGORITHM.

```
procedure sort (sequence, k, M, B)
    k' = max(⌊M/B⌋, 2)      // compute k'
    if (k > 2) then
        call one-pass sorting (sequence, k', M, B)
        for i = 1 to k'
            bucket = the ith bucket sorted
            call sort (bucket, ⌈k/k'⌉, M, B)
```

THEOREM 4.1. *Let S be a sequence of N keys from $\{1, \ldots, k\}$, let M be the available memory, and let B be the transfer block size. A sequence residing on disk can be sorted in-place using the bundle sorting algorithm, while the number of I/Os is at most*

$$\left\lceil \frac{3N}{B} \log_{\lfloor M/B \rfloor} k \right\rceil + 4k \left\lfloor \frac{M}{B} \right\rfloor.$$

Proof. We first show that bundle sorting results in a sorting of the input sequence. Since we map each key i to $\lceil ik'/k \rceil$, it follows from the correctness of the one-pass

sorting that, after the first call to one-pass sorting, the sequence will be sorted such that for all i, keys in the range $\{\lceil (i-1)k'/k \rceil + 1, \ldots, \lceil ik'/k \rceil\}$ precede all keys greater than $\lceil ik'/k \rceil$. Each of the resulting range of keys is then recursively sorted. After at most $\log_{\lfloor M/B \rfloor} k$ recursive iterations, the number of distinct keys will be less than k', in which case the one-pass sorting will result in a full sorting of the sequence.

For the number of I/Os, we can view the bundle sorting algorithm as proceeding in levels of recursion, where at the first level of recursion bundle sorting is applied once, at the second level it is applied k' times, and at the ith level it is applied k'^{i-1} times. The total number of levels of recursion is $\log_{\lfloor M/B \rfloor} k$. Even though at the ith recursive level bundle sorting is applied k'^{i-1} times, each application is given a disjoint sequence shorter than N as input, and all applications of bundle sorting at the same recursive level cover the N input sequence exactly once. Thus, the counting pass of all applications at the same recursive level will still require $\lceil N/B \rceil$ I/Os, and all such applications will result in a read and write of each block, incurring an additional $\lceil 2N/B \rceil$ I/Os. Finally, since in general the number of distinct keys will not be a multiple of B, there might be an overlap of at most one block between every pair of consecutive pointers in one-pass sorting. Thus, we require an additional $2\lfloor M/B \rfloor$ I/Os for each application of one-pass sorting. One-pass sorting is called once for the first level of recursion, k' for the second level, and k'^{i-1} for the ith level, and thus the total number of times that one-pass sorting is called is $\frac{k'^{1+log_{k'} k}-1}{k'-1} = \frac{k'k-1}{k'-1} \leq 2k$. Hence, we add an additional $4k\lfloor M/B \rfloor$ I/Os, which results in the desired bound on the number of I/Os. ∎

4.2. General bundle sorting. In section 4.1 we assumed that the input was in the range $1, \ldots, k$, where k is the number of distinct keys in the sequence. We now discuss how to construct a mapping function when the input is not in this range.

In the simple case where the input is from a universe that is not ordered (i.e., the sorting is done just to cluster keys together), we can simply select any universal hash function as our mapping function. This ensures that the number of distinct keys that will be distributed to each bucket is fairly equal and that our algorithm performs without any loss of performance.

For the general case we assume that the input is from an ordered universe U and consists of k distinct keys. We show how to construct a mapping function from U to $1, \ldots, k$. More specifically, we need a way to map the keys into the range $[1, M/B]$ at every application of the one-pass sorting procedure. A solution to this mapping is to build an M/B-ary tree, whose leaves are the k distinct keys in sorted order and each internal node stores the minimum and the maximum values of its M/B children. Each application of one-pass sorting in integer sorting corresponds to an internal node in the tree (starting from the root) along with its children, and so the tree provides the appropriate mapping. This is because in each run of one-pass sorting the keys are within the range of the minimum and maximum values stored in the corresponding internal node, and the mapping into $1, \ldots, M/B$ is done according to the ranges of the internal node's children.

Constructing the sorted leaves can be done via count sort, in which we are given a sequence of size N with k distinct keys and we need to produce a sorted list of the k distinct keys and their counts. An easy way to do count sort is via merge sort, in which identical keys are combined together (and their counts summed) whenever they appear together. In each merge sort pass, the output run will never be longer than k/B blocks. Initially, the runs contain at most M/B blocks. After $\log_{M/B}(k/B)$ passes, the runs will be of length at most k/B blocks, and after that point the number

of runs decreases geometrically, and the running time is thus linear in the number of I/Os. The rest of the tree can be computed in at most one extra scan of the leaves-array and lower order postprocessing. We can show the following.

LEMMA 4.2 (see [WVI98]). *A sequence of size N consisting of k distinct keys can be count-sorted, using a memory of size M and block transfer size B, within an I/O bound of*

$$\frac{2N}{B}\log_{M/B}\frac{k}{B}.$$

An interesting observation is that by adding a count to each leaf representing its frequency in the sequence, and a count to each internal node which is the sum of the counts of its children, we can eliminate the count phase of the one-pass sorting procedure in the integer sorting algorithm. Thus, the general bundle sorting algorithm is as follows. Initially, we use count sort and produce the tree. We now traverse the tree, and on each internal node we call one-pass sorting, where the mapping function is simply the ranges of values of the node's M/B children. By combining Theorem 4.1 and Lemma 4.2 we can prove the bound for general bundle sorting.

THEOREM 4.2. *Let S be a sequence of size N, which consists of k distinct keys; let M be the available memory; and let B be the transfer block size. Then we can in-place sort S using the bundle sorting algorithm, while the number of I/Os is at most*

$$\frac{2N}{B}\left(\log_{\lfloor M/B\rfloor} k + \log_{\lfloor M/B\rfloor}\frac{k}{B}\right).$$

For all $k < B^2$, this bound would be better than the bound for integer sorting. Note that we can traverse the tree in either BFS (breadth first search) or DFS (depth first search). If we choose BFS, the sorting will be done concurrently, and we get an algorithm that gradually refines the sort. If we choose DFS, we get fully sorted items quickly, while the rest of the items are left completely unsorted. The overhead we incur by using the mapping will be in memory, where we now have to perform a search over the M/B children of the internal node that we are traversing in order to determine the mapping of each key into the range $1,\ldots,M/B$. Using a simple binary search over the ranges, the overhead will be an additional $\log_2(M/B)$ memory operations per key.

5. Lower bound for external bundle sorting. In this section we present a lower bound for the I/O complexity of bundle sorting. We let k be the number of distinct keys, M be the available memory, N be the size of the sequence, and B be the transfer block size. We then differentiate between the following two cases:

1. $k/B = B^{\Omega(1)}$ or $M/B = B^{\Omega(1)}$. We prove the lower bound for this case by proving a lower bound on bundle permutation, which is an easier problem than bundle sorting.
2. $k/B = B^{o(1)}$ and $M/B = B^{o(1)}$. We prove the lower bound for this case by proving a lower bound on a special case of matrix transposition, which is easier than bundle sorting.

Lower bound using bundle-permutation. We assume that $k/B = B^{\Omega(1)}$ or $M/B = B^{\Omega(1)}$ and use a similar approach as in the lower bound for general sorting of Aggarwal and Vitter [AV88] (see also [Vit99]). They proved the lower bound on the problem of computing an arbitrary permutation, which is easier than sorting. Bundle sorting is not necessarily harder than computing an arbitrary permutation, since

the output sequence may consist of one out of a set of permutations, denoted as a bundle-permutation. A *bundle-permutation* is an equivalence class of permutations, where two permutations can be in the same class if one can be obtained from the other by permuting within bundles. Computing a permutation from an arbitrary bundle-permutation, which we will refer to as the bundle-permutation problem, is easier than bundle sorting.

LEMMA 5.1. *Under the assumption that $k/B = B^{\Omega(1)}$ or $M/B = B^{\Omega(1)}$, the number of I/Os required in the worst case for sorting N data items of k distinct keys, using a memory of size M and block transfer size B, is*

$$\Omega\left(\frac{N}{B}\log_{M/B} k\right).$$

Proof. Given a sequence of N data items consisting of k bundles of sizes $\alpha_1, \alpha_2, \ldots, \alpha_k$, the number of distinct bundle-permutations is

$$\frac{N!}{\alpha_1! \cdot \alpha_2! \cdot \cdots \cdot \alpha_k!} \geq \frac{N!}{\left(\left(\frac{N}{k}\right)!\right)^k};$$

the inequality is obtained using a convexity argument.

For the bundle-permutation problem, for each $t \geq 0$ we measure the number of distinct orderings that are realizable by at least one sequence of t I/Os. The value of t for which the number of distinct orderings first exceeds the minimum orderings needed to be considered is a lower bound on the worst-case number of I/Os needed for the bundle-permutation problem and thus on the bundle sorting on disks.

Initially, the number of different permutations defined is 1. We consider the effect of an output operation. There can be at most $N/B + t - 1$ full blocks before the tth output, and hence the tth output changes the number of permutations generated by at most a multiplicative factor of $N/B+t$, which can be bounded trivially by $N \log N$.

For an input operation, we consider a block of B records input from a specific block on disk. The B data keys in the block can intersperse among the M keys in the internal memory in at most $\binom{M}{B}$ ways, so that the number of realizable orderings increases by a factor of $\binom{M}{B}$. If the block has never before resided in internal memory, the number of realizable orderings increases by an extra factor of $B!$, since the keys in the block can be permuted among themselves. This extra contribution can occur only once for each of the N/B original blocks. Hence, the number of distinct orderings that can be realized by some sequence of t I/Os is at most

$$(B!)^{N/B}\left(N \log N \binom{M}{B}\right)^t.$$

We want to find the minimum t for which the number of realizable orderings exceeds the minimum orderings required. Hence we have

$$(B!)^{N/B}\left(N \log N \binom{M}{B}\right)^t \geq \frac{N!}{\left(\left(\frac{N}{k}\right)!\right)^k}.$$

Taking the logarithm and applying Stirling's formula, with some algebraic manipulations, we get

$$t\left(\log N + B \log \frac{M}{B}\right) = \Omega\left(N \log \frac{k}{B}\right).$$

By solving for t, we get

$$\text{number of IOs} = \Omega\left(\frac{N}{B}\log_{M/B}\frac{k}{B}\right).$$

Recall that we assume either $k/B = B^{\Omega(1)}$ or $M/B = B^{\Omega(1)}$. In either case, it is easy to see that $\log_{M/B}(k/B) = \Theta(\log_{M/B} k)$, which gives us the desired bound. □

Lower bound using a special case of matrix transposition. We now assume that $k/B = B^{o(1)}$ and $M/B = B^{o(1)}$ (the case not handled earlier) and prove a lower bound on a special case of matrix transposition, which is easier than bundle sorting. Our proof proceeds under the normal assumption that the records are treated indivisibly and that no compression of any sort is utilized.

LEMMA 5.2. *Under the assumption that $k/B = B^{o(1)}$ and $M/B = B^{o(1)}$, the number of I/Os required in the worst case for sorting N data items of k distinct keys, using a memory of size M block transfer size B, is*

$$\Omega\left(\frac{N}{B}\log_{M/B} k\right).$$

Proof. Consider the problem of transposing a $k \times N/k$ matrix, in which the final order of the elements in each row is not important. More specifically, let us assume that the elements of the matrix are originally in column-major order. The problem is to convert the matrix into row-major order, but the place in a row to which the element goes can be arbitrary, as long as it is transferred to the proper row. Each element that ends up in row i can be thought of as having the same key i. This problem is a special case of sorting N keys consisting of exactly N/k records for each of the k distinct keys. Hence, this problem is easier than bundle sorting. We now prove a lower bound for this problem of

$$\Omega\left(\frac{N}{B}\log_{M/B}\min(k, B)\right)$$

I/Os. Under our assumption that $k/B = B^{o(1)}$, this proves the desired bound for bundle sorting.

We can assume that $k \le N/B$, since otherwise bundle sorting can be executed by using any general sorting algorithm. We assume, without loss of generality, by the assumption of the indivisibility of records, that there is always exactly one copy of each record, and it is either on disk or in memory but not in both. At time t, let X_{ij} for $1 \le i \le k$ and $1 \le j \le N/B$ be the number of elements in the jth block on disk that need to end up on the ith row of the transposed matrix. At time t, let Y_i be the number of elements currently in internal memory that need to go on the ith row in the transposed matrix. We use the potential function $f(x) = x \log x$ for all $x \ge 0$. Its value at $x = 0$ is $f(0) = 0$. We define the overall potential function POT to be

$$POT = \sum_{i,j} f(X_{ij}) + \sum_i f(Y_i).$$

When the algorithm terminates, we have $Y_i = 0$ for all i, and the final value of potential POT is

$$\frac{N}{B}(B \log B) + 0 = N \log B.$$

If $k < B$, the initial potential is

$$\frac{N}{B} k \left(\frac{B}{k} \log \frac{B}{k} \right) = N \log \frac{B}{k},$$

and the initial potential is 0 otherwise (if $k \geq B$).

Note that our potential function satisfies

$$f(a+b) = (a+b)\log(a+b) \geq f(a) + f(b)$$

for all $a, b \geq 0$. Consider an output operation that writes a complete block of size B from memory to disk. If we write x_i records that need to go to the ith row and there are y_i such records in memory, then the change in potential is $\sum_i \big(f(x_i) + f(y_i) - f(x_i + y_i)\big) \leq 0$. Hence, output operations can only decrease the potential, and thus we need to consider only how much an input operation increases the potential.

If we read during an input operation a complete block of B records that contains x_i records that need to go to the ith row and there are y_i such records already in memory, then the change in the potential is

$$\sum_{1 \leq i \leq k} \left(f(x_i + y_i) - f(x_i) - f(y_i) \right).$$

By a convexity argument, this quantity is maximized when $x_i = B/k$ and $y_i = (M-B)/k$ for each $1 \leq i \leq k$, in which case the change in potential is bounded by $B \log(M/B)$.

We get a lower bound on the number of read operations by dividing the difference of the initial and final potentials by the bound on the maximum change in potential per read. For $k < B$, we get the I/O bound

$$\frac{N \log B - N \log \frac{B}{k}}{B \log \frac{M}{B}} = \frac{N}{B} \log_{M/B} k.$$

For $k \geq B$, we get the I/O bound

$$\frac{N \log B - 0}{B \log \frac{M}{B}} = \frac{N}{B} \log_{M/B} B.$$

We have thus proved a lower bound of $\Omega((N/B) \log_{M/B} \min(k, B))$ I/Os. Under our assumption that $k/B = B^{o(1)}$, this gives us an I/O lower bound for this case of bundle sorting of

$$\Omega \left(\frac{N}{B} \log_{M/B} k \right). \qquad \square$$

Theorem 5.1 for the lower bound of bundle sorting follows from Lemmas 5.1 and 5.2, since together they cover all possibilities for k, M, and B.

THEOREM 5.1. *The number of I/Os required in the worst case for sorting N data items of k distinct keys, using a memory of size M and block transfer size B, is*

$$\Omega \left(\frac{N}{B} \log_{M/B} k \right).$$

6. The disk latency model. In this section we consider the necessary modifications in the external bundle sorting algorithm in order to achieve an optimum number of I/Os in a more performance-sensitive model, as in [FFM98]. In this model, we differentiate between two types of I/Os: sequential I/Os and random I/Os, where there is a reduced cost for sequential I/Os. We start by presenting the model, followed by the modifications necessary in the bundle sorting, as presented in section 4.2. We also provide an additional, slightly different integer sorting algorithm that, depending on the setting, may enhance performance by up to 33% in this model for the integer sorting problem.

6.1. The model. The only difference between this model and the external memory model presented in section 3 is that we now differentiate between costs of two types of I/O: sequential and random I/Os. We define ℓ to be the latency to move the disk read/write head to a new position during a random seek. We define r to be the cost of reading a block of size B into internal memory once the read/write head is positioned at the start of the block.

The parameters N, M, and B, as before, are referred to as the *file size*, *memory size*, and *transfer block size*, respectively, and they satisfy $1 \leq B \leq M/2$ and $M < N$. We will consider the case where $D = 1$, meaning that there is no disk parallelism. It should be clear, from the above parameters, that the cost of a random I/O that loads one transfer block into memory is $\ell + r$, and the cost of a sequential I/O is simply r.

6.2. Optimal bundle sorting in the disk latency model. The modification for bundle sorting is based on the observation that in the worst-case scenario of the algorithm as described in section 4.2, every I/O in the sorting pass can be a random I/O. This is because we are loading $\lfloor M/B \rfloor$ blocks from disk into $\lfloor M/B \rfloor$ buckets, and in the worst case they may be written back in a round robin fashion resulting solely in random I/Os. However, if we decide to read more blocks into each bucket, we will increase the total number of I/Os, which will result in the worst case with sequential I/Os in addition to random I/Os.

Let α be the number of blocks that we load into each bucket, where clearly, $1 \leq \alpha \leq (M/2B)$. Thus, in each call to one-pass sorting of bundle sorting we sort into $\lfloor M/(\alpha B) \rfloor$ distinct keys, resulting in a total of $\log_{M/(\alpha B)} k$ passes over the sequence. However, we are now sure that at least $(\alpha - 1)/\alpha$ of the I/Os are sequential. We differentiate between the I/Os required in the external count sort, in which we perform only sequential I/Os, and the sorting pass, in which we also have random I/Os. Using Theorem 4.2, the performance is now

$$\frac{2N}{B}\left(\frac{1}{\alpha}(\ell + \alpha r)\log_{M/\alpha B} k + r \log_{M/B} \frac{k}{B}\right)$$

I/Os, and the optimal value of α can be determined via an optimization procedure. In section 7 we show experimentally how the execution time varies in this model as we change α.

7. Experiments. We conducted several experiments with various data sets and settings, while changing the size of the data sets N, the available memory M, the transfer block size B, and the number of distinct items k. The data sets were generated by the IBM test data generator (http://www.almaden.ibm.com/cs/quest), or taken from the U.S. Census data, and the following experiments were executed on both data sources. In all our experiments, the records consisted of 10-byte keys in 100-byte records. All experiments were run on a Pentium2, 300 MHz, 128 MB RAM machine.

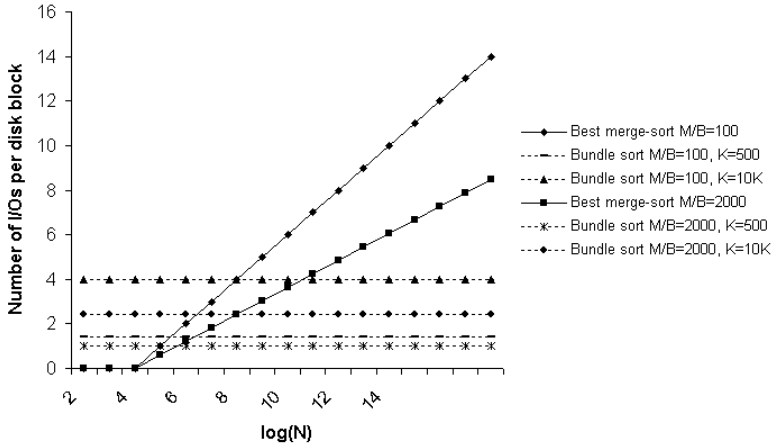


FIG. 2. *Bundle sorting versus regular sorting (best merge sort, for instance). The x-axis is the size of the data set drawn on a log scale. The y-axis is the number of I/Os performed per block of input. As can be seen, in contrast to merge sort, the number of I/Os per block in bundle sorting remains the same for a constant k as N increases.*

We first demonstrate an important feature of bundle sorting: As long as the number k of distinct keys remains constant, it performs the same number of I/O accesses per disk block with no dependence on the size of the data set. This is in contrast to general sort algorithms such as merge sort, which require more I/Os per disk block as the size of the data set increases. See Figure 2. The parameter B was set to 10 kB, and we tested for a memory of 1 MB and a memory of 20 MB. In both these cases merge sort, as expected, increased the number of I/Os per disk block as the size of the data set increased. In contrast, bundle sort performed a constant number of I/O accesses per disk block. As N increases, the improvement in performance becomes significant, demonstrating the advantages of bundle sorting. For instance, even when $k = 10000$ and the available memory is 20 MB, the break-even point occurs at $N = 1$ GB. As N increases, bundle sorting will perform better. If $k \leq 500$, then in the setting above, the break-even point occurs at $N = 10$ MB, making bundle sorting most appealing.

The next experiments demonstrate the performance of bundle sort as a function of k. See Figure 3. We set N at a fixed size of 1 GB and B at 10 kB. We ran the tests with a memory of 1 MB and 20 MB and counted the number of I/Os. We let k vary over a wide range of values from 2 to 10^9 ($k \leq N$ is always true). Since merge sort does not depend on the number of distinct keys, it performed the same number of I/O accesses per disk block in all these settings. In all these runs, as long as $k \leq N/B$, bundle sort performed better. When k is small the difference in performance is significant.

As for the disk-latency model, we show the optimal α values for various settings. Recall that in this model we attribute different costs to sequential and random I/Os. See Figure 4. We measured α for different ratios between ℓ, the cost of moving the disk reader to a random location (the latency), and r, the cost of reading a transfer block of size B. Parameter α also depends on the relation between M and B, so we plot M/B on the x-axis of the graph. As can be seen, when the ratio is 1, the optimal algorithm is exactly our bundle sorting algorithm, which counts only I/Os (hence it

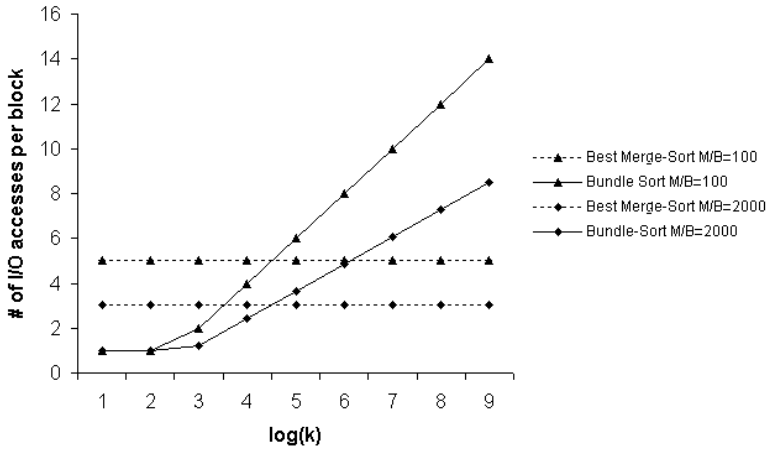


FIG. 3. *Bundle sorting versus regular sorting (best merge sort, for instance). The x-axis is the number of distinct keys (k) in the sequence drawn on a log scale. The y-axis is the number of I/Os per disk block. As can be seen, for $k \leq N/B$, bundle sorting performs better than merge sort, and the difference is large as k is smaller.*

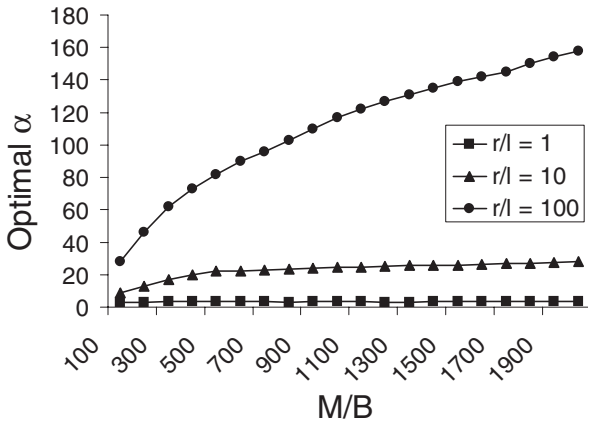


FIG. 4. *Optimum bundle sorting in the disk latency model—resolving α as a function of r, ℓ, and M/B.*

assumes that the cost of a random and a sequential I/O are equivalent). As this ratio increases, α increases, calling for a larger adaptation of our algorithm. Also affecting α, but in a more moderate way, is M/B. As this ratio increases, the optimum is achieved for a larger α.

8. Conclusions. We considered the sorting problem for large data sets with a moderate number of distinct keys, which we denote as bundle sorting, and identified it as a problem that is inherently easier than general sorting. We presented a simple in-place sorting algorithm for external memory, which may provide significant improvement over current sorting techniques. We also provided a matching lower bound, indicating that our solution is optimal.

Sorting is a fundamental problem, and any improvement in its solution may have

many applications. For instance, consider the sort join algorithm that computes join queries by first sorting the two relations that are to be joined, after which the join can be done efficiently in only one pass on both relations. Clearly, if the relations are large and their keys are taken from a universe of moderate size, then bundle sorting could provide more efficient execution than general sort.

It is interesting to note that the nature of the sorting algorithm is such that after the ith pass over the data set, the sequence is fully sorted into $(\lfloor M/B \rfloor)^i$ keys. In effect, the sequence is gradually sorted, where after each pass a further refinement is achieved, until finally the sequence is sorted. We can take advantage of this feature and use it in applications that benefit from quick, rough estimates that are gradually refined as we perform additional passes over the sequence. For instance, we could use it to produce intermediate join estimates, while refining the estimates by additional passes over the sequence. We can estimate the join after each iteration over the data set, improving the estimate after each such pass, and arrive at the final join after bundle sorting has completely finished.

The bundle sorting algorithm can be adapted efficiently and in a most straightforward way in the parallel disk model (PDM) described in [Vit99]. We now assume, in the external memory model, that $D > 1$, meaning that we can transfer D blocks into memory concurrently. This is like having D independent parallel disks. Assume that the data to be stored is initially located on one of the disks. In the first step we sort the data into exactly D buckets, writing each bucket into a distinct disk. Next, we sort, in parallel on each of the disks, the data set that was partitioned into each of the disks. Except for the initial partitioning step, we make full utilization of the parallel disks, thus enhancing performance by a factor of nearly D over all the bounds given in this paper. Note that extending bundle sorting to fit the PDM model was straightforward because of its top-down nature. Bundle sorting can also be utilized to enhance the performance of general sorting when the available working space is substantially smaller than the input set.

Bundle sorting is a fully in-place algorithm, which in effect causes the available memory to be doubled as compared to non–in-place algorithms. The performance gain from this feature can be significant. For instance, even if $M/B = 1000$, the performance gain is 10% and can be much higher for a smaller ratio. In some cases, an in-place sorting algorithm can avoid the use of high cost memory such as virtual memory.

We considered the disk latency model, which is a more performance-sensitive model where we differentiate between two types of I/Os—sequential and random I/Os—with a reduced cost for sequential I/Os. This model can be more realistic for performance analysis, and we have shown the necessary adaptation in the bundle sorting algorithm to arrive at an optimal solution in this model.

We have shown experimentation with real and synthetic data sets, which demonstrates that the theoretical analysis gives an accurate prediction of the actual performance.

REFERENCES

[ADADC+97] A. C. Arpaci-Dussaeu, R. H. Arpaci-Dussaeu, D. E. Culler, J. M. Hellerstein, and D. A. Patterson, *High-performance sorting on networks of workstations*, in Proceedings of the ACM SIGMOD International Conference on Management of Data, 1997, ACM, New York, 1997, pp. 243–254.

[Aga96] R. C. Agarwal, *A super scalar sort algorithm for RISC processors*, in Proceed-

ings of the ACM SIGMOD International Conference on Management of Data, Montral, QC, 1996, ACM, New York, 1996, pp. 240–246.

[AV88] A. Aggarwal and J. S. Vitter, *The input/output complexity of sorting and related problems*, Comm. ACM, 31 (1988), pp. 1116–1127.

[BBW86] M. Beck, D. Bitton, and W. K. Wilkinson, *Sorting Large Files on a Backend Multiprocessor*, Technical Report 86-741, Department of Computer Science, Cornell University, Ithaca, NY, 1986.

[BGK90] B. Baugsto, J. Greipsland, and J. Kamerbeek, *Sorting large data files on POMA*, in Proceedings of CONPAR-90: Proceedings of the Joint International Conference on Vector and Parallel Processing, Zurich, Switzerland, 1990, Springer-Verlag, pp. 536–547.

[FFM98] M. Farach, P. Ferragina, and S. Muthukrishnan, *Overcoming the memory bottleneck in suffix tree construction*, in Proceedings of the 39th IEEE Annual Symposium on Foundations of Computer Science, Palo Alto, CA, 1998, IEEE Press, Piscataway, NJ, 1998, pp. 174–183.

[Gra93] G. Graefe, *Query evaluation techniques for large databases*, ACM Comput. Surveys, 25 (1993), pp. 73–170.

[IBM95] IBM, *Database* 2*, Administration Guide for Common Servers*, Version 2, 1995.

[Knu73] D. E. Knuth, *The Art of Computer Programming. Vol.* 3. *Sorting and Searching*, Addison Wesley Longman Publishing, Redwood City, CA, 1973.

[MSV00] Y. Matias, E. Segal, and J. S. Vitter, *Efficient bundle sorting*, in Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, 2000, SIAM, Philadelphia, 2000, pp. 839–848.

[Vit99] J. S. Vitter, *External memory algorithms and data structures*, in External Memory Algorithms and Visualization, J. Abello and J. S. Vitter, eds., AMS, Providence, RI, 1999; updated version available online at http://www.cs.duke.edu/~jsv/.

[WVI98] M. Wang, J. S. Vitter, and B. R. Iyer, *Scalable mining for classification rules in relational databases*, in Proceedings of the International Database Engineering & Application Symposium, Cardiff, Wales, 1998, IEEE Computer Society Press, Washington, DC, pp. 58–67.

[ZL98] W. Zhang and P.-A. Larson, *Buffering and read-ahead strategies for external mergesort*, in Proceedings of the International Conference on Very Large Data Bases (VLDB), 1998, Morgan Kaufmann, San Francisco, CA, pp. 523–533.

SIAM J. COMPUT.
Vol. 36, No. 2, pp. 411–432

APPROXIMATION ALGORITHMS FOR METRIC FACILITY LOCATION PROBLEMS*

MOHAMMAD MAHDIAN†, YINYU YE‡, AND JIAWEI ZHANG§

Abstract. In this paper we present a 1.52-approximation algorithm for the metric uncapacitated facility location problem, and a 2-approximation algorithm for the metric capacitated facility location problem with soft capacities. Both these algorithms improve the best previously known approximation factor for the corresponding problem, and our soft-capacitated facility location algorithm achieves the integrality gap of the standard linear programming relaxation of the problem. Furthermore, we will show, using a result of Thorup, that our algorithms can be implemented in quasi-linear time.

Key words. approximation algorithms, facility location problem, greedy method, linear programming

AMS subject classifications. 90C59, 90C27, 90B80

DOI. 10.1137/S0097539703435716

1. Introduction.

Variants of the facility location problem (FLP) have been studied extensively in the operations research and management science literatures and have received considerable attention in the area of approximation algorithms (see [21] for a survey). In the metric uncapacitated facility location problem (UFLP), which is the most basic FLP, we are given a set $\mathcal{F}$ of *facilities*, a set $\mathcal{C}$ of *cities* (i.e., clients), a cost f_i for opening facility $i \in \mathcal{F}$, and a connection cost c_{ij} for connecting client j to facility i. The objective is to open a subset of the facilities in $\mathcal{F}$ and connect each city to an open facility so that the total cost, that is the cost of opening facilities and connecting the clients, is minimized. We assume that the connection costs form a metric, meaning that they are symmetric and satisfy the triangle inequality.

Since the first constant factor approximation algorithm due to Shmoys, Tardos, and Aardal [22], a large number of approximation algorithms have been proposed for UFLP [23, 12, 25, 14, 2, 4, 6, 8, 15]. Table 1 shows a summary of these results. Prior to this work, the best known approximation factor for UFLP was 1.58, given by Sviridenko [23], which was achieved using linear programming (LP) rounding. Guha and Khuller [8] proved that it is impossible to get an approximation guarantee of 1.463 for UFLP, unless $\mathbf{NP} \subseteq \mathrm{DTIME}[n^{O(\log \log n)}]$. In this paper, we give a 1.52-approximation algorithm for UFLP, which can be implemented in quasi-linear time, using a result of Thorup [24]. Our algorithm combines the greedy algorithm of Jain, Mahdian, and Saberi [13] and Jain et al. [12] with the idea of cost scaling, and is analyzed using a factor-revealing linear program.

The growing interest in UFLP is not only due to its applications in a large number of settings [7], but also due to the fact that UFLP is one of the most basic models

*Received by the editors October 6, 2003; accepted for publication (in revised form) February 7, 2006; published electronically June 30, 2006. This paper is based on preliminary versions [18, 19].
http://www.siam.org/journals/sicomp/36-2/43571.html

†Microsoft Research, One Microsoft Way, Redmond, WA 98052 (mahdian@microsoft.com).

‡Department of Management Science and Engineering, School of Engineering, Stanford University, Stanford, CA 94305 (yinyu-ye@stanford.edu). This author's research was supported in part by NSF grant DMI-0231600.

§IOMS-Operations Management, Stern School of Business, New York University, 44 West 4th St., Suite 8-66, New York, NY 10012-1126 (jzhang@stern.nyu.edu).

TABLE 1
Approximation algorithms for UFLP.

Approx. factor	Reference	Technique/running time
$O(\ln n_c)$	[10]	Greedy algorithm/$O(n^3)$
3.16	[22]	LP rounding
2.41	[8]	LP rounding + greedy augmentation
1.736	[6]	LP rounding
$5+\epsilon$	[15]	Local search/$O(n^6 \log(n/\epsilon))$
3	[14]	Primal-dual method/$O(n^2 \log n)$
1.853	[4]	Primal-dual method + greedy augmentation/$O(n^3)$
1.728	[4]	LP rounding + primal-dual method + greedy augmentation
1.861	[16, 12]	Greedy algorithm/$O(n^2 \log n)$
1.61	[13, 12]	Greedy algorithm/$O(n^3)$
1.582	[23]	LP rounding
1.52	This paper	Greedy algorithm + cost scaling/$\tilde{O}(n)$

among discrete location problems. The insights gained in dealing with UFLP may also apply to more complicated location models, and in many cases the latter can be reduced directly to UFLP.

In the second part of this paper, we give a 2-approximation algorithm for the soft-capacitated facility location problem (SCFLP) by reducing it to UFLP. SCFLP is similar to UFLP, except that there is a capacity u_i associated with each facility i, which means that if we want this facility to serve x cities, we have to open it $\lceil x/u_i \rceil$ times at a cost of $f_i\lceil x/u_i \rceil$. This problem is also known in the operations research literature as the facility location problem with integer decision variables (see [3] and [20]). Chudak and Shmoys [5] gave a 3-approximation algorithm for SCFLP with uniform capacities (i.e., $u_i = u$ for all $i \in \mathcal{F}$) using LP rounding. For nonuniform capacities, Jain and Vazirani [14] showed how to reduce this problem to UFLP, and by solving UFLP through a primal-dual algorithm, they obtained a 4-approximation. Arya et al. [2] proposed a local search algorithm that achieves an approximation ratio of 3.72. Following the approach of Jain and Vazirani [14], Jain and coworkers [13, 12] showed that SCFLP can be approximated within a factor of 3. This was the best previously known algorithm for this problem. We improve this factor to 2, achieving the integrality gap of the natural LP relaxation of the problem. The main idea of our algorithm is to consider algorithms and reductions that have separate (not necessarily equal) approximation factors for the facility and connection costs. We will define the concept of *bifactor* approximate reduction in this paper, and show how it can be used to get an approximation factor of 2 for SCFLP. The idea of using bifactor approximation algorithms and reductions can be used to improve the approximation factor of several other problems.

The rest of this paper is organized as follows: In section 2 the necessary definitions and notation are presented. In section 3, we present the algorithm for UFLP and its underlying intuition, and we prove the upper bound of 1.52 on the approximation factor of the algorithm. In section 4 we present a lemma on the approximability of the linear-cost FLP. In section 5 we define the concept of a bifactor approximate reduction between facility location problems. Using bifactor reductions to the linear-cost FLP and the lemma proved in section 4, we present algorithms for SCFLP and the concave SCFLP. Concluding remarks are given in section 6.

2. Preliminaries. In this paper, we will define reductions between various FLPs. Many such problems can be considered as special cases of the *universal FLP*, as defined below. This problem was first defined in [9] and further studied in [17].

DEFINITION 1. *In the* metric universal FLP*, we are given a set $\mathcal{C}$ of n_c cities, a set $\mathcal{F}$ of n_f facilities, a connection cost c_{ij} between city j and facility i for every $i \in \mathcal{F}, j \in \mathcal{C}$, and a facility cost function $f_i : \{0, \ldots, n_c\} \mapsto \mathcal{R}^+$ for every $i \in \mathcal{F}$. Connection costs are symmetric and obey the triangle inequality. The value of $f_i(k)$ equals the cost of opening facility i if it is used to serve k cities. A solution to the problem is a function $\phi : \mathcal{C} \to \mathcal{F}$ assigning each city to a facility. The* facility cost F_ϕ *of the solution ϕ is defined as $\sum_{i \in \mathcal{F}} f_i(|\{j : \phi(j) = i\}|)$, i.e., the total cost for opening facilities. The* connection cost *(or service cost) C_ϕ of ϕ is $\sum_{j \in \mathcal{C}} c_{\phi(j),j}$, i.e., the total cost of opening each city to its assigned facility. The objective is to find a solution ϕ that minimizes the sum $F_\phi + C_\phi$.*

For the metric universal FLP, we distinguish two models by how the connection costs are given. In the distance oracle model, the connection costs are explicitly given by a matrix (c_{ij}) for any $i \in \mathcal{F}$ and $j \in \mathcal{C}$. In the sparse graph model, $\mathcal{C}$ and $\mathcal{F}$ are nodes of an undirected graph (which may not be complete) in which the cost of each edge is given, and the connection cost between a facility i and a client j is implicitly given by the shortest distance between i and j.

Now we can define the uncapacitated and soft-capacitated FLPs as special cases of the universal FLP, as follows.

DEFINITION 2. *The* metric uncapacitated facility location problem (UFLP) *is a special case of the universal FLP in which all facility cost functions are of the following form: for each $i \in \mathcal{F}$, $f_i(k) = 0$ if $k = 0$, and $f_i(k) = f_i$ if $k > 0$, where f_i is a constant which is called the facility cost of i.*

DEFINITION 3. *The* metric soft-capacitated facility location problem (SCFLP) *is a special case of the universal FLP in which all facility cost functions are of the form $f_i(k) = f_i\lceil k/u_i \rceil$, where f_i and u_i are constants for every $i \in \mathcal{F}$, and u_i is called the capacity of facility i.*

The algorithms presented in this paper build upon an earlier approximation algorithm of Jain and coworkers [13, 12], which is sketched below. We call this algorithm the JMS algorithm.

THE JMS ALGORITHM.

1. At the beginning, all cities are *unconnected*, all facilities are *unopened*, and the *budget* of every city j, denoted by B_j, is initialized to 0. At every moment, each city j offers some money from its budget to each *unopened* facility i. The amount of this offer is equal to $\max(B_j - c_{ij}, 0)$ if j is unconnected, and $\max(c_{i'j} - c_{ij}, 0)$ if it is connected to some other facility i'.
2. While there is an unconnected city, increase the budget of each *unconnected* city at the same rate, until one of the following events occurs:
   (a) For some unopened facility i, the total offer that it receives from cities is equal to the cost of opening i. In this case, we open facility i, and for every city j (connected or unconnected) which has a nonzero offer to i, we connect j to i.
   (b) For some unconnected city j, and some facility i that is already open, the budget of j is equal to the connection cost c_{ij}. In this case, we connect j to i.

The analysis of the JMS algorithm has the feature that allows the approximation factor for the facility cost to be different from the approximation factor for the connection cost, and gives a way to compute the tradeoff between these two factors. The following definition captures this notion.

DEFINITION 4. *An algorithm is called a (γ_f, γ_c)-approximation algorithm for the universal FLP if, for every instance $\mathcal{I}$ of the universal FLP and for* every *solution*

SOL for $\mathcal{I}$ with facility cost F_{SOL} and connection cost C_{SOL}, the cost of the solution found by the algorithm is at most $\gamma_f F_{SOL} + \gamma_c C_{SOL}$.

Recall the following theorem of Jain and coworkers [13, 12] on the approximation factor of the JMS algorithm.

THEOREM A (see [13, 12]). *Let $\gamma_f \geq 1$ be fixed and $\gamma_c := \sup_k\{z_k\}$, where z_k is the solution of the following optimization program, referred to as the factor-revealing LP:*

$$\text{(LP1)} \quad \text{maximize } \frac{\sum_{i=1}^{k} \alpha_i - \gamma_f f}{\sum_{i=1}^{k} d_i}$$

$$\text{(1)} \quad \text{subject to } \forall\, 1 \leq i < k: \ \alpha_i \leq \alpha_{i+1},$$

$$\text{(2)} \quad \forall\, 1 \leq j < i < k: \ r_{j,i} \geq r_{j,i+1},$$

$$\text{(3)} \quad \forall\, 1 \leq j < i \leq k: \ \alpha_i \leq r_{j,i} + d_i + d_j,$$

$$\text{(4)} \quad \forall\, 1 \leq i \leq k: \ \sum_{j=1}^{i-1} \max(r_{j,i} - d_j, 0) + \sum_{j=i}^{k} \max(\alpha_i - d_j, 0) \leq f,$$

$$\text{(5)} \quad \forall\, 1 \leq j \leq i \leq k: \ \alpha_j, d_j, f, r_{j,i} \geq 0.$$

Then the JMS algorithm is a (γ_f, γ_c)-approximation algorithm for UFLP. Furthermore, for $\gamma_f = 1$ we have $\gamma_c \leq 2$.

3. The uncapacitated facility location algorithm.

3.1. Description of the algorithm. We use the JMS algorithm to solve the UFLP with an improved approximation factor. Our algorithm has two phases. In the *first* phase, we scale up the opening costs of all facilities by a factor of δ (which is a constant that will be fixed later) and then run the JMS algorithm to find a solution. The technique of cost scaling has been previously used by Charikar and Guha [4] for the FLP in order to take advantage of the asymmetry between the performance of the algorithm with respect to the facility and that with respect to the connection costs.

Here we give a different intuitive reason: Intuitively, the facilities that are opened by the JMS algorithm with the scaled-up facility costs are those that are very economical, because we weigh the facility cost more than the connection cost in the objective function. Therefore, we open these facilities in the first phase of the algorithm.

One important property of the JMS algorithm is that it finds a solution in which there is no unopened facility that one can open to decrease the cost (without closing any other facility). This is because for each city j and facility i, j offers to i the amount that it would save in the connection cost if it gets its service from i. This is, in fact, the main advantage of the JMS algorithm over a previous algorithm of Mahdian et al. [16].

However, the facility costs have been scaled up in the first phase of our algorithm. Therefore, it is possible that the total cost (in terms of the original cost) can be reduced more by opening an unopened facility and by reconnecting each city to its closest open facility. This motivates the second phase of our algorithm.

In the *second* phase of the algorithm, we decrease the scaling factor δ at rate 1, so that at time t the cost of facility i has reduced to $(\delta - t)f_i$. If at any point during this process a facility could be opened without increasing the total cost (i.e., if the opening cost of the facility equals the total amount that cities can save by switching their "service provider" to that facility), then we open the facility and connect each

city to its closest open facility. We stop when the scaling factor becomes 1. This is equivalent to a greedy procedure introduced by Guha and Khuller [8] and Charikar and Guha [4]. In this procedure, in each iteration we pick a facility u of opening cost f_u such that if by opening u the total connection cost decreases from C to C'_u, the ratio $(C - C'_u - f_u)/f_u$ is maximized. If this ratio is positive, then we open the facility u and iterate; otherwise we stop. It is not hard to see that the second phase of our algorithm is equivalent to the Charikar–Guha–Khuller procedure: in the second phase of our algorithm, the first facility u that is opened corresponds to the minimum value of t, or the maximum value of $\delta - t$, for which we have $(\delta - t)f_u = C - C'_u$. In other words, our algorithm picks the facility u for which the value of $(C - C'_u)/f_u$ is maximized, and stops when this value becomes less than or equal to 1 for all u. This is the same as what the Charikar–Guha–Khuller procedure does. The original analysis of our algorithm in [18] was based on a lemma by Charikar and Guha [4]. Here we give an alternative analysis of our algorithm that uses only a single factor-revealing LP.

We call our two-phase algorithm Algorithm A. In the remainder of this section, we analyze Algorithm A and prove that it always outputs a solution to the UFLP of cost at most 1.52 times the optimum. The analysis is divided into three parts. First, in section 3.2, we derive the factor-revealing linear program whose solution gives the approximation ratio of our algorithm. Next, in section 3.3, we analyze this linear program, and compute its solution in terms of the approximation factors of the JMS algorithm. This gives the following result.

THEOREM 1. *Let (γ_f, γ_c) be a pair obtained from the factor-revealing linear program* (LP1). *Then for every $\delta \geq 1$, Algorithm A is a $(\gamma_f + \ln(\delta) + \epsilon, 1 + \frac{\gamma_c - 1}{\delta})$-approximation algorithm for the UFLP.*

Finally, we analyze the factor-revealing linear program (1) and show that the JMS algorithm is a $(1.11, 1.78)$-approximation algorithm for the UFLP. This, together with the above theorem for $\delta = 1.504$, implies that Algorithm A is a 1.52-approximation algorithm for the UFLP. We will show in section 3.4 that this algorithm can be implemented in quasi-linear time, both for the distance oracle model and for the sparse graph model.

3.2. Deriving the factor-revealing LP. Recall that the JMS algorithm, in addition to finding a solution for the scaled instance, outputs the *share* of each city in the total cost of the solution. Let α_j denote the share of city j in the total cost. In other words, α_j is the value of the variable B_j at the end of the JMS algorithm. Therefore the total cost of the solution is $\sum_j \alpha_j$. Consider an arbitrary collection $\mathcal{S}$ consisting of a single facility $f_{\mathcal{S}}$ and k cities. Let δf (f in the original instance) denote the opening cost of facility $f_{\mathcal{S}}$; α_j denote the share of city j in the total cost (where cities are ordered such that $\alpha_1 \leq \cdots \leq \alpha_k$); d_j denote the connection cost between city j and facility $f_{\mathcal{S}}$; and $r_{j,i}$ $(i > j)$ denote the connection cost between city j and the facility that it is connected to at time α_i, right before city i gets connected for the first time (or if cities i and j get connected at the same time, define $r_{j,i} = \alpha_i = \alpha_j$). The main step in the analysis of the JMS algorithm is to prove that for any such collection $\mathcal{S}$, the δf, d_j, α_j, and $r_{j,i}$ values constitute a feasible solution to the program (LP1), where f is now replaced with δf since the facility costs have been scaled up by δ.

We implement and analyze the second phase as the following. Instead of decreasing the scaling factor continuously from δ to 1, we decrease it discretely in L steps, where L is a constant. Let δ_i denote the value of the scaling factor in the ith step. Therefore, $\delta = \delta_1 > \delta_2 > \cdots > \delta_L = 1$. We will fix the value of the δ_i's later. After decreasing the scaling factor from δ_{i-1} to δ_i, we consider facilities in an *arbitrary* or-

der, and open those that can be opened without increasing the total cost. We denote this modified algorithm by A_L. Clearly, if L is sufficiently large (depending on the instance), the algorithm A_L computes the same solution as Algorithm A.

In order to analyze the above algorithm, we need to add extra variables and inequalities to the inequalities in the factor-revealing program (LP1) given in Theorem A. Let $r_{j,k+i}$ denote the connection cost that city j in $\mathcal{S}$ pays after we change the scaling factor to δ_i and process all facilities as described above. (Thus, $r_{j,k+1}$ is the connection cost of city j after the first phase.) Therefore, by the description of the algorithm, we have

$$\forall\, 1 \le i \le L: \ \sum_{j=1}^{k} \max(r_{j,k+i} - d_j, 0) \le \delta_i f.$$

This is because if the above inequality is violated and if $f_{\mathcal{S}}$ is not open, we could open $f_{\mathcal{S}}$ and decrease the total cost. If $f_{\mathcal{S}}$ is open, then $r_{j,k+i} \le d_j$ for all j, and the inequality holds.

Now, we compute the share of the city j in the total cost of the solution that algorithm A_L finds. In the first phase of the algorithm, the share of city j in the total cost is α_j. Of this amount, $r_{j,k+1}$ is spent on the connection cost, and $\alpha_j - r_{j,k+1}$ is spent on the facility costs. However, since the facility costs are scaled up by a factor of δ in the first phase, therefore the share of city j in the *facility costs* in the original instance is equal to $(\alpha_j - r_{j,k+1})/\delta$. After we reduce the scaling factor from δ_i to δ_{i+1} $(i = 1, \ldots, L-1)$, the connection cost of city j is reduced from $r_{j,k+i}$ to $r_{j,k+i+1}$. Therefore, in this step, the share of city j in the facility costs is $r_{j,k+i} - r_{j,k+i+1}$ with respect to the scaled instance, or $(r_{j,k+i} - r_{j,k+i+1})/\delta_{i+1}$ with respect to the original instance. Thus, at the end of the algorithm, the total share of city j in the facility costs is

$$\frac{\alpha_j - r_{j,k+1}}{\delta} + \sum_{i=1}^{L-1} \frac{r_{j,k+i} - r_{j,k+i+1}}{\delta_{i+1}}.$$

We also know that the final amount that city j pays for the connection cost is $r_{j,k+L}$. Therefore, the share of the city j in the total cost of the solution is

$$\frac{\alpha_j - r_{j,k+1}}{\delta} + \sum_{i=1}^{L-1} \frac{r_{j,k+i} - r_{j,k+i+1}}{\delta_{i+1}} + r_{j,k+L+1} = \frac{\alpha_j}{\delta} + \sum_{i=1}^{L-1} \left(\frac{1}{\delta_{i+1}} - \frac{1}{\delta_i} \right) r_{j,k+i}. \tag{6}$$

This, together with a *dual fitting* argument similar to [12], implies the following.

THEOREM 2. *Let (ξ_f, ξ_c) be such that $\xi_f \ge 1$ and ξ_c is an upper bound on the solution of the following maximization program for every k:*

$$\text{(LP2)} \quad \text{maximize} \ \frac{\sum_{j=1}^{k} \left(\frac{\alpha_j}{\delta} + \sum_{i=1}^{L-1} \left(\frac{1}{\delta_{i+1}} - \frac{1}{\delta_i} \right) r_{j,k+i} \right) - \xi_f f}{\sum_{i=1}^{k} d_i}$$

$$\text{subject to } \forall\, 1 \le i < k: \ \alpha_i \le \alpha_{i+1}, \tag{7}$$

$$\forall\, 1 \le j < i < k: \ r_{j,i} \ge r_{j,i+1}, \tag{8}$$

$$\forall\, 1 \le j < i \le k: \ \alpha_i \le r_{j,i} + d_i + d_j, \tag{9}$$

$$\forall\, 1 \le i \le k: \ \sum_{j=1}^{i-1} \max(r_{j,i} - d_j, 0) + \sum_{j=i}^{k} \max(\alpha_i - d_j, 0) \le \delta f, \tag{10}$$

$$\forall\, 1 \le i \le L: \quad \sum_{j=1}^{k} \max(r_{j,k+i} - d_j, 0) \le \delta_i f, \tag{11}$$

$$\forall\, 1 \le j \le i \le k: \quad \alpha_j, d_j, f, r_{j,i} \ge 0. \tag{12}$$

Then, algorithm A_L is a (ξ_f, ξ_c)-approximation algorithm for UFLP.

3.3. Analyzing the factor-revealing LP. In the following theorem, we analyze the factor-revealing linear program (LP2) and prove Theorem 1. In order to do this, we need to set the values of the δ_i's. Here, for simplicity of computations, we set δ_i to $\delta^{\frac{L-i}{L-1}}$; however, it is easy to observe that any choice of δ_i's such that $\delta = \delta_1 > \delta_2 > \cdots > \delta_L = 1$ and the limit of $\max_i(\delta_i - \delta_{i+1})$ as L tends to infinity is zero will also work.

THEOREM 3. *Let (γ_f, γ_c) be a pair given by the maximization program* (LP1) *in Theorem* A, *and $\delta \ge 1$ be an arbitrary number. Then for every ϵ, if L is a sufficiently large constant, algorithm A_L is a $(\gamma_f + \ln(\delta) + \epsilon, 1 + \frac{\gamma_c - 1}{\delta})$-approximation algorithm for the UFLP.*

Proof. Since the inequalities of the factor-revealing program (7) are a superset of the inequalities of the factor-revealing program (1), by Theorem A and the definition of (γ_f, γ_c), we have

$$\sum_{j=1}^{k} \alpha_j \le \gamma_f \delta f + \gamma_c \sum_{j=1}^{k} d_j. \tag{13}$$

By inequality (11), for every $i = 1, \ldots, L$ we have

$$\sum_{j=1}^{k} r_{j,k+i} \le \sum_{j=1}^{k} \max(r_{j,k+i} - d_j, 0) + \sum_{j=1}^{k} d_j \le \delta_i f + \sum_{j=1}^{k} d_j. \tag{14}$$

Therefore,

$$\begin{aligned}
&\sum_{j=1}^{k} \left(\frac{\alpha_j}{\delta} + \sum_{i=1}^{L-1} \left(\frac{1}{\delta_{i+1}} - \frac{1}{\delta_i} \right) r_{j,k+i} \right) \\
&\quad = \frac{1}{\delta} \left(\sum_{j=1}^{k} \alpha_j \right) + \sum_{i=1}^{L-1} \left(\left(\frac{1}{\delta_{i+1}} - \frac{1}{\delta_i} \right) \sum_{j=1}^{k} r_{j,k+i} \right) \\
&\quad \le \frac{1}{\delta} \left(\gamma_f \delta f + \gamma_c \sum_{j=1}^{k} d_j \right) + \sum_{i=1}^{L-1} \left(\left(\frac{1}{\delta_{i+1}} - \frac{1}{\delta_i} \right) \left(\delta_i f + \sum_{j=1}^{k} d_j \right) \right) \\
&\quad = \gamma_f f + \frac{\gamma_c}{\delta} \sum_{j=1}^{k} d_j + \sum_{i=1}^{L-1} \left(\frac{\delta_i}{\delta_{i+1}} - 1 \right) f + \left(\frac{1}{\delta_L} - \frac{1}{\delta_1} \right) \sum_{j=1}^{k} d_j \\
&\quad = \left(\gamma_f + (L-1)(\delta^{1/(L-1)} - 1) \right) f + \left(\frac{\gamma_c}{\delta} + 1 - \frac{1}{\delta} \right) \sum_{j=1}^{k} d_j.
\end{aligned}$$

This, together with Theorem 2, shows that A_L is a $(\gamma_f + (L-1)(\delta^{1/(L-1)} - 1), 1 + \frac{\gamma_c - 1}{\delta})$-approximation algorithm for the UFLP. The fact that the limit of $(L-1)(\delta^{1/(L-1)} - 1)$ as L tends to infinity is $\ln(\delta)$ completes the proof. ∎

We observe that the proof of Theorem 3 goes through as long as the limit of $\sum_{i=1}^{L-1}(\frac{\delta_i}{\delta_{i+1}}-1)$ as L tends to infinity is $\ln(\delta)$. This condition holds if we choose the δ_i's such that $\delta=\delta_1>\delta_2>\cdots>\delta_L=1$ and the limit of $\max_i(\delta_i-\delta_{i+1})$ as L tends to infinity is zero. It can be seen as follows. Let $x_i=\frac{\delta_i}{\delta_{i+1}}-1>0$. Then, for $i=1,2,\ldots,L-1$,

$$x_i-o(x_i)\le \ln\left(\frac{\delta_i}{\delta_{i+1}}\right)\le x_i.$$

It follows that

$$\sum_{i=1}^{L-1} x_i\left(1-\frac{o(x_i)}{x_i}\right)\le \ln(\delta)\le \sum_{i=1}^{L-1} x_i.$$

Since $\lim_{L\to\infty}\frac{o(x_i)}{x_i}=\lim_{x_i\to 0}\frac{o(x_i)}{x_i}=0$, we conclude that $\lim_{L\to\infty}\sum_{i=1}^{L-1}x_i=\ln(\delta)$.

Now we analyze the factor-revealing linear program (LP1) and show that the JMS algorithm is a $(1.11, 1.78)$-approximation algorithm.

LEMMA 4. *Let $\gamma_f=1.11$. Then for every k, the solution of the factor-revealing linear program* (LP1) *is at most* 1.78.

Proof. For the proof, see the appendix. □

Remark 1. Numerical computations using CPLEX show that $z_{500}\approx 1.7743$ and therefore $\gamma_c>1.774$ for $\gamma_f=1.11$. Thus, the estimate provided by the above lemma for the value of γ_c is close to its actual value.

3.4. Running time. The above analysis of Algorithm A, together with a recent result of Thorup [24], enables us to prove the following result.

COROLLARY 5. *For every $\epsilon>0$, there is a quasi-linear time $(1.52+\epsilon)$-approximation algorithm for the UFLP, both in the distance oracle model and in the sparse graph model.*

Proof sketch. We use the algorithm A_L for a large constant L. Thorup [24] shows that for every $\epsilon>0$, the JMS algorithm can be implemented in quasi-linear time (in both the distance oracle and the sparse graph models) with an approximation factor of $1.61+\epsilon$. It is straightforward to see that his argument actually implies the stronger conclusion that the quasi-linear algorithm is a $(\gamma_f+\epsilon,\gamma_c+\epsilon)$-approximation, where (γ_f,γ_c) are given by Theorem A. This shows that the first phase of algorithm A_L can be implemented in quasi-linear time. The second phase consists of a constant number of rounds. Therefore, we need to show only that each of these rounds can be implemented in quasi-linear time. This is easy to see in the distance oracle model. In the sparse graph model, we can use the exact same argument as that used by Thorup in the proof of Lemma 5.1 of [24]. □

4. The linear-cost FLP. The *linear-cost FLP* is a special case of the universal FLP in which the facility costs are of the form

$$f_i(k)=\begin{cases}0, & k=0,\\ a_ik+b_i, & k>0,\end{cases}$$

where a_i and b_i are nonnegative values for each $i\in\mathcal{F}$. a_i and b_i are called the marginal (or incremental) and setup costs, respectively, of facility i.

We denote an instance of the linear-cost FLP with marginal costs (a_i), setup costs (b_i), and connection costs (c_{ij}) by $LFLP(a,b,c)$. Clearly, the regular UFLP is

a special case of the linear-cost FLP with $a_i = 0$, i.e., $LFLP(0, b, c)$. Furthermore, it is straightforward to see that $LFLP(a, b, c)$ is equivalent to an instance of the regular UFLP in which the marginal costs are added to the connection costs. More precisely, let $\bar{c}_{ij} = c_{ij} + a_i$ for $i \in \mathcal{F}$ and $j \in \mathcal{C}$, and consider an instance of UFLP with facility costs (b_i) and connection costs $(\bar{c}_{ij})$. We denote this instance by $UFLP(b, c + a)$. It is easy to see that $LFLP(a, b, c)$ is equivalent to $UFLP(b, c + a)$. Thus, the linear-cost FLP can be solved using any algorithm for UFLP, and the overall approximation ratio will be the same. However, for applications in the next section, we need bifactor approximation factors of the algorithm (as defined in Definition 4).

It is not necessarily true that applying a (γ_f, γ_c)-approximation algorithm for UFLP on the instance $UFLP(b, a + c)$ will give a (γ_f, γ_c)-approximate solution for $LFLP(a, b, c)$. However, we will show that the JMS algorithm has this property. The following lemma generalizes Theorem A for the linear-cost FLP.

LEMMA 6. *Let (γ_f, γ_c) be a pair obtained from the factor-revealing LP in Theorem* A. *Then applying the JMS algorithm on the instance $UFLP(b, a + c)$ will give a (γ_f, γ_c)-approximate solution for $LFLP(a, b, c)$.*

Proof. Let SOL be an arbitrary solution for $LFLP(a, b, c)$, which can also be viewed as a solution for $UFLP(b, \bar{c})$ for $\bar{c} = c + a$. Consider a facility f that is open in SOL and the set of clients connected to it in SOL. Let k denote the number of these clients, $f(k) = ak + b$ (for $k > 0$) be the facility cost function of f, and $\bar{d}_j$ denote the connection cost between client j and the facility f in the instance $UFLP(b, a + c)$. Therefore, $d_j = \bar{d}_j - a$ is the corresponding connection cost in the original instance $LFLP(a, b, c)$. Recall the definition of α_j and $r_{j,i}$ in the factor-revealing linear program of Theorem A. By inequality (3) we also know that $\alpha_i \leq r_{j,i} + \bar{d}_j + \bar{d}_i$. We strengthen this inequality as follows.

CLAIM 7. $\alpha_i \leq r_{j,i} + d_j + d_i$.

Proof. The claim is true if $\alpha_i = \alpha_j$, since it happens only if $r_{j,i} = \alpha_j$. Otherwise, consider clients i and $j(< i)$ at time $t = \alpha_i - \epsilon$. Let s be the facility to which j is assigned at time t. By the triangle inequality, we have

$$\bar{c}_{si} = c_{si} + a_s \leq c_{sj} + d_i + d_j + a_s = \bar{c}_{sj} + d_i + d_j \leq r_{j,i} + d_i + d_j.$$

On the other hand, $\alpha_i \leq \bar{c}_{si}$ since otherwise i could have connected to facility s at a time earlier than t. □

Also, by inequality (4), we know that

$$\sum_{j=1}^{i-1} \max(r_{j,i} - \bar{d}_j, 0) + \sum_{j=i}^{k} \max(\alpha_i - \bar{d}_j, 0) \leq b.$$

Notice that $\max(a - x, 0) \geq \max(a, 0) - x$ if $x \geq 0$. Therefore, we have

$$\sum_{j=1}^{i-1} \max(r_{j,i} - d_j, 0) + \sum_{j=i}^{k} \max(\alpha_i - d_j, 0) \leq b + ka. \tag{15}$$

Claim 7 and inequality (15) show that the values α_j, $r_{j,i}$, d_j, a, and b constitute a feasible solution of the following optimization program:

$$\text{maximize } \frac{\sum_{i=1}^{k} \alpha_i - \gamma_f(ak + b)}{\sum_{i=1}^{k} d_i}$$

$$\begin{aligned}
\text{subject to } & \forall\, 1 \le i < k:\ \alpha_i \le \alpha_{i+1},\\
& \forall\, 1 \le j < i < k:\ r_{j,i} \ge r_{j,i+1},\\
& \forall\, 1 \le j < i \le k:\ \alpha_i \le r_{j,i} + d_i + d_j,\\
& \forall\, 1 \le i \le k:\ \sum_{j=1}^{i-1} \max(r_{j,i} - d_j, 0) + \sum_{j=i}^{k} \max(\alpha_i - d_j, 0) \le b + ka,\\
& \forall\, 1 \le j \le i \le k:\ \alpha_j, d_j, a, b, r_{j,i} \ge 0.
\end{aligned}$$

However, it is clear that the above optimization program and the factor-revealing linear program in Theorem A are equivalent. This completes the proof of this lemma. □

The above lemma and Theorem A give us the following corollary, which will be used in the next section.

COROLLARY 8. *There is a* $(1, 2)$*-approximation algorithm for the linear-cost FLP.*

It is worth mentioning that Algorithm A can also be generalized for the linear-cost FLP. The only trick is to scale up both a and b in the first phase by a factor of δ, and scale them both down in the second phase. The rest of the proof is almost the same as the proof of Lemma 6.

5. The SCFLP. In this section we will show how the SCFLP can be reduced to the linear-cost FLP. In section 5.1 we define the concept of reduction between FLPs. We will use this concept in sections 5.2 and 5.3 to obtain approximation algorithms for SCFLP and a generalization of SCFLP and the concave-cost FLP.

5.1. Reduction between FLPs.

DEFINITION 5. *A reduction from an FLP* $\mathcal{A}$ *to another FLP* $\mathcal{B}$ *is a polynomial-time procedure* R *that maps every instance* $\mathcal{I}$ *of* $\mathcal{A}$ *to an instance* $R(\mathcal{I})$ *of* $\mathcal{B}$*. This procedure is called a* (σ_f, σ_c)*-reduction if the following conditions hold:*

1. *For any instance* $\mathcal{I}$ *of* $\mathcal{A}$ *and any feasible solution for* $\mathcal{I}$ *with facility cost* $F^*_{\mathcal{A}}$ *and connection cost* $C^*_{\mathcal{A}}$*, there is a corresponding solution for the instance* $R(\mathcal{I})$ *with facility cost* $F^*_{\mathcal{B}} \le \sigma_f F^*_{\mathcal{A}}$ *and connection cost* $C^*_{\mathcal{B}} \le \sigma_c C^*_{\mathcal{A}}$*.*
2. *For any feasible solution for the instance* $R(\mathcal{I})$*, there is a corresponding feasible solution for* $\mathcal{I}$ *whose total cost is at most as much as the total cost of the original solution for* $R(\mathcal{I})$*. In other words, cost of the instance* $R(\mathcal{I})$ *is an overestimate of cost of the instance* $\mathcal{I}$*.*

THEOREM 9. *If there is a* (σ_f, σ_c)*-reduction from an FLP* $\mathcal{A}$ *to another FLP* $\mathcal{B}$*, and a* (γ_f, γ_c)*-approximation algorithm for* $\mathcal{B}$*, then there is a* $(\gamma_f\sigma_f, \gamma_c\sigma_c)$*-approximation algorithm for* $\mathcal{A}$*.*

Proof. On an instance $\mathcal{I}$ of the problem $\mathcal{A}$, we compute $R(\mathcal{I})$, run the (γ_f, γ_c)-approximation algorithm for $\mathcal{B}$ on $R(\mathcal{I})$, and output the corresponding solution for $\mathcal{I}$. In order to see why this is a $(\gamma_f\sigma_f, \gamma_c\sigma_c)$-approximation algorithm for $\mathcal{A}$, let SOL denote an arbitrary solution for $\mathcal{I}$, ALG denote the solution that the above algorithm finds, and $F^*_{\mathcal{P}}$ and $C^*_{\mathcal{P}}$ ($F^{ALG}_{\mathcal{P}}$ and $C^{ALG}_{\mathcal{P}}$, respectively) denote the facility and connection costs of SOL (ALG, respectively) when viewed as a solution for the problem $\mathcal{P}$ ($\mathcal{P} = \mathcal{A}, \mathcal{B}$). By the definition of (σ_f, σ_c)-reductions and (γ_f, γ_c)-approximation algorithms, we have

$$F^{ALG}_{\mathcal{A}} + C^{ALG}_{\mathcal{A}} \le F^{ALG}_{\mathcal{B}} + C^{ALG}_{\mathcal{B}} \le \gamma_f F^*_{\mathcal{B}} + \gamma_c C^*_{\mathcal{B}} \le \gamma_f\sigma_f F^*_{\mathcal{A}} + \gamma_c\sigma_c C^*_{\mathcal{A}},$$

which completes the proof of the lemma. □

We will see examples of reductions in the rest of this paper.

5.2. The SCFLP. In this subsection, we give a 2-approximation algorithm for the soft-capacitated FLP by reducing it to the linear-cost FLP.

THEOREM 10. *There is a* 2*-approximation algorithm for the SCFLP.*

Proof. We use the following reduction. Construct an instance of the linear-cost FLP, where we have the same sets of facilities and clients. The connection costs remain the same. However, the facility cost of the ith facility is $(1 + \frac{k-1}{u_i})f_i$ if $k \geq 1$ and 0 if $k = 0$. Note that, for every $k \geq 1$, $\lceil \frac{k}{u_i} \rceil \leq 1 + \frac{k-1}{u_i} \leq 2 \cdot \lceil \frac{k}{u_i} \rceil$. Therefore, it is easy to see that this reduction is a $(2, 1)$-reduction. By Corollary 8, there is a $(1, 2)$-approximation algorithm for the linear-cost FLP, which together with Theorem 9 completes the proof. □

Furthermore, we now illustrate that the following natural LP formulation of the SCFLP has an integrality gap of 2. This means that we cannot obtain a better approximation ratio using this LP relaxation as the lower bound.

$$\begin{aligned}
\text{minimize } & \sum_{i \in \mathcal{F}} f_i y_i + \sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{C}} c_{ij} x_{ij} \\
\text{subject to } & \forall\, i \in \mathcal{F}, j \in \mathcal{C} :\ x_{ij} \leq y_i, \\
& \forall\, i \in \mathcal{F} :\ \sum_{j \in \mathcal{C}} x_{ij} \leq u_i y_i, \\
& \forall\, j \in \mathcal{C} :\ \sum_{i \in \mathcal{F}} x_{ij} = 1, \\
& \forall\, i \in \mathcal{F}, j \in \mathcal{C} :\ x_{ij} \in \{0, 1\}, \qquad (16) \\
& \forall\, i \in \mathcal{F} :\ y_i \text{ is a nonnegative integer.} \qquad (17)
\end{aligned}$$

In a natural linear program relaxation, we replace the constraints (16) and (17) by $x_{ij} \geq 0$ and $y_i \geq 0$. Here we see that even if we relax only constraint (17), the integrality gap is 2. Consider an instance of the SCFLP that consists of only one potential facility i and $k \geq 2$ clients. Assume that the capacity of facility i is $k - 1$, the facility cost is 1, and all connection costs are 0. It is clear that the optimal integral solution has cost 2. However, after relaxing constraint (17), the optimal fractional solution has cost $1 + \frac{1}{k-1}$. Therefore, the integrality gap between the integer program and its relaxation is $\frac{2(k-1)}{k}$, which tends to 2 as k tends to infinity.

5.3. The concave SCFLP. In this subsection, we consider a common generalization of the SCFLP and the concave-cost FLP. This problem, which we refer to as the *concave SCFLP*, is the same as the SCFLP except that if $r \geq 0$ copies of facility i are open, then the facility cost is $g_i(r)a_i$, where $g_i(r)$ is a given concave increasing function of r. In other words, the concave SCFLP is a special case of the universal FLP in which the facility cost functions are of the form $f_i(x) = a_i g_i(\lceil x/u_i \rceil)$ for constants a_i, u_i and a concave increasing function g_i. It is also a special case of the so-called staircase cost FLP [11]. On the other hand, it is a common generalization of the SCFLP (when $g_i(r) = r$) and the concave-cost FLP (when $u_i = 1$ for all i). The concave-cost FLP is a special case of the universal FLP in which facility cost functions are required to be concave and increasing (see [9]). The main result of this subsection is the following.

THEOREM 11. *The concave SCFLP is* $(\max_{i \in \mathcal{F}} \frac{g_i(2)}{g_i(1)}, 1)$*-reducible to the linear-cost FLP.*

The above theorem is established by the following lemmas, which show the reductions between the concave SCFLP, the concave-cost FLP, and the linear-cost FLP. Notice that $\max_{i\in\mathcal{F}} \frac{g_i(2)}{g_i(1)} \le 2$.

LEMMA 12. *The concave soft-capacitated FLP is* $(\max_{i\in\mathcal{F}} \frac{g_i(2)}{g_i(1)}, 1)$ *reducible to the concave-cost FLP.*

Proof. Given an instance $\mathcal{I}$ of the concave SCFLP, where the facility cost function of the facility i is $f_i(k) = g_i(\lceil k/u_i \rceil)a_i$, we construct an instance $R(\mathcal{I})$ of the concave-cost FLP as follows. We have the same sets of facilities and clients and the same connection costs as in $\mathcal{I}$. The facility cost function of the ith facility is given by

$$f_i'(k) = \begin{cases} \left(g_i(r) + (g_i(r+1) - g_i(r))(\frac{k-1}{u_i} - r + 1)\right) a_i & \text{if } k > 0,\ r := \lceil k/u_i \rceil, \\ 0 & \text{if } k = 0. \end{cases}$$

Concavity of g_i implies that the above function is also concave, and therefore $R(\mathcal{I})$ is an instance of the concave-cost FLP. Also, it is easy to see from the above definition that

$$g_i(\lceil k/u_i \rceil)a_i \le f_i'(k) \le g_i(\lceil k/u_i \rceil + 1)a_i.$$

By the concavity of the function g_i, we have $\frac{g_i(r+1)}{g_i(r)} \le \frac{g_i(2)}{g_i(1)}$ for every $r \ge 1$. Therefore, for every facility i and number k,

$$f_i(k) \le f_i'(k) \le \frac{g_i(2)}{g_i(1)} f_i(k).$$

This completes the proof of the lemma. ∎

Now, we will show a simple $(1,1)$-reduction from the concave-cost FLP to the linear-cost FLP. This, together with the above lemma, reduces the concave SCFLP to the linear-cost FLP.

LEMMA 13. *There is a* $(1,1)$*-reduction from the concave-cost FLP to the linear-cost FLP.*

Proof. Given an instance $\mathcal{I}$ of concave-cost FLP, we construct an instance $R(\mathcal{I})$ of linear-cost FLP as follows: Corresponding to each facility i in $\mathcal{I}$ with facility cost function $f_i(k)$, we put n copies of this facility in $R(\mathcal{I})$ (where n is the number of clients) and let the facility cost function of the lth copy be

$$f_i^{(l)}(k) = \begin{cases} f_i(l) + (f_i(l) - f_i(l-1))(k-l) & \text{if } k > 0, \\ 0 & \text{if } k = 0. \end{cases}$$

In other words, the facility cost function is the line that passes through the points $(l-1, f(l-1))$ and $(l, f(l))$. The set of clients and the connection costs between clients and facilities are unchanged. We prove that this reduction is a $(1,1)$-reduction.

For any feasible solution SOL for $\mathcal{I}$, we can construct a feasible solution SOL' for $R(\mathcal{I})$ as follows. If a facility i is open and k clients are connected to it in SOL, we open the kth copy of the corresponding facility in $R(\mathcal{I})$ and connect the clients to it. Since $f_i(k) = f_i^{(k)}(k)$, the facility and connection costs of SOL' are the same as those of SOL.

Conversely, consider an arbitrary feasible solution SOL for $R(\mathcal{I})$. We construct a solution SOL' for $\mathcal{I}$ as follows. For any facility i, if at least one of the copies of i

is open in SOL, we open i and connect to it all clients that were served by a copy of i in SOL. We show that this does not increase the total cost of the solution as follows. Assume that the l_1th, l_2th, ..., and l_sth copies of i were open in SOL, serving $k_1, k_2, \ldots,$ and k_s clients, respectively. By the concavity of f_i and the fact that $f_i^{(l)}(k) \geq f_i^{(k)}(k) = f_i(k)$ for every l, we have

$$f_i(k_1 + \cdots + k_s) \leq f_i(k_1) + \cdots + f_i(k_s) \leq f_i^{(l_1)}(k_1) + \cdots + f_i^{(l_s)}(k_s).$$

This shows that the facility cost of SOL' is at most the facility cost of SOL. □

6. Conclusion. We have obtained the best approximation ratios for two well-studied facility location problems, 1.52 for the UFLP and 2 for the SCFLP, respectively. The approximation ratio for the UFLP almost matches the lower bound of 1.463, and the approximation ratio for the SCFLP achieves the integrality gap of the standard LP relaxation of the problem. An interesting open question in this area is how to close the gap between 1.52 and 1.463 for the UFLP.

Although the performance guarantee of our algorithm for the UFLP is very close to the lower bound of 1.463, it would be nice to show that the bound of 1.52 is actually tight. In [12], it was shown that a solution to the factor-revealing linear program for the JMS algorithm provides a tight bound on the performance guarantee of the JMS algorithm. It is reasonable to expect that a solution to linear program (LP2) may also be used to construct a tight example for our 1.52-approximation algorithm. However, we were unsuccessful in constructing such an example.

Our results (Theorem 1 and Lemma 4) for the UFLP and/or the idea of bifactor reduction have been used to get the currently best known approximations ratios for several multilevel FLPs [1, 26]. Since the UFLP is the most basic FLP, we expect to see more applications of our results.

Appendix. Proof of Lemma 4.

Proof. By doubling a feasible solution of the factor-revealing program (LP1) (as in the proof of Lemma 12 in [13]), it is easy to show that for every k, $z_k \leq z_{2k}$. Therefore, without loss of generality, we can assume that k is sufficiently large.

Consider a feasible solution of the factor-revealing linear program. Let $x_{j,i} := \max(r_{j,i} - d_j, 0)$. Inequality (4) of the factor-revealing linear program implies that for every $i \leq i'$,

$$(i' - i + 1)\alpha_i - f \leq \sum_{j=i}^{i'} d_j - \sum_{j=1}^{i-1} x_{j,i}. \tag{18}$$

Now, we define l_i as follows:

$$l_i = \begin{cases} p_2 k & \text{if } i \leq p_1 k, \\ k & \text{if } i > p_1 k, \end{cases}$$

where p_1 and p_2 are two constants with $p_1 < p_2$ that will be fixed later. Consider inequality (18) for every $i \leq p_2 k$ and $i' = l_i$:

$$(l_i - i + 1)\alpha_i - f \leq \sum_{j=i}^{l_i} d_j - \sum_{j=1}^{i-1} x_{j,i}. \tag{19}$$

For every $i = 1, \ldots, k$, we define θ_i as follows. Here p_3 and p_4 are two constants with $p_1 < p_3 < 1 - p_3 < p_2$ and $p_4 \leq 1 - p_2$ that will be fixed later.

$$\theta_i = \begin{cases} \frac{1}{l_i - i + 1} & \text{if } i \leq p_3 k, \\ \frac{1}{(1-p_3)k} & \text{if } p_3 k < i \leq (1-p_3)k, \\ \frac{p_4 k}{(k-i)(k-i+1)} & \text{if } (1-p_3)k < i \leq p_2 k, \\ 0 & \text{if } i > p_2 k. \end{cases} \tag{20}$$

By multiplying both sides of inequality (19) by θ_i and adding up this inequality for $i = 1, \ldots, p_1 k$, $i = p_1 k + 1, \ldots, p_3 k$, $i = p_3 k + 1 \ldots, (1 - p_3)k$, and $i = (1 - p_3)k + 1, \ldots, p_2 k$, we get the following inequalities:

$$\sum_{i=1}^{p_1 k} \alpha_i - \left(\sum_{i=1}^{p_1 k} \theta_i \right) f \leq \sum_{i=1}^{p_1 k} \sum_{j=i}^{p_2 k} \frac{d_j}{p_2 k - i + 1} - \sum_{i=1}^{p_1 k} \sum_{j=1}^{i-1} \frac{\max(r_{j,i} - d_j, 0)}{p_2 k - i + 1}, \tag{21}$$

$$\sum_{i=p_1 k+1}^{p_3 k} \alpha_i - \left(\sum_{i=p_1 k+1}^{p_3 k} \theta_i \right) f \leq \sum_{i=p_1 k+1}^{p_3 k} \sum_{j=i}^{k} \frac{d_j}{k - i + 1} - \sum_{i=p_1 k+1}^{p_3 k} \sum_{j=1}^{i-1} \frac{\max(r_{j,i} - d_j, 0)}{k - i + 1}, \tag{22}$$

$$\begin{aligned} &\sum_{i=p_3 k+1}^{(1-p_3)k} \frac{k - i + 1}{(1-p_3)k} \alpha_i - \left(\sum_{i=p_3 k+1}^{(1-p_3)k} \theta_i \right) f \\ &\quad \leq \sum_{i=p_3 k+1}^{(1-p_3)k} \sum_{j=i}^{k} \frac{d_j}{(1-p_3)k} - \sum_{i=p_3 k+1}^{(1-p_3)k} \sum_{j=1}^{i-1} \frac{\max(r_{j,i} - d_j, 0)}{(1-p_3)k}, \end{aligned} \tag{23}$$

$$\begin{aligned} \sum_{i=(1-p_3)k+1}^{p_2 k} \frac{p_4 k}{k - i} \alpha_i - \left(\sum_{i=(1-p_3)k+1}^{p_2 k} \theta_i \right) f \leq &\sum_{i=(1-p_3)k+1}^{p_2 k} \sum_{j=i}^{k} \frac{p_4 k d_j}{(k-i)(k-i+1)} \\ &- \sum_{i=(1-p_3)k+1}^{p_2 k} \sum_{j=1}^{i-1} \frac{p_4 k \max(r_{j,i} - d_j, 0)}{(k-i)(k-i+1)}. \end{aligned} \tag{24}$$

We define $s_i := \max_{l \geq i}(\alpha_l - d_l)$. Using this definition and inequalities (2) and (3) of the factor-revealing linear program (LP1), we obtain

$$\forall i : \ r_{j,i} \geq s_i - d_j \Longrightarrow \forall i : \ \max(r_{j,i} - d_j, 0) \geq \max(s_i - 2d_j, 0), \tag{25}$$

$$\forall i : \ \alpha_i \leq s_i + d_i, \tag{26}$$

$$s_1 \geq s_2 \geq \cdots \geq s_k \ (\geq 0). \tag{27}$$

We assume $s_k \geq 0$ here because, if instead $\alpha_k < d_k$, we can always set α_k equal to d_k without violating any constraint in the factor-revealing (LP1) and increase z_k.

Inequality (26) and $p_4 \le 1 - p_2$ imply

$$\sum_{i=p_3k+1}^{(1-p_3)k} \left(1 - \frac{k-i+1}{(1-p_3)k}\right)\alpha_i + \sum_{i=(1-p_3)k+1}^{p_2k} \left(1 - \frac{p_4k}{k-i}\right)\alpha_i + \sum_{i=p_2k+1}^{k} \alpha_i$$
$$\le \sum_{i=p_3k+1}^{(1-p_3)k} \frac{i-p_3k-1}{(1-p_3)k}(s_i+d_i) + \sum_{i=(1-p_3)k+1}^{p_2k} \left(1 - \frac{p_4k}{k-i}\right)(s_i+d_i) + \sum_{i=p_2k+1}^{k} (s_i+d_i). \tag{28}$$

Let $\zeta := \sum_{i=1}^{k} \theta_i$. Thus,

$$\begin{aligned}\zeta &= \sum_{i=1}^{p_1k} \frac{1}{p_2k-i+1} + \sum_{i=p_1k+1}^{p_3k} \frac{1}{k-i+1} + \sum_{i=p_3k+1}^{(1-p_3)k} \frac{1}{(1-p_3)k} \\ &\quad + \sum_{i=(1-p_3)k+1}^{p_2k} \left(\frac{p_4k}{k-i} - \frac{p_4k}{k-i+1}\right) \\ &= \ln\left(\frac{p_2}{p_2-p_1}\right) + \ln\left(\frac{1-p_1}{1-p_3}\right) + \frac{1-2p_3}{1-p_3} + \frac{p_4}{1-p_2} - \frac{p_4}{p_3} + o(1). \end{aligned} \tag{29}$$

By adding the inequalities (21), (22), (23), (24), (28) and using (25), (27), and the fact that $\max(x,0) \ge \delta x$ for every $0 \le \delta \le 1$, we obtain

$$\begin{aligned}&\sum_{i=1}^{k} \alpha_i - \zeta f \\ &\le \sum_{i=1}^{p_1k}\sum_{j=i}^{p_2k} \frac{d_j}{p_2k-i+1} - \sum_{i=1}^{p_1k}\sum_{j=1}^{i-1} \frac{s_i-2d_j}{2(p_2k-i+1)} \\ &\quad + \sum_{i=p_1k+1}^{p_3k}\sum_{j=i}^{k} \frac{d_j}{k-i+1} - \sum_{i=p_1k+1}^{p_3k}\sum_{j=1}^{i-1} \frac{s_i-2d_j}{k-i+1} \\ &\quad + \sum_{i=p_3k+1}^{(1-p_3)k}\sum_{j=i}^{k} \frac{d_j}{(1-p_3)k} - \sum_{i=p_3k+1}^{(1-p_3)k}\sum_{j=1}^{i-1} \frac{s_i-2d_j}{(1-p_3)k} \\ &\quad + \sum_{i=(1-p_3)k+1}^{p_2k}\sum_{j=i}^{k} \frac{p_4kd_j}{(k-i)(k-i+1)} - \sum_{i=(1-p_3)k+1}^{p_2k}\sum_{j=1}^{i-1} \frac{p_4k\max(s_{p_2k+1}-2d_j,0)}{(k-i)(k-i+1)} \\ &\quad + \sum_{i=p_3k+1}^{(1-p_3)k} \frac{i-p_3k-1}{(1-p_3)k}(s_i+d_i) \\ &\quad + \sum_{i=(1-p_3)k+1}^{p_2k} \left(1 - \frac{p_4k}{k-i}\right)(s_i+d_i) + \sum_{i=p_2k+1}^{k} (s_{p_2k+1}+d_i) \\ &= \sum_{j=1}^{p_2k}\sum_{i=1}^{\min(j,p_1k)} \frac{d_j}{p_2k-i+1} - \sum_{i=1}^{p_1k} \frac{i-1}{2(p_2k-i+1)}s_i + \sum_{j=1}^{p_1k}\sum_{i=j+1}^{p_1k} \frac{d_j}{p_2k-i+1} \end{aligned}$$

$$
\begin{aligned}
&+\sum_{j=p_1k+1}^{k}\sum_{i=p_1k+1}^{\min(j,p_3k)}\frac{d_j}{k-i+1}-\sum_{i=p_1k+1}^{p_3k}\frac{i-1}{k-i+1}s_i+\sum_{j=1}^{p_3k}\sum_{i=\max(j,p_1k)+1}^{p_3k}\frac{2d_j}{k-i+1}\\
&+\sum_{j=p_3k+1}^{k}\sum_{i=p_3k+1}^{\min(j,(1-p_3)k)}\frac{d_j}{(1-p_3)k}-\sum_{i=p_3k+1}^{(1-p_3)k}\frac{i-1}{(1-p_3)k}s_i\\
&+\sum_{j=1}^{(1-p_3)k}\sum_{i=\max(j,p_3k)+1}^{(1-p_3)k}\frac{2d_j}{(1-p_3)k}\\
&+\sum_{j=(1-p_3)k+1}^{k}\sum_{i=(1-p_3)k+1}^{\min(j,p_2k)}\left(\frac{1}{k-i}-\frac{1}{k-i+1}\right)p_4kd_j\\
&-\sum_{j=1}^{p_2k}\sum_{i=\max(j,(1-p_3)k)+1}^{p_2k}p_4k\left(\frac{1}{k-i}-\frac{1}{k-i+1}\right)\max(s_{p_2k+1}-2d_j,0)\\
&+\sum_{i=p_3k+1}^{(1-p_3)k}\frac{i-p_3k-1}{(1-p_3)k}(s_i+d_i)+\sum_{i=(1-p_3)k+1}^{p_2k}\left(1-\frac{p_4k}{k-i}\right)(s_i+d_i)+\sum_{i=p_2k+1}^{k}d_i\\
&+(1-p_2)ks_{p_2k+1}\\
=&\sum_{j=1}^{p_2k}(\mathrm{H}_{p_2k}-\mathrm{H}_{p_2k-\min(j,p_1k)})d_j-\sum_{j=1}^{p_1k}\frac{j-1}{2(p_2k-j+1)}s_j+\sum_{j=1}^{p_1k}(\mathrm{H}_{p_2k-j}-\mathrm{H}_{(p_2-p_1)k})d_j\\
&+\sum_{j=p_1k+1}^{k}(\mathrm{H}_{(1-p_1)k}-\mathrm{H}_{k-\min(j,p_3k)})d_j\\
&-\sum_{j=p_1k+1}^{p_3k}\frac{j-1}{k-j+1}s_j+\sum_{j=1}^{p_3k}2(\mathrm{H}_{k-\max(j,p_1k)}-\mathrm{H}_{(1-p_3)k})d_j\\
&+\sum_{j=p_3k+1}^{k}\frac{\min(j,(1-p_3)k)-p_3k}{(1-p_3)k}d_j-\sum_{j=p_3k+1}^{(1-p_3)k}\frac{j-1}{(1-p_3)k}s_j\\
&+\sum_{j=1}^{(1-p_3)k}\frac{2((1-p_3)k-\max(j,p_3k))}{(1-p_3)k}d_j\\
&+\sum_{j=(1-p_3)k+1}^{k}\left(\frac{1}{k-\min(j,p_2k)}-\frac{1}{p_3k}\right)p_4kd_j\\
&-\sum_{j=1}^{p_2k}\left(\frac{p_4}{1-p_2}-\frac{p_4k}{k-\max(j,(1-p_3)k)}\right)\max(s_{p_2k+1}-2d_j,0)\\
&+\sum_{j=p_3k+1}^{(1-p_3)k}\frac{j-p_3k-1}{(1-p_3)k}(s_j+d_j)+\sum_{j=(1-p_3)k+1}^{p_2k}\left(1-\frac{p_4k}{k-j}\right)(s_j+d_j)
\end{aligned}
$$

$$
\begin{aligned}
&+\sum_{j=p_2k+1}^{k} d_j + (1-p_2)ks_{p_2k+1}\\
&\le \sum_{j=1}^{p_1k}\left(\mathrm{H}_{p_2k}-\mathrm{H}_{p_2k-j}+\mathrm{H}_{p_2k-j}-\mathrm{H}_{(p_2-p_1)k}+2\mathrm{H}_{(1-p_1)k}-2\mathrm{H}_{(1-p_3)k}+\frac{2(1-2p_3)}{1-p_3}\right)d_j\\
&\quad+\sum_{j=p_1k+1}^{p_3k}\left(\mathrm{H}_{p_2k}-\mathrm{H}_{(p_2-p_1)k}+\mathrm{H}_{(1-p_1)k}-\mathrm{H}_{k-j}+2\mathrm{H}_{k-j}-2\mathrm{H}_{(1-p_3)k}+\frac{2(1-2p_3)}{1-p_3}\right)d_j\\
&\quad+\sum_{j=p_3k+1}^{(1-p_3)k}\left(\mathrm{H}_{p_2k}-\mathrm{H}_{(p_2-p_1)k}+\mathrm{H}_{(1-p_1)k}-\mathrm{H}_{(1-p_3)k}+\frac{j-p_3k}{(1-p_3)k}\right.\\
&\qquad\qquad\left.+\frac{2((1-p_3)k-j)}{(1-p_3)k}+\frac{j-p_3k-1}{(1-p_3)k}\right)d_j\\
&\quad+\sum_{j=(1-p_3)k+1}^{p_2k}\left(\mathrm{H}_{p_2k}-\mathrm{H}_{(p_2-p_1)k}+\mathrm{H}_{(1-p_1)k}-\mathrm{H}_{(1-p_3)k}+\frac{1-2p_3}{1-p_3}\right.\\
&\qquad\qquad\left.+\frac{p_4k}{k-j}-\frac{p_4k}{p_3k}+\frac{(1-p_4)k-j}{k-j}\right)d_j\\
&\quad+\sum_{j=p_2k+1}^{k}\left(\mathrm{H}_{(1-p_1)k}-\mathrm{H}_{(1-p_3)k}+\frac{1-2p_3}{1-p_3}+\frac{p_4k}{(1-p_2)k}-\frac{p_4k}{p_3k}+1\right)d_j\\
&\quad-\sum_{j=1}^{p_3k}\left(\frac{p_4}{1-p_2}-\frac{p_4}{p_3}\right)\max(s_{p_2k+1}-2d_j,0)-\sum_{j=p_3k+1}^{(1-p_3)k}\left(\frac{p_4}{1-p_2}-\frac{p_4}{p_3}\right)(s_{p_2k+1}-2d_j)\\
&\quad-\sum_{j=1}^{p_1k}\frac{j-1}{2(p_2k-j+1)}s_j-\sum_{j=p_1k+1}^{p_3k}\frac{j-1}{k-j+1}s_j-\sum_{j=p_3k+1}^{(1-p_3)k}\frac{p_3k}{(1-p_3)k}s_j\\
&\quad+\sum_{j=(1-p_3)k+1}^{p_2k}\left(1-\frac{p_4k}{k-j}\right)s_j+(1-p_2)ks_{p_2k+1}.
\end{aligned}
\tag{30}
$$

Let us denote the coefficients of d_j in the above expression by λ_j. Therefore, we have

$$
\begin{aligned}
&\sum_{i=1}^{k}\alpha_i-\zeta f\\
&\quad\le \sum_{j=1}^{k}\lambda_j d_j-\sum_{j=1}^{p_1k}\frac{j-1}{2(p_2k-j+1)}s_j-\sum_{j=p_1k+1}^{p_3k}\frac{j-1}{k-j+1}s_j-\sum_{j=p_3k+1}^{(1-p_3)k}\frac{p_3k}{(1-p_3)k}s_j\\
&\qquad+\sum_{j=(1-p_3)k+1}^{p_2k}\left(1-\frac{p_4k}{k-j}\right)s_j+\left(1-p_2-(1-2p_3)\left(\frac{p_4}{1-p_2}-\frac{p_4}{p_3}\right)\right)ks_{p_2k+1}\\
&\qquad-\left(\frac{p_4}{1-p_2}-\frac{p_4}{p_3}\right)\sum_{j=1}^{p_3k}\max(s_{p_2k+1}-2d_j,0),
\end{aligned}
\tag{31}
$$

where

$$\lambda_j := \begin{cases} \ln\left(\frac{p_2}{p_2-p_1}\right) + 2\ln\left(\frac{1-p_1}{1-p_3}\right) + \frac{2(1-2p_3)}{1-p_3} + o(1) \\ \qquad\qquad \text{if } 1 \le j \le p_1 k, \\ \ln\left(\frac{p_2}{p_2-p_1}\right) + \ln\left(\frac{1-p_1}{1-p_3}\right) + \frac{2(1-2p_3)}{1-p_3} + \mathrm{H}_{k-j} - \mathrm{H}_{(1-p_3)k} + o(1) \\ \qquad\qquad \text{if } p_1 k < j \le p_3 k, \\ \ln\left(\frac{p_2}{p_2-p_1}\right) + \ln\left(\frac{1-p_1}{1-p_3}\right) + \frac{2(1-2p_3)}{1-p_3} + \frac{2p_4}{1-p_2} - \frac{2p_4}{p_3} + o(1) \\ \qquad\qquad \text{if } p_3 k < j \le (1-p_3)k, \\ \ln\left(\frac{p_2}{p_2-p_1}\right) + \ln\left(\frac{1-p_1}{1-p_3}\right) + \frac{1-2p_3}{1-p_3} + 1 - \frac{p_4}{p_3} + o(1) \\ \qquad\qquad \text{if } (1-p_3)k < j \le p_2 k, \\ \ln\left(\frac{1-p_1}{1-p_3}\right) + \frac{1-2p_3}{1-p_3} + 1 + \frac{p_4}{1-p_2} - \frac{p_4}{p_3} + o(1) \\ \qquad\qquad \text{if } p_2 k < j \le k. \end{cases}$$

For every $j \le p_3 k$, we have

(32)
$$\lambda_{(1-p_3)k} - \lambda_j \le \frac{2p_4}{1-p_2} - \frac{2p_4}{p_3} \Rightarrow \delta_j := (\lambda_{(1-p_3)k} - \lambda_j) \Big/ \left(\frac{2p_4}{1-p_2} - \frac{2p_4}{p_3}\right) \le 1.$$

Also, if we choose p_1, p_2, p_3, p_4 in such a way that

$$\ln\left(\frac{1-p_1}{1-p_3}\right) < \frac{2p_4}{1-p_2} - \frac{2p_4}{p_3}, \tag{33}$$

then for every $j \le p_3 k$, $\lambda_j \le \lambda_{(1-p_3)k}$ and therefore $\delta_j \ge 0$. Then, since $0 \le \delta_j \le 1$, we can replace $\max(s_{p_2k+1} - 2d_j, 0)$ by $\delta_j(s_{p_2k+1} - 2d_j)$ in (31). This gives us

$$\begin{aligned} &\sum_{i=1}^{k} \alpha_i - \zeta f \\ &\le \sum_{j=1}^{k} \lambda_j d_j - \sum_{j=1}^{p_1 k} \frac{j-1}{2(p_2 k - j + 1)} s_j - \sum_{j=p_1k+1}^{p_3 k} \frac{j-1}{k-j+1} s_j - \sum_{j=p_3k+1}^{(1-p_3)k} \frac{p_3 k}{(1-p_3)k} s_j \\ &\quad + \sum_{j=(1-p_3)k+1}^{p_2 k} \left(1 - \frac{p_4 k}{k-j}\right) s_j + \left(1 - p_2 - (1-2p_3)\left(\frac{p_4}{1-p_2} - \frac{p_4}{p_3}\right)\right) k s_{p_2k+1} \\ &\quad - \frac{1}{2} \sum_{j=1}^{p_3 k} (\lambda_{(1-p_3)k} - \lambda_j)(s_{p_2k+1} - 2d_j). \end{aligned} \tag{34}$$

Let μ_j denote the coefficient of s_j in the above expression. Therefore the above inequality can be written as

$$\sum_{i=1}^{k} \alpha_i - \zeta f \le \lambda_{(1-p_3)k} \sum_{j=1}^{(1-p_3)k} d_j + \sum_{j=(1-p_3)k+1}^{k} \lambda_j d_j + \sum_{j=1}^{p_2 k+1} \mu_j s_j, \tag{35}$$

where

$$\mu_j = \begin{cases} -\dfrac{j-1}{2(p_2k-j+1)} & \text{if } 1 \le j \le p_1k, \\ -\dfrac{j-1}{k-j+1} & \text{if } p_1k < j \le p_3k, \\ -\dfrac{p_3}{1-p_3} & \text{if } p_3k < j \le (1-p_3)k, \\ 1-\dfrac{p_4k}{k-j} & \text{if } (1-p_3)k < j \le p_2k, \end{cases} \tag{36}$$

and

(37)

$$\begin{aligned} &\mu_{p_2k+1} \\ &= \left(1-p_2-(1-2p_3)\left(\frac{p_4}{1-p_2}-\frac{p_4}{p_3}\right)\right)k - \frac{1}{2}\lambda_{(1-p_3)k}p_3k + \frac{1}{2}\sum_{j=1}^{p_3k}\lambda_j \\ &= \left(1-p_2-(1-2p_3)\left(\frac{p_4}{1-p_2}-\frac{p_4}{p_3}\right)\right)k - \frac{1}{2}\lambda_{(1-p_3)k}p_3k \\ &\quad + \frac{p_1k}{2}\left(\ln\left(\frac{p_2}{p_2-p_1}\right)+2\ln\left(\frac{1-p_1}{1-p_3}\right)+\frac{2(1-2p_3)}{1-p_3}+o(1)\right) \\ &\quad + \frac{(p_3-p_1)k}{2}\left(\ln\left(\frac{p_2}{p_2-p_1}\right)+\ln\left(\frac{1-p_1}{1-p_3}\right)+\frac{2(1-2p_3)}{1-p_3}+o(1)\right) \\ &\quad + \frac{1}{2}\sum_{j=p_1k+1}^{p_3k}\sum_{i=(1-p_3)k+1}^{k-j}\frac{1}{i} \\ &= \left(\ln\left(\frac{1-p_1}{1-p_3}\right)+2-2p_2-p_3+p_1-2(1-p_3)\left(\frac{p_4}{1-p_2}-\frac{p_4}{p_3}\right)+o(1)\right)\frac{k}{2}. \end{aligned}$$

Now, if we pick p_1, p_2, p_3, p_4 in such a way that $\lambda_j \le \gamma$ for every $j \ge (1-p_3)k$, i.e.,

$$\ln\left(\frac{p_2}{p_2-p_1}\right)+\ln\left(\frac{1-p_1}{1-p_3}\right)+\frac{2(1-2p_3)}{1-p_3}+\frac{2p_4}{1-p_2}-\frac{2p_4}{p_3} < \gamma, \tag{38}$$

$$\ln\left(\frac{p_2}{p_2-p_1}\right)+\ln\left(\frac{1-p_1}{1-p_3}\right)+\frac{1-2p_3}{1-p_3}+1-\frac{p_4}{p_3} < \gamma, \tag{39}$$

and

$$\ln\left(\frac{1-p_1}{1-p_3}\right)+\frac{1-2p_3}{1-p_3}+1+\frac{p_4}{1-p_2}-\frac{p_4}{p_3} < \gamma, \tag{40}$$

then the term $\lambda_{(1-p_3)k}\sum_{j=1}^{(1-p_3)k} d_j + \sum_{j=(1-p_3)k+1}^{k}\lambda_j d_j$ on the right-hand side of (35) is at most $\gamma\sum_{j=1}^{k} d_j$. Also, if for every $i \le p_2k+1$ we have

$$\mu_1+\mu_2+\cdots+\mu_i \le 0, \tag{41}$$

then by inequality (27) we have $\sum_{j=1}^{p_2k+1} \mu_j s_j \le 0$. Therefore, if p_1, p_2, p_3, p_4 are chosen in such a way that, in addition to the above inequalities, we have

$$\ln\left(\frac{p_2}{p_2-p_1}\right)+\ln\left(\frac{1-p_1}{1-p_3}\right)+\frac{1-2p_3}{1-p_3}+\frac{p_4}{1-p_2}-\frac{p_4}{p_3}<1.11, \tag{42}$$

then inequality (35) can be written as

$$\sum_{i=1}^{k}\alpha_i - 1.11f \le \gamma\sum_{j=1}^{k} d_j, \tag{43}$$

which shows that the solution of the maximization program (LP1) is at most γ. From (36), it is clear that $\mu_j \le 0$ for every $j \le (1-p_3)k$ and $\mu_j \ge 0$ for every $(1-p_3)k \le j \le p_2k$. Therefore, it is enough to check inequality (41) for $i = p_2k$ and $i = p_2k+1$. We have

$$\begin{aligned}
\sum_{j=1}^{p_2k}\mu_j &= -\sum_{j=1}^{p_1k}\frac{p_2k-p_2k+j-1}{2(p_2k-j+1)} - \sum_{j=p_1k+1}^{p_3k}\frac{k-k+j-1}{k-j+1} - \frac{p_3(1-2p_3)k}{1-p_3} \\
&\quad + (p_2-1+p_3)k - \sum_{j=(1-p_3)k+1}^{p_2k}\frac{p_4k}{k-j} \\
&= -\frac{p_2k}{2}(\mathrm{H}_{p_2k}-\mathrm{H}_{(p_2-p_1)k}) + \frac{p_1k}{2} - k(\mathrm{H}_{(1-p_1)k}-\mathrm{H}_{(1-p_3)k}) + (p_3-p_1)k \\
&\quad - \frac{p_3(1-2p_3)k}{1-p_3} + (p_2-1+p_3)k - p_4k(\mathrm{H}_{p_3k}-\mathrm{H}_{(1-p_2)k}) \\
&= \left(-\frac{p_1}{2}+p_2+2p_3-1-\frac{p_2}{2}\ln\left(\frac{p_2}{p_2-p_1}\right)-\ln\left(\frac{1-p_1}{1-p_3}\right)-\frac{p_3(1-2p_3)}{1-p_3}\right. \\
&\quad \left. - p_4\ln\left(\frac{p_3}{1-p_2}\right)+o(1)\right)k.
\end{aligned} \tag{44}$$

Therefore, inequality (41) is equivalent to the following two inequalities:

$$-\frac{p_1}{2}+p_2+2p_3-1-\frac{p_2}{2}\ln\left(\frac{p_2}{p_2-p_1}\right)-\ln\left(\frac{1-p_1}{1-p_3}\right)-\frac{p_3(1-2p_3)}{1-p_3}-p_4\ln\left(\frac{p_3}{1-p_2}\right)<0, \tag{45}$$

$$\begin{aligned}
&-\frac{p_1}{2}+p_2+2p_3-1-\frac{p_2}{2}\ln\left(\frac{p_2}{p_2-p_1}\right)-\ln\left(\frac{1-p_1}{1-p_3}\right)-\frac{p_3(1-2p_3)}{1-p_3}-p_4\ln\left(\frac{p_3}{1-p_2}\right) \\
&\quad +\frac{1}{2}\ln\left(\frac{1-p_1}{1-p_3}\right)+1-p_2-\frac{p_3}{2}+\frac{p_1}{2}-(1-p_3)\left(\frac{p_4}{1-p_2}-\frac{p_4}{p_3}\right)<0.
\end{aligned} \tag{46}$$

Now, it is enough to observe that if we let $p_1 = 0.225$, $p_2 = 0.791$, $p_3 = 0.30499$, $p_4 = 0.06984$, and $\gamma = 1.7764$, then $p_1 < p_3 < 1-p_3 < p_2$ and $p_4 < 1-p_2$, as specified earlier, and inequalities (33), (38), (39), (40), (42), (45), and (46) are all satisfied. Therefore, the solution of the optimization program (LP1) is at most $1.7764 < 1.78$. □

Acknowledgments. We would like to thank Asaf Levin for pointing out that our analysis of the 2-approximation algorithm for the SCFLP is tight. We also want

to mention that an idea for deriving better approximation factors for the UFLP using the $(1, 2)$ bifactor guarantee was independently proposed earlier by Kamal Jain in a private communication to the first author, and by the last two authors. We thank the anonymous referees for their helpful suggestions that significantly improved the exposition of our paper.

REFERENCES

[1] A. Ageev, Y. Ye, and J. Zhang, *Improved combinatorial approximation algorithms for the k-level facility location problem*, SIAM J. Discrete Math., 18 (2004), pp. 207–217.

[2] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit, *Local search heuristics for k-median and facility location problems*, SIAM J. Comput., 33 (2004), pp. 544–562.

[3] P. Bauer and R. Enders, *A capacitated facility location problem with integer decision variables*, in Proceedings of the International Symposium on Mathematical Programming (ISMP), 1997.

[4] M. Charikar and S. Guha, *Improved combinatorial algorithms for facility location and k-median problems*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, New York, 1999, IEEE Press, Piscataway, NJ, pp. 378–388.

[5] F. A. Chudak and D. B. Shmoys, *Improved approximation algorithms for the capacitated facility location problem*, in Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, 1999, SIAM, Philadelphia, 1999, pp. S875–S876.

[6] F. A. Chudak and D. B. Shmoys, *Improved approximation algorithms for the uncapacitated facility location problem*, SIAM J. Comput., 33 (2003), pp. 1–25.

[7] G. Cornuejols, G. L. Nemhauser, and L. A. Wolsey, *The uncapacitated facility location problem*, in Discrete Location Theory, P. Mirchandani and R. Francis, eds., John Wiley and Sons, New York, 1990, pp. 119–171.

[8] S. Guha and S. Khuller, *Greedy strikes back: Improved facility location algorithms*, J. Algorithms, 31 (1999), pp. 228–248.

[9] M. Hajiaghayi, M. Mahdian, and V. S. Mirrokni, *The facility location problem with general cost functions*, Networks, 42 (2003), pp. 42–47.

[10] D. S. Hochbaum, *Heuristics for the fixed cost median problem*, Math. Programming, 22 (1982), pp. 148–162.

[11] K. Holmberg, *Solving the staircase cost facility location problem with decomposition and piecewise linearization*, European J. Oper. Res., 74 (1994), pp. 41–61.

[12] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. V. Vazirani, *Approximation algorithms for facility location via dual fitting with factor-revealing LP*, J. ACM, 50 (2003), pp. 795–824.

[13] K. Jain, M. Mahdian, and A. Saberi, *A new greedy approach for facility location problems*, in Proceedings of the 11th Annual European Symposium on Algorithms (ESA), G. Di Battista and U. Zwick, eds., Lecture Notes in Comput. Sci. 2832, Springer, New York, 2003, pp. 409–421.

[14] K. Jain and V. V. Vazirani, *Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation*, J. ACM, 48 (2001), pp. 274–296.

[15] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, *Analysis of a local search heuristic for facility location problems*, J. Algorithms, 37 (2000), pp. 146–188.

[16] M. Mahdian, E. Markakis, A. Saberi, and V. V. Vazirani, *A greedy facility location algorithm analyzed using dual fitting*, in Proceedings of 5th International Workshop on Randomization and Approximation Techniques in Computer Science, Lecture Notes in Comput. Sci. 2129, Springer-Verlag, New York, Berlin, 2001, pp. 127–137.

[17] M. Mahdian and M. Pál, *Universal facility location*, in Proceedings of the 11th Annual European Symposium on Algorithms (ESA), 2003.

[18] M. Mahdian, Y. Ye, and J. Zhang, *Improved approximation algorithms for metric facility location problems*, in Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX 2002), Lecture Notes in Comput. Sci. 2462, Springer-Verlag, New York, Berlin, 2002, pp. 229–242.

[19] M. Mahdian, Y. Ye, and J. Zhang, *A 2-approximation algorithm for the soft-capacitated facility location problem*, in Proceedings of 6th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX 2003), Lecture Notes in Comput. Sci. 2764, Springer-Verlag, New York, Berlin, 2003, pp. 129–140.

[20] C. S. Revell and G. Laporte, *The plant location problem: New models and research prospects*, Oper. Res., 44 (1996), pp. 864–874.

[21] D. B. Shmoys, *Approximation algorithms for facility location problems*, in Approximation Algorithms for Combinatorial Optimization, K. Jansen and S. Khuller, eds., Lecture Notes in Comput. Sci. 1913, Springer, Berlin, 2000, pp. 27–33.

[22] D. B. Shmoys, E. Tardos, and K. I. Aardal, *Approximation algorithms for facility location problems*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El Paso, TX, 1997, ACM, New York, 1997, pp. 265–274.

[23] M. Sviridenko, *An improved approximation algorithm for the metric uncapacitated facility location problem*, in Integer Programming and Combinatorial Optimization: 9th International IPCO Conference, Cambridge, MA, 2002, W. J. Cook and A. S. Schulz, eds., Lecture Notes in Comput. Sci. 2337, Springer, New York, 2002, pp. 240–257.

[24] M. Thorup, *Quick and good facility location*, in Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, 2003, SIAM, Philadelphia, 2003, pp. 178–185.

[25] M. Thorup, *Quick k-median, k-center, and facility location for sparse graphs*, SIAM J. Comput., 34 (2005), pp. 405–432.

[26] J. Zhang, *Approximating the two-level facility location problem via a quasi-greedy approach*, Math. Programming, 108 (2006), pp. 159–176.

SIAM J. COMPUT.
Vol. 36, No. 2, pp. 433–456

AN UNCONDITIONAL LOWER BOUND ON THE TIME-APPROXIMATION TRADE-OFF FOR THE DISTRIBUTED MINIMUM SPANNING TREE PROBLEM*

MICHAEL ELKIN†

Abstract. The design of distributed approximation protocols is a relatively new and rapidly developing area of research. However, so far, little progress has been made in the study of the *hardness* of distributed approximation. In this paper we initiate the systematic study of this subject and show strong unconditional *lower bounds* on the *time-approximation trade-off* of the distributed *minimum spanning tree* problem, and show some of its variants.

Key words. minimum spanning tree, distributed algorithms, hardness of approximation

AMS subject classifications. 05C05, 05C85, 90C27

DOI. 10.1137/S0097539704441058

1. Introduction.

1.1. Distributed computing. Consider a synchronous network of processors with unbounded computational power, modeled by an undirected n-vertex graph. The initial knowledge of the processors (henceforth, vertices) is very limited. Specifically, each of them has its own *local* perspective of the network (henceforth, graph), which is confined to its immediate neighborhood. The vertices, however, have to compute some *global* function of the graph, such as its *minimum spanning tree* (henceforth, *MST*).

To this end, distributed algorithms (henceforth, *protocols*) are designed. There are several measures of efficiency of protocols, but in this paper we restrict our attention to the running time, defined as the worst-case number of rounds of distributed communication. On each round of communication at most B bits can be sent through each edge, where B is a parameter of the model. The running time efficiency measure of protocols naturally gives rise to a complexity measure of problems, called the time complexity.

The design of efficient protocols for this model, as well as proving lower bounds on their efficiency, is a lively area of study known as locality-sensitive distributed computing (henceforth, *distributed computing*); see [31] and references therein.

1.2. Distributed approximation and hardness of approximation. While traditionally the research in the area of distributed computing concentrated on designing protocols that solve the problem at hand *exactly*, some of the more recent research focuses on providing *approximate* solutions for various distributed problems. Most notably, several approximation protocols were recently devised for the minimum dominating set problem [22, 9, 25, 35], for the minimum edge-coloring problem [33, 20, 5], for the maximum matching problem [15, 21, 7], and for the distance estimation problem [10, 14]. Another relevant important direction is the study of distributed

*Received by the editors February 16, 2004; accepted for publication (in revised form) February 9, 2006; published electronically June 30, 2006. This work was done in the Department of Computer Science, Yale University, New Haven, CT, and in the School of Mathematics, Institute for Advanced Study, Princeton, NJ. A preliminary version of this paper was published as [12].

http://www.siam.org/journals/sicomp/36-2/44105.html

†Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel (elkinm@cs.bgu.ac.il).

algorithms for finding approximate solutions for positive linear programs initiated by Papadimitriou and Yannakakis in [34] and continued with the works of Bartal, Byers, and Raz [4] and Kuhn and Wattenhofer [26]. Recently Moscibroda and Wattenhofer [30] and Grandoni et al. [19] used techniques based on linear programming for developing distributed approximation algorithms for several natural problems.

However, the situation with lower bounds on approximability of distributed problems is by far less satisfactory. Specifically, the existing results on hardness of distributed approximation can be divided into two categories.

First, there are inapproximability results that are based on lower bounds on the time required for the *exact* solution of certain problems, and on integrality of the objective functions of these problems. For example, there is a fundamental result due to Linial [27] saying that 3-coloring an n-vertex ring requires $\Omega(\log^* n)$ time. In particular, it implies that any 3/2-approximation protocol for vertex-coloring problem requires $\Omega(\log^* n)$ time.

Second, there are inapproximability results that assume that the vertices are computationally limited, e.g., are allowed to perform at most polynomial in n number of operations. Obviously, under this assumption any NP-hardness inapproximability result immediately gives rise to an analogous result in the distributed model.

Note, however, that results of this sort are just somewhat different semantic interpretations of already known lower bounds, and as such, they provide no new insights that were not provided by the original lower bounds. Moreover, we believe that the study of the role of *locality* in distributed computing should be conducted in the cleanest possible model, and, consequently, restricting the computational power of the vertices is obstructive to the goals of this study (see [27, 31]).

To summarize, while sophisticated distributed approximation protocols were developed for various problems, so far no real progress has been made in the study of the *hardness of distributed approximation*. In this paper we initiate the systematic study of this subject. Specifically, we study the inapproximability of the distributed *MST* problem and show strong unconditional *lower bounds* on the *time-approximation trade-offs* for this problem and some of its variants.

1.3. Distributed *MST* problem. The (distributed) *MST* problem is one of the most important problems in the area of distributed computing and has been the subject of extensive research [16, 8, 18, 1, 17, 23, 32, 28, 11].

The most time-efficient protocol known for this problem was devised in [11], and its running time is $O(\mu(G,\omega) \cdot \log^3 n + \sqrt{\frac{n \log^* n}{B}} \log n)$, where $\mu(G,\omega)$ stands for the *MST*-radius of the weighted graph $(G = (V,E),\omega)$, $\omega : E \to R^+$. The definition of the *MST*-radius $\mu(G,\omega)$ is somewhat involved (see [11]), but for the rest of this discussion it is sufficient to keep in mind that for any graph (G,ω), $\mu(G,\omega) \leq \Lambda(G) \leq n$, where $\Lambda(G)$ stands for the *unweighted* diameter of the graph G.

On the negative side, Peleg and Rubinovich [32] have shown a lower bound of $\Omega(\frac{\sqrt{n}}{B})$ on the time complexity of the *MST* problem restricted to graphs of small diameter (at most $O(n^\delta)$ for an arbitrarily small positive $\delta > 0$).

In this paper we show that *approximating* the *MST* problem within a ratio H on graphs of small diameter requires $T = \Omega(\sqrt{\frac{n}{H \cdot B}})$ time. (The result applies to graphs of diameter $\Omega(n^\delta)$ for an arbitrarily small constant $\delta > 0$, exactly like the result of [32]; see section 2 for the formal definitions of the notions of *approximation* and *randomization* in this context.) In other words, we derive an unconditional lower bound on the time-approximation trade-off for the *MST* problem, specifically, $T^2 \cdot H =$

$\Omega(\frac{n}{B})$. Substituting $H = O(1)$ into this formula shows that approximating the *MST* problem within any constant factor requires $\Omega(\sqrt{\frac{n}{B}})$ time, improving the lower bound of [32] by a factor of $\sqrt{B}$ (recall that the lower bound of [32] applies only for the exact solution of the *MST* problem).

Moreover, our lower bound implies that for any $0 < \epsilon < 1$, approximating the *MST* problem within a factor of $\left(\frac{n}{B}\right)^{1-\epsilon}$ requires $\Omega\left(\left(\frac{n}{B}\right)^{\epsilon/2}\right)$ time. The latter means, in particular, that the $\left(\frac{n}{B}\right)^{1-\epsilon}$-approximate *MST* problem is *not a local* problem, i.e., cannot be solved in time polylogarithmic in n. This lower bound, like all the other lower bounds that we prove in this paper, applies even to *randomized* protocols. We also remark that our lower bounds, as well as the lower bounds of [32, 28], apply to the scenario when the vertices have their own distinct identity numbers at the beginning of computation.

1.4. Additional results. One direction of recent research on the distributed *MST* problem was to refine the lower bound of Peleg and Rubinovich [32] that applies to the *exact* computation of the *MST* on graphs G with diameter $\Lambda(G) = O(n^\delta)$, $0 < \delta < 1/2$, and to prove similar lower bounds for the *MST* problem restricted to graphs of even smaller diameter. Specifically, Peleg and Rubinovich themselves [32] have shown a lower bound of $\Omega(\frac{\sqrt{n}}{B \cdot \log n})$ for the *MST* problem restricted to graphs G of diameter $\Lambda(G) = O(\log n)$, and Lotker, Patt-Shamir, and Peleg [28] have shown lower bounds of $\Omega(\frac{n^{1/3}}{B})$ (resp., $\Omega(\frac{n^{1/4}}{\sqrt{B}})$) for the *MST* problem restricted to graphs of diameter $\Lambda(G) \leq 4$ (resp., $\Lambda(G) \leq 3$). For graphs of diameter $\Lambda(G) \leq 2$ (resp., $\Lambda(G) = 1$) upper bounds of $O(\log n)$ (resp., $O(\log \log n)$) were shown in [28] (resp., [29]). Recall that all the aforementioned lower bounds apply only to the exact *MST* problem.

In addition to the lower bound on the time-approximation trade-off for the general variant of the *MST* problem, in this paper we also show a lower bound on the time-approximation trade-off for the *MST* problem restricted to graphs of diameter $\Lambda(G) \leq \Lambda$, for $\Lambda = 3$ and all even Λ in the range $4 \leq \Lambda = O(\log n)$. Specifically, denoting the running time of an approximation protocol by T, and its approximation ratio by H, we show that $T^{2+\frac{2}{\Lambda-2}} \cdot H = \Omega(\frac{n}{\Lambda \cdot B})$.

Note that this result improves all the previous lower bounds for the exact computation of the *MST*. Specifically, it improves the result of [32] for $\Lambda = O(\log n)$ by a factor of $\sqrt{B \cdot \log n}$, and the results of [28] for $\Lambda = 4$ (resp., $\Lambda = 3$) by a factor of $B^{2/3}$ (resp., $B^{1/4}$). Moreover, our result gives rise to a lower bound of $\Omega((\frac{n}{B})^{1/2-\epsilon})$ for the exact computation (or even approximation within any constant factor) of the *MST* on graphs of constant diameter $O(1/\epsilon)$, significantly improving the previously best-known lower bound of $\Omega(\frac{n^{1/3}}{B})$. Table 1.1 summarizes the previously known lower bounds on the time complexity of the *MST* problem restricted to graphs of diameter at most Λ, parameterized by Λ, along with our improved lower bounds on this problem.

On the positive side, we devise an H-approximation protocol for the *MST* problem with running time $O(\Lambda(G) + \frac{\omega_{max}}{H-1} \cdot \log^* n)$, where ω_{max} is the ratio between the maximal and the minimal weight of an edge in the input graph (G, ω). It follows that the approximate *MST* problem becomes easy when ω_{max} is small (our lower bounds on the H-approximate *MST* problem apply for $\omega_{max} = \Omega(\sqrt{n} \cdot H^{3/2})$). Finally, our techniques enable us to derive some similar upper and lower bounds on the time-

TABLE 1.1

The summary of previously known and new lower bounds on the MST problem restricted to graphs of diameter at most Λ.

Diameter Λ	Previous lower bound on the exact computation	New lower bound on the exact computation	Lower bound on the time-approximation trade-off
n^{δ}, $0 < \delta < 1/2$	$\Omega(\frac{\sqrt{n}}{B})$ [32]	$\Omega(\sqrt{\frac{n}{B}})$	$T^2 \cdot H = \Omega(\frac{n}{B})$
$\Theta(\log n)$	$\Omega(\frac{\sqrt{n}}{B \cdot \log n})$ [32]	$\Omega(\sqrt{\frac{n}{B \cdot \log n}})$	$T^2 \cdot H = \Omega(\frac{n}{B \cdot \log n})$
Constant (at least 3)	$\Omega(\frac{n^{1/3}}{B})$ [28]	$\Omega((\frac{n}{B})^{\frac{1}{2} - \frac{1}{2\Lambda - 2}})$	$T^{2 + \frac{2}{\Lambda - 2}} \cdot H = \Omega(\frac{n}{B \cdot \Lambda})$
4	$\Omega(\frac{n^{1/3}}{B})$ [28]	$\Omega((\frac{n}{B})^{1/3})$	$T^3 \cdot H = \Omega(\frac{n}{B})$
3	$\Omega(\frac{n^{1/4}}{\sqrt{B}})$ [28]	$\Omega((\frac{n}{B})^{1/4})$	$T^4 \cdot H = \Omega(\frac{n}{B})$

approximation trade-off of the *shortest-path tree* problem, but they are omitted for the sake of brevity.

Structure of the paper. Our main result (the lower bound of $T^2 \cdot H = \Omega(\frac{n}{B})$ on the time-approximation trade-off for the general variant of the *MST* problem) is proved in section 3. In section 4 we present the extensions of this result to the *MST* problem restricted to graphs of diameter smaller than n^{δ}. In section 5 we generalize our lower bounds even further and show that they apply for $\omega_{max} = \Omega(\sqrt{n} \cdot H^{3/2})$. In section 6 we describe our approximation protocol for the *MST* problem.

Related work. Our line of research was continued in a recent work of Kuhn, Moscibroda, and Wattenhofer [24], who proved remarkable lower bounds on time-approximation trade-offs for several problems, including the *maximum dominating set* and *maximum matching*. See also the recent survey [13] and references therein.

2. Preliminaries. For an undirected graph $G = (V, E)$, a *spanning tree* is an acyclic connected subgraph $\tau = (V, E')$, $E' \subseteq E$. For a weighted graph $(G = (V, E), \omega)$ with a nonnegative weight function $\omega : E \rightarrow \mathtt{R}^+$, an *MST* is a spanning tree $\tau = (V, E')$ with minimum weight $\omega(\tau) = \sum_{e \in E'} \omega(e)$.

An H-approximate *MST* τ for a graph (G, ω) is a spanning tree of weight that is at most H times greater than the weight of the *MST* of the graph (G, ω). A protocol Π is said to be an *H-approximation* for the *MST* problem if for every input graph (G, ω) it outputs an H-approximate *MST* τ.

In a *distributed* exact (resp., H-approximate) *MST* problem, at the beginning of computation each vertex v knows its own identity number, the identity numbers of all its neighbors, and the weights of the edges adjacent to v. Let E_v denote the set of these edges. A correct protocol Π must guarantee that at the end of computation every vertex v will know which edges of E_v end up belonging to the computed exact (resp., H-approximate) *MST*. Formally, the output sets K_v of different vertices v should satisfy (1) that for every $v \in V$, $K_v \subseteq E_v$; (2) $\bigcup_{v \in V} K_v = T$, where T is the exact (resp., H-approximate) *MST* tree; and (3) for every edge $e = (v, w) \in E$, either ($e \in K_v$ and $e \in K_w$) or ($e \notin K_v$ and $e \notin K_w$).

Our lower bounds apply to *randomized* protocols with bounded worst-case running time. In other words, these protocols necessarily terminate within specified time bounds, but they are allowed to err with some constant probability $0 \leq q < 1/2$. Two possible types of error are allowed. First, a protocol may produce a subgraph of the input graph (G, ω) that is not an H-approximate *MST* of (G, ω). This subgraph may contain cycles or multiple connectivity components. Second, the protocol may return $\perp$, indicating that it failed to compute the correct answer.

For a pair of vertices $u, w \in V$, we denote by $dist_G(u, w)$ the *unweighted* distance between u and w in the graph $G = (V, E)$.

3. A lower bound on the time-approximation trade-off. In this section we show a lower bound on the trade-off between the possible approximation ratio for the *MST* problem and the running time of a distributed protocol that may achieve this approximation ratio.

3.1. Preliminaries, overview, and discussion. We start with describing the family of graphs that will be used in the proof of our lower bounds. For a sufficiently large positive integer n, let Γ, m, and p be positive integer parameters that satisfy $p \leq \log n$ and $(m+1)\Gamma + \frac{(m+1)^{1+1/p}-1}{(m+1)^{1/p}-1} = n$. Note that $n = \Theta(\Gamma \cdot m)$. Let $d = (m+1)^{1/p}$ (assume that d is integer; nonintegrality issues affect only lower-order terms of our results, and are, therefore, ignored).

Consider a family $\mathcal{G}$ of graphs that contains one unweighted n-vertex graph $G_n = (V_n, E_n)$ for infinitely many positive integers n. The vertex set V_n is comprised of Γ vertex-disjoint paths $P_1, P_2, \ldots, P_\Gamma$ with $m+1$ vertices each, and a d-regular tree τ of depth p with its own vertex set $V(\tau)$ (that is disjoint from $\bigcup_{i=1}^{\Gamma} V(P_i)$). Observe that $|V(\tau)| = 1 + d + \cdots + d^p = \frac{d^{p+1}-1}{d-1} = \frac{(m+1)^{1+1/p}-1}{(m+1)^{1/p}-1}$. Let rt denote the root of τ, i.e., the only vertex that had degree d in τ; all the other vertices have either d children and a parent, or they have only a parent. The latter vertices are called *leaves* of τ. Let $s = z_0, z_1, \ldots, z_m = r$ denote the leaves of τ. The edge set of the graph G_n consists of the edge set $E(\tau)$ of the tree τ, the edge set $\bigcup_{j=1}^{\Gamma} E(P_j)$ of the Γ paths $P_1, P_2, \ldots, P_\Gamma$, and $m+1$ stars S_i, $i \in \{0, 1, \ldots, m\}$. Let $P_j = (v_j^{(0)}, v_j^{(1)}, \ldots, v_j^{(m)})$, $j \in \{1, 2, \ldots, \Gamma\}$, denote the vertices of the path P_j, $j \in \{1, 2, \ldots, \Gamma\}$. Then the edge set of the star S_i, for $i \in \{0, 1, \ldots, m\}$, is the set $\{(z_i, v_j^{(i)}) \mid j \in \{1, 2, \ldots, \Gamma\}\}$. (See Figure 3.1.)

A family $\mathcal{G}^\omega$ of *weighted* graphs contains 2^Γ n-vertex graphs for each n such that $G_n \in \mathcal{G}$. Each of these 2^Γ graphs has the vertex set V_n and the edge set E_n, but the weights of edges are different for at least one edge in any two distinct graphs of $\mathcal{G}^\omega$. The edges of the paths $P_1, P_2, \ldots, P_\Gamma$, as well as the edges of the tree τ, are all of weight zero in all the 2^Γ graphs. The edges of the stars S_i for $i \in \{1, 2, \ldots, m-1\}$ are all of weight infinity in all the 2^Γ graphs. All the edges of the star S_m have unit weight in all the 2^Γ graphs. Each of the Γ edges of the star S_0 may have weight zero or infinity. This induces 2^Γ different n-vertex graphs in $\mathcal{G}^\omega$.

The family $\mathcal{G}$ of unweighted graphs generalizes the family (that we refer to as $\mathcal{G}''$) of unweighted graphs that was used in [28] to prove a lower bound of $\Omega(n^{1/3}/B)$ on the number of rounds that are required for a distributed protocol to compute exact *MST* on graphs of diameter 4. However, the choice of weights above is inherently different from the one used in [28], and this difference will be discussed below.

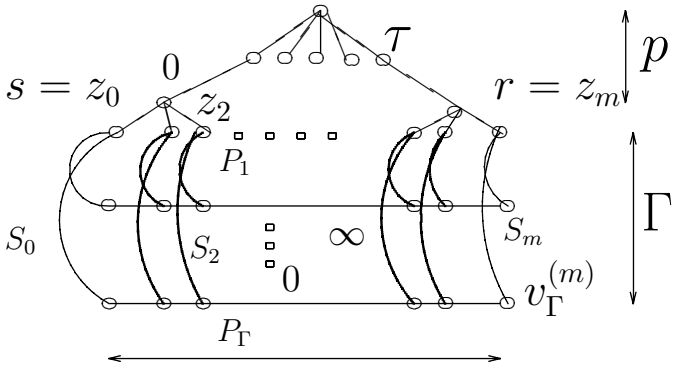


FIG. 3.1. *The family $\mathcal{G}$. The edges of the tree τ and of the paths $P_1, \ldots, P_\Gamma$ have weight zero. The edges of the "internal" stars $S_1, \ldots, S_{m-1}$ have infinite weights. The edges of the rightmost star S_m have unit weights, and the edges of the leftmost star S_0 have weights zero or one, depending on the particular graph.*

Consider the *MST* problem restricted to the family $\mathcal{G}^\omega$ of graphs. We will show that for any $1 \leq H = o(n/B)$, *any* distributed protocol that, given a graph $G \in \mathcal{G}^\omega$ constructs an H-approximate *MST* for G, requires $\Omega((\frac{n}{p \cdot B \cdot H})^{\frac{1}{2} - \frac{1}{2(2p+1)}})$ rounds.

This proof is done by a reduction from a problem of distributed delivery of information throughout the graphs of family $\mathcal{G}$. The problem, referred by us as the CORRUPTEDMAIL problem, generalizes the mailing problem of [32, 28] in two senses. First, the family $\mathcal{G}$ of graphs is somewhat more general than the corresponding families $\mathcal{G}'$ and $\mathcal{G}''$ in [32, 28], and this enables us to get lower bounds that are parameterized on the diameter. Second, the mailing problem of [32, 28] requires exact delivery of all the input bits, whereas our CORRUPTEDMAIL problem allows the protocol to make a restricted number of *one-sided errors* (i.e., some zero input bits might be delivered as ones, but not vice versa). This modification is geared toward capturing the situation when the (oracle) protocol for the *MST* problem (its existence is assumed by the reduction) does not compute the exact *MST*, as it is assumed in [32, 28], but rather provides the reduction to some (possibly very loose) approximation of it.

For a bit string $\chi \in \{0,1\}^\Gamma$ and an index $j \in \{1, 2, \ldots, \Gamma\}$, let χ_j denote the jth bit of χ. The *Hamming weight* of χ, denoted $||\chi||$, is the number of indices $j \in \{1, 2, \ldots, \Gamma\}$ such that $\chi_j = 1$. For two bit strings $\chi, \chi' \in \{0,1\}^\Gamma$, the string χ' is said to *dominate* χ if for each $j \in \{1, 2, \ldots, \Gamma\}$, $\chi_j = 1$ implies $\chi'_j = 1$. Consider some mapping $\phi : \{0,1\}^\Gamma \to \{0,1\}^\Gamma$, and suppose $\phi(\chi) = \chi'$. The mapping ϕ is said to make an error of the *first* (resp., *second*) *type* in the jth position if $\chi_j = 0$ and $\chi'_j = 1$ (resp., $\chi_j = 1$ and $\chi'_j = 0$).

Let α and β, $0 < \alpha < \beta < 1$, be two additional parameters of the construction that will be fixed later. The CORRUPTEDMAIL(α, β) problem is defined on unweighted graphs $G_n \in \mathcal{G}$. Recall that for each graph $G \in \mathcal{G}$, there are two designated vertices s and r, $s, r \in V_n$. The input to this problem is a bit string $\chi \in \{0,1\}^\Gamma$ of length Γ with Hamming weight $||\chi|| = \alpha\Gamma$. The input is provided to the vertex s only. The output, returned by the vertex r, is a string $\chi' \in \{0,1\}^\Gamma$ of Hamming weight at most $\beta\Gamma$, and it is required that the output string χ' dominate the input string χ. As the parameters α and β are fixed throughout this and the next sections, we will refer to the CORRUPTEDMAIL(α, β) problem as the CORRUPTEDMAIL problem.

Observe, however, that the restriction that χ' should dominate χ guarantees that the errors that are done throughout the delivery of the bit string χ are *one-sided* (i.e., all the errors are of the first type). This is in contrast to the usual setting of error-correcting codes, where a two-sided error is allowed. One of the crucial properties required for a reduction from the CorruptedMail problem to the approximate *MST* problem to work is that approximating the *MST* with sufficiently small factor causes only one-sided errors in the delivery of the bit string χ. In our reduction, the latter is ensured by an appropriate choice of the weights of the edges in $\mathcal{G}^{\omega}$. This is the essential difference between our choice of weights of edges for $\mathcal{G}^{\omega}$ and the choices of weights of edges in analogous families of graphs in [32] and [28].

We remark that allowing a symmetric two-sided error in the CorruptedMail problem, and using error-correcting codes (possibly with list-decoding) directly would also lead to a proof of hardness of distributed approximation of the *MST* problem within some small constant factor (smaller than 2, to the best of our knowledge). The bottleneck in this case is the fact that corrupting a bit string χ that has Hamming weight $\alpha\Gamma$ in at least $\alpha\Gamma$ arbitrary positions in an arbitrary way may make the string almost indistinguishable from a string that is drawn out of the uniform distribution over all the strings of Hamming weight at most $2\alpha\Gamma$, even from an information-theoretic point of view. However, this is not the case when only a one-sided error is allowed. Then, as we will show, one can allow $(\beta-\alpha)\Gamma$ corrupted positions for $1 > \beta \gg \alpha > 0$, and still the corrupted string χ' will carry on a significant amount of information (roughly, $\Gamma \cdot \alpha \cdot \log(1/\beta)$ bits for sufficiently small positive $\alpha > 0$; note that the amount of information carried by χ is $\Gamma \cdot \mathit{Entropy}(\alpha) = \Gamma \cdot (\alpha \log(1/\alpha) + (1-\alpha)\log(1/(1-\alpha)))$). This way, we are able to prove hardness of distributed approximation of the *MST* problem within a factor of, roughly, $1/\alpha$, for an arbitrarily small $\alpha > 0$.

As we mentioned, to ensure a one-sided error of the reduction, an appropriate setting of the weights of edges of graphs in $\mathcal{G}^{\omega}$ is required. The setting that we described has, however, the drawback of a very high ratio, denoted ω_{max}, between the largest weight of an edge and the smallest one. Specifically, this ratio is $\Theta(n^2)$. If one wants to prove a similar lower bound for instances of the *MST* problem with smaller ω_{max} ratios, it is no longer possible to guarantee a one-sided error in the delivery of the bit string χ in the reduction. Nevertheless, as we will show in section 5, controlling the ratio between n^2 and ω_{max} also controls the number of errors of the second type. In this case we get into a situation when the delivery may suffer an *asymmetric two-sided error*, with a huge possible number of errors of the first type (specifically, $(\beta-\alpha)\Gamma$ for some $1 > \beta \gg \alpha > 0$) and a reasonably small number of errors of the second type (specifically, some constant fraction of $\alpha\Gamma$). We will show that this way one can achieve the same lower bound on time-approximation trade-off as in the case that $\omega_{max} = \Theta(n^2)$ (that is, $T^2 \cdot H = \Omega(n/B)$), but with $\omega_{max} = \sqrt{n} \cdot H^{3/2}$, which is significantly smaller than n^2 for $H(n) = o(n)$. We remark that one cannot expect to achieve such a trade-off with $\omega_{max} = o(\frac{\sqrt{nH/B}}{\log^* n})$. This is because, as we will show in section 6, one can approximate the *MST* on an n-vertex graph G within a ratio of $O(\omega_{max}/T')$ in $O(T' \log^* n + \Lambda(G))$ rounds. For $\omega_{max} = o(\frac{\sqrt{nH/B}}{\log^* n})$, the approximation ratio is $H = O(\omega_{max}/T') = o(\frac{\sqrt{nH/B}}{T' \log^* n})$. The number of rounds is $T = T' \log^* n = o(\sqrt{nH/B})$, i.e., $T^2 H = o(\sqrt{n/B})$, contradicting our lower bound.

3.2. A lower bound for the CorruptedMail problem. In this section we prove a lower bound on the time complexity of the CorruptedMail problem.

For $0 < \alpha < \beta < 1$, let

$$l(\alpha,\beta) = (\beta-\alpha)\log(\beta-\alpha) - (1-\alpha)\log(1-\alpha) - \beta\log\beta\ . \tag{1}$$

Lemma 3.1. *For any α, β, $0 < \alpha < \beta < 1$,*

$$l(\alpha,\beta) \geq \alpha \cdot \log(1/\beta). \tag{2}$$

Proof. Let $f(\alpha,\beta) = l(\alpha,\beta) + \alpha \cdot \log\beta$. It is easy to verify that for a fixed β, $0 < \beta < 1$, it holds that $f(0,\beta) = 0$. Also,

$$\frac{\partial f}{\partial\alpha}(\alpha,\beta) \ = \ \frac{\partial l}{\partial\alpha}(\alpha,\beta) + \log\beta \ = \ \log\left(\frac{\beta-\beta\alpha}{\beta-\alpha}\right) \ > \ 0$$

for any α, β in the range $0 < \alpha < \beta < 1$, and $\frac{\partial f}{\partial\alpha}(0,\beta) = 0$ for any β, $0 < \beta < 1$. Hence,

$$f(\alpha,\beta) = f(0,\beta) + \int_0^\alpha \frac{\partial f}{\partial\alpha}(x,\beta)dx \ \geq \ 0$$

for any α, β in the same range. □

In the following lemma the Ω-notation hides a universal constant.

Lemma 3.2. *For any deterministic protocol Π for the* CorruptedMail(α,β) *problem, its set of all possible outputs $\{\Pi(\chi) \mid \chi \in \{0,1\}^\Gamma, ||\chi|| = \alpha\Gamma\}$ contains at least $\Omega(2^{l(\alpha,\beta)\Gamma})$ elements.*

Proof. Let $\mathcal{A} = \{\chi \in \{0,1\}^\Gamma \mid ||\chi|| = \alpha\cdot\Gamma\}$ be the set of all possible bit strings that may serve as input for the vertex s. Let $\Upsilon = \{\chi' \in \{0,1\}^\Gamma \mid ||\chi'|| \leq \beta\cdot\Gamma\}$ be the set of all possible bit strings that may be returned by the vertex r. Consider the bipartite graph $(\mathcal{A}, \Upsilon, E(\mathcal{A},\Upsilon))$ with $\mathcal{A}$ serving as the set of the left-hand vertices, Υ serving as the set of the right-hand vertices, and $E(\mathcal{A},\Upsilon) = \{(\chi,\chi') \mid \chi \in \mathcal{A}, \chi' \in \Upsilon \text{ s.t. } \chi' \text{ dominates } \chi\}$. Observe that $|\mathcal{A}| = \binom{\Gamma}{\alpha\Gamma} = \frac{\Gamma!}{(\alpha\Gamma)!((1-\alpha)\Gamma)!}$. Consider some bit string $\chi' \in \Upsilon$. The number of the bit strings $\chi \in \mathcal{A}$ that are dominated by χ' is at most $D = \binom{\beta\Gamma}{\alpha\Gamma} = \frac{(\beta\Gamma)!}{((\beta-\alpha)\Gamma)!(\alpha\Gamma)!}$.

A subset $\Upsilon' \subseteq \Upsilon$ is said to *dominate* the set $\mathcal{A}$ if for each bit string $\chi \in \mathcal{A}$ there exists a bit string $\chi' \in \Upsilon'$ that dominates χ. As each bit string $\chi' \in \Upsilon$ may dominate at most D bit strings of $\mathcal{A}$, it follows that any subset $\Upsilon' \subseteq \Upsilon$ that dominates $\mathcal{A}$ has cardinality at least

$$|\Upsilon'| \geq \frac{|\mathcal{A}|}{D} = \frac{\Gamma!((\beta-\alpha)\Gamma)!}{((1-\alpha)\Gamma)!(\beta\Gamma)!}\ .$$

Using the Stirling formula to approximate the factorials, we get

$$|\Upsilon'| \ \geq \ \left(\frac{(\beta-\alpha)^{\beta-\alpha}}{(1-\alpha)^{1-\alpha}\beta^\beta}\right)^\Gamma \cdot \sqrt{\frac{\beta-\alpha}{(1-\alpha)\beta}} \cdot (1-o(1))\ .$$

For β and α such that $\beta - \alpha > 0$ is at least some constant, it follows that $|\Upsilon'| = \Omega\big(\big(\frac{(\beta-\alpha)^{\beta-\alpha}}{(1-\alpha)^{1-\alpha}\beta^\beta}\big)^\Gamma\big)$.

Let $h(\alpha,\beta) = \frac{(\beta-\alpha)^{\beta-\alpha}}{(1-\alpha)^{1-\alpha}\beta^{\beta}}$, and note that $l(\alpha,\beta) = \log h(\alpha,\beta)$. It follows that $|\Upsilon'| = \Omega(2^{l(\alpha,\beta)\Gamma})$ for any subset $\Upsilon' \subseteq \Upsilon$ of bit strings that dominate $\mathcal{A}$. □

We next show that for any protocol Π with worst-case running time at most t for t in a certain range, its set of possible outputs (over all possible inputs) is at most exponential in t. To argue this we use the determinism of the protocol and the fact that the vertex r that returns the output could not get "too much" information about the input. It will follow that the running time t of a protocol Π cannot be too small, unless the protocol Π is incorrect. To prove an upper bound on the size of the set of possible outputs of any correct protocol, we show that the set of all possible *configurations* of the vertex r is relatively small as a function of t.

For some fixed sufficiently large n, consider again a graph $G = (V,E) = G_n \in \mathcal{G}$ that was described in section 3.1. Intuitively, we next argue that information can be delivered through G_n from s to r in a quite slow rate. A similar statement was proved in [32, 28] regarding somewhat different families of graphs $\mathcal{G}'$ and $\mathcal{G}''$.

Our proof has a similar structure to that of [32], but as the structure of the family $\mathcal{G}$ is somewhat more complicated than that of $\mathcal{G}'$ or $\mathcal{G}''$, the proof requires a more delicate argument. Basically, all three proofs (due to [32, 28] and this paper) construct a sequence of low-capacity cuts and argue that each bit has to cross all these cuts. As the cuts have low capacities, no cut can be crossed by "many" bits simultaneously, implying a lower bound on the number of rounds of distributed computation. However, while the choice of the cuts and the proof that they have low capacity are rather straightforward in [32, 28], it is somewhat more involved in our case.

We first need a few definitions. For a rooted tree (τ', rt'), let the *ancestor-descendent* (resp., *parent-child*) relation, denoted $AD(\tau', rt')$ (resp., $PC(\tau', rt')$), be the set of pairs of distinct vertices $(u,w) \in V(\tau') \times V(\tau')$ such that the vertex u is an ancestor (resp., parent) of the vertex w in the tree (τ', rt'). For a vertex $u \in V(\tau')$, let $par_{(\tau',rt')}(u)$ denote the *parent* of u in the rooted tree (τ', rt').

Recall that the graph $G = G_n$ contains as a subgraph the d-regular rooted tree (τ, rt) of height p, with $m+1$ leaves $s = z_0, z_1, \ldots, z_m = r$. For $i \in \{0,1,2,\ldots,m\}$, let $\tau(i)$ denote the connected subtree of τ with minimal number of vertices, such that its set of leaves, $Leaves(\tau(i))$, is equal to $\{z_i, z_{i+1}, \ldots, z_m\}$. Let the *root* of $\tau(i)$, denoted $rt(\tau(i))$, be the closest vertex of $\tau(i)$ to the root rt of the tree τ.

Observe that for a pair of vertices $x, y \in V(\tau(i))$ such that x is an ancestor of y in the rooted tree (τ, rt), x is an ancestor of y in the rooted tree $(\tau(i), rt(\tau(i)))$ as well; i.e., $(x,y) \in AD(\tau, rt)$ and $x, y \in V(\tau(i))$ imply that $(x,y) \in AD(\tau(i), rt(\tau(i)))$. Also, $AD(\tau(i), rt(\tau(i))) \subseteq AD(\tau, rt)$ for each $i \in \{0,1,\ldots,m\}$. Note that both statements are true for the parent-child relation PC as well.

Let Cut_i denote the subset of edges of $E(\tau)$ that are in the cut between $V(\tau(i))$ and the rest of $V(\tau)$ for $i \in \{1,2,\ldots,m\}$. We next argue that the size of this cut is never greater than $p \cdot d$.

LEMMA 3.3. *For $i \in \{1,2,\ldots,m\}$, $|Cut_i| \leq p \cdot d$.*

Proof. Fix an index $i \in \{1,2,\ldots,m\}$. Let $L_j = \{(u,w) \in Cut_i \cap AD(\tau, rt) \mid dist_\tau(rt,u) = j\}$, for $j \in \{0,1,\ldots,p-1\}$, be the *jth layer* of the edge set Cut_i. Note that $Cut_i = \bigcup_{j=0}^{p-1} L_j$. It remains to argue that $|L_j| \leq d$ for $j \in \{0,1,\ldots,p-1\}$.

To this end, we will show that for any pair of edges $e_1 = (u_1, w_1), e_2 = (u_2, w_2) \in L_j$ (assume, without loss of generality, that $u_1 = par_{(\tau,rt)}(w_1)$ and $u_2 = par_{(\tau,rt)}(w_2)$), u_1 is equal to u_2. It will follow that all the edges of L_j share a common endpoint, and, furthermore, this endpoint is a parent of all the other endpoints of these edges. As every vertex has at most d children in the tree τ, this would imply $|L_j| \leq d$.

Consider a pair of edges e_1, e_2 as above. First, suppose $u_1 \notin V(\tau(i))$, $w_1 \in V(\tau(i))$. As the parent-child relation in $\tau(i)$ is a subset of the parent-child relation in τ, it follows that $w_1 = rt(\tau(i))$. In this case it is easy to see that (u_1, w_1) is the only edge in L_j, and consequently, $e_1 = e_2$, and $|L_j| = 1$.

It remains to consider the case when $u_1, u_2 \in V(\tau(i))$ and $w_1, w_2 \in V(\tau) \setminus V(\tau(i))$. However, in this case $u_1 \neq u_2$ and $u_1, u_2 \in V(\tau(i))$ imply that either all the descendents of u_1 or all the descendents of u_2 belong to $V(\tau(i))$, and both of them contradict $w_1, w_2 \in V(\tau) \setminus V(\tau(i))$. □

We define the *tail* sets, $T_0, T_1, \ldots, T_m$ as follows. The tail set T_0 contains the entire vertex set V of G except the vertex s, i.e., $T_0 = V \setminus \{s\}$. For $i \in \{1, 2, \ldots, m\}$, $T_i = \{v_j^{(i')} \mid j \in \{1, 2, \ldots, \Gamma\}, i' \in \{i, i+1, \ldots, m\}\} \cup V(\tau(i))$. It will be more convenient to view the vertex sets T_i as sequences of vertices. The order of the vertices in these sequences can be arbitrary, and for simplicity we will assume that the vertices are ordered as a monotone increasing sequence of their (distinct) identity numbers.

Consider some protocol Π for the CorruptedMail problem, and consider an execution φ_χ of this protocol on some input bit string χ. Let the *state* of the vertex v at the beginning of round t during the execution φ_χ of the protocol Π, denoted $\sigma(v, t, \chi)$, be the $deg(v)$-tuple of sequences of messages that the vertex v received on its incoming links. For the vertex s, the state of s also includes the input string χ. Other vertices receive no input in the CorruptedMail problem.

For some subset $U = (u_1, u_2, \ldots, u_\ell) \subseteq V$ of vertices, let the *configuration* of U in the execution φ_χ at the beginning of round t, denoted $C(U, t, \chi)$, be the sequence of states $(\sigma(u_1, t, \chi), \sigma(u_2, t, \chi), \ldots, \sigma(u_\ell, t, \chi))$. Let $\mathcal{C}(U, t)$ denote the collection of all possible configurations of the subset U at the beginning of round t over all possible executions φ_χ (i.e., for all possible legal input bit strings χ; given a bit string, the execution φ_χ is fixed). Let $\rho(U, t)$ denote the cardinality of $\mathcal{C}(U, t)$. We remark that our proof does not attempt to analyze the actual sets of configurations, but rather only the cardinalities of those sets.

In what follows let us assume that the rounds are indexed starting from 0 and not from 1. Obviously, this may affect the lower bounds by at most an additive term of 1. At the beginning of round $t = 0$, i.e., at the beginning of the execution of the protocol, all the vertices except s are in some fixed initial state, that is, independent of the input bit string χ. Hence, $\rho(T_0, 0) = \rho(V \setminus \{s\}, 0) = 1$. We next prove an upper bound on the number of configurations of the tail set T_t after a relatively small number of rounds $t \leq m - 1$.

Lemma 3.4. *For $t \in \{0, 1, \ldots, m-1\}$, $\rho(T_t, t) \leq (2^{B+1} - 1)^{t \cdot p \cdot d}$.*

Remark. To provide some intuition for the proof of Lemma 3.4, we illustrate its central idea on a simple example. Consider a path $P = (v_0, v_1, \ldots, v_n)$ with the vertex v_0 serving as a sender, and v_n serving as a receiver, that is, $s = v_0$, $r = v_n$. Also, let $T_i = \{v_{i+1}, v_{i+2}, \ldots, v_n\}$ be the tail sets, for $i \in \{0, 1, \ldots, n-1\}$.

Suppose we are given a configuration C of the vertices of T_i at the beginning of round i. This configuration uniquely determines the messages sent on round i by the vertices of T_i, and, in particular, by the vertex v_{i+1}. Hence, the unique configuration of T_0 at the beginning of round 0 (note that $T_0 = V \setminus \{s\}$, and s is the only vertex that receives input) determines the unique configuration of T_1 at the beginning of round 1. This is because the only link that connects T_1 to $V \setminus T_1$ is the edge (v_1, v_2), but the message sent by the vertex v_1 over this link on round 0 is determined by the

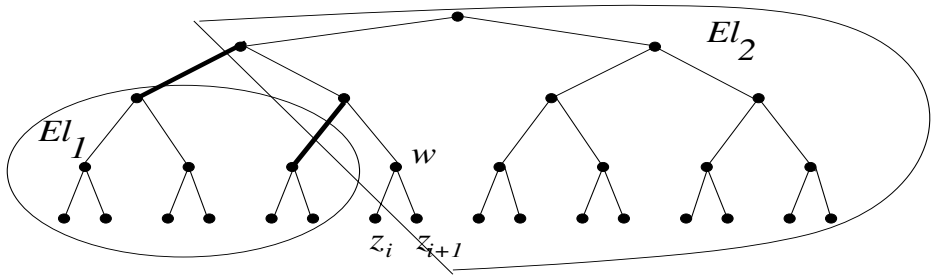


FIG. 3.2. *The sets $V(\tau) \setminus V(\tau(i))$ and $V(\tau(i+1))$ are depicted by ellipses El_1 and El_2, respectively. Edges of $E_{i,i+1}$ are depicted by thick solid lines. Note that, somewhat surprisingly, the edge (z_i, w) does not belong to $E_{i,i+1}$ because $z_i \in V(\tau(i))$.*

configuration of T_0 at the beginning of round 0, and this configuration is independent of the bit string χ that the sender s accepts as input.

Applying this consideration inductively we conclude that the configuration of the tail set $T_{n-1} = \{r\} = \{v_n\}$ at the beginning of round $n-1$ is uniquely determined by the initial configuration of the vertices of the set $T_0 = V \setminus \{s\}$ and is independent of the input string χ.

For a more elaborate example consider the graph $H = P \cup S$, where P is the n-vertex path that we have just discussed, and $S = \{(v_0, v_i) \mid i \in \{2, 3, \ldots, n\}\}$ is the star rooted at v_0. The tail sets are left unchanged.

In this graph the unique configuration of the vertices of T_0 at the beginning of round 0 branches into roughly $2^{B(n-2)}$ possible configurations of the vertices of T_1 at the beginning of round 1, because the vertices of T_1 could receive up to $B(n-2)$ different bits from the vertex v_0 on this round, and each distinct combination of these bits may determine a new configuration of the vertices of T_1. Repeating this argument we derive that there are roughly $2^{B(n-2)} \cdot 2^{B(n-3)}$ possible configurations of T_2 at the beginning of round 2, and, ultimately, at most $2^{O(B \cdot n^2)}$ possible configurations of the tail set $T_{n-1} = \{r\}$ at the beginning of round $n-1$. In other words, intuitively, at most $O(Bn^2)$ bits could have been delivered through this network in $n-1$ rounds.

Proof. The proof is by induction on t. The induction base was argued above. We next prove the induction step. Observe that $T_0 \supseteq T_1 \supseteq \cdots \supseteq T_m$. Suppose we are given a configuration $C \in \mathcal{C}(T_i, i)$. Note that C uniquely determines the messages that are sent at round i by vertices of T_i. Therefore, the only messages that are not yet determined are those that are sent by the vertices of $V \setminus T_i$ to the vertices of T_{i+1}. Denote this set of edges by $E_{i,i+1}$. Observe that the edges of the paths $P_1, P_2, \ldots, P_\Gamma$ do not belong to $\bigcup_{i=0}^{m-1} E_{i,i+1}$. It follows that $E_{i,i+1} \subseteq E(\tau)$ for $i \in \{0, 1, \ldots, m-1\}$. For $i = 0$, clearly, $E_{0,1} = \{(z_0, par_{(\tau,rt)}(z_0))\}$. More generally, $E_{i,i+1}$ is a subset of edges of $E(\tau)$ that are incident both to $V(\tau(i+1))$ and to $V \setminus V(\tau(i))$, $i \in \{0, 1, \ldots, m-1\}$. See Figure 3.2 for an illustration.

Note that $E_{i,i+1} \subseteq Cut_{i+1}$ for $i \in \{0, 1, \ldots, m-1\}$. Hence, by Lemma 3.3, $|E_{i,i+1}| \leq p \cdot d$ for $i \in \{0, 1, \ldots, m-1\}$. In other words for $i \in \{0, 1, \ldots, m-1\}$, there are at most $p \cdot d$ edges that are incident to the tail set T_{i+1}, such that given a configuration C in $\mathcal{C}(T_i, i)$ of the vertices of T_i at the beginning of round i, the messages that are sent over these edges at round i are not determined by this configuration.

Recall that at most B bits can be delivered through an edge in each round. Therefore, the number of possible messages that may be sent through an edge in a

given direction in one round is at most $\sum_{\ell=0}^{B} 2^\ell = 2^{B+1} - 1$. (In this summation, the case $\ell = 0$ reflects the possibility of transmitting nothing through a certain edge on a certain round.)

Observe that there is only one relevant direction of sending messages in our case, that is, towards the vertices of the tail set T_{i+1}. Hence, the number of possible messages that may be sent through at most $p \cdot d$ edges in one round is at most $(2^{B+1}-1)^{p \cdot d}$. It follows that for $i \in \{0, 1, \ldots, m-1\}$, $\rho(T_{i+1}, i+1) \leq (2^{B+1}-1)^{p \cdot d} \cdot \rho(T_i, i)$. Along with $\rho(T_0, 0) = 1$, this implies the statement of the lemma. □

Note that the proof argument that counts the number of configurations holds when each vertex knows its unique identity number at the beginning of computation.

In the following lemma and its proof, the asymptotic Ω- and O-notations hide universal constants.

LEMMA 3.5. *The deterministic complexity of the* CORRUPTEDMAIL *problem is* $\Omega(\min\{m, \frac{\Gamma}{p \cdot B \cdot m^{1/p}} \cdot \alpha \cdot \log(\frac{1}{\beta})\})$.

Proof. Consider a protocol Π for the CORRUPTEDMAIL problem, and let t denote its worst-case running time on inputs of fixed size Γ. By Lemma 3.4, it follows that if $t \leq m-1$, then $\rho(T_t, t) \leq (2^{B+1}-1)^{t \cdot p \cdot d}$. On the other hand, observe that the number of possible configurations of the vertex r upon the termination of the protocol Π is at least the cardinality of some subset $\Upsilon' \subseteq \Upsilon$ of a subset of bit strings that dominates $\mathcal{A}$. By Lemma 3.2, $|\Upsilon'| = \Omega(2^{l(\alpha,\beta)\Gamma})$. It follows that the (worst-case) running time t of the protocol Π for the CORRUPTEDMAIL problem is either at least $t \geq m$ rounds, or it satisfies the following inequality

$$\Omega(2^{l(\alpha,\beta)\Gamma}) = |\Upsilon'| \leq \rho(\{r\}, t) \leq \rho(T_t, t) \leq (2^{B+1}-1)^{t \cdot p \cdot d} \ . \tag{3}$$

(The inequality $\rho(\{r\}, t) \leq \rho(T_t, t)$ follows from the fact that $r \in T_t$ for $t \in \{0, 1, \ldots, m\}$.) Hence, $\Omega(l(\alpha,\beta)\Gamma) - O(1) \leq t \cdot p \cdot d \cdot B$; i.e., $t = \Omega(l(\alpha,\beta) \cdot \Gamma/(p \cdot d \cdot B))$. In other words, in both cases $t = \Omega(\min\{m, (l(\alpha,\beta) \cdot \Gamma)/(p \cdot d \cdot B)\})$. Recall that $n = O(\Gamma m)$ and $d = (m+1)^{1/p}$. Hence,

$$t = \Omega\left(\min\left\{m, l(\alpha,\beta)\frac{n}{p \cdot m^{1+1/p} \cdot B}\right\}\right). \tag{4}$$

By Lemma 3.1, for $\alpha > 0$ it holds that $l(\alpha,\beta) \geq \alpha \cdot \log(1/\beta)$. Set β to be a constant between 0 and 1, and the lemma follows. □

Finally, set $m = \left(\frac{n \cdot \alpha}{p \cdot B}\right)^{\frac{1}{2} - \frac{1}{2(2p+1)}}$. It follows that

$$t = \Omega\left(\left(\frac{n \cdot \alpha}{p \cdot B}\right)^{\frac{1}{2} - \frac{1}{2(2p+1)}}\right). \tag{5}$$

It follows that any deterministic protocol Π that solves the CORRUPTEDMAIL problem on *every* input bit string χ requires $\Omega\left(\left(\frac{n \cdot \alpha}{p \cdot B}\right)^{\frac{1}{2} - \frac{1}{2(2p+1)}}\right)$ rounds in the worst case.

Henceforth, the term "randomized protocol" will be used as a shortcut for "randomized protocol that succeeds with at least some constant probability on every input."

LEMMA 3.6. *Any randomized protocol* Π *for the* CORRUPTEDMAIL *problem requires* $\Omega\left(\left(\frac{n \cdot \alpha}{p \cdot B}\right)^{\frac{1}{2} - \frac{1}{2(2p+1)}}\right)$ *rounds.*

Proof. Consider a deterministic protocol Π' that accepts as input a string drawn at random from the uniform distribution over the bit strings χ of Hamming weight

$\alpha\Gamma$. For such a protocol to succeed with at least a constant probability q in $t \leq m-1$ rounds, the number of configurations of the vertex r at the end of the protocol has to be at least the size of a subset $\Upsilon'' \subseteq \Upsilon$ that dominates at least a fraction q of the set $\mathcal{A}$. However, by the same considerations as above, such a subset Υ'' must have cardinality at least $q|\mathcal{A}|/D$. In other words, for protocol Π', $\rho(T_t, t) \geq \rho(\{r\}, t) \geq q|\mathcal{A}|/D = \Omega(2^{l(\alpha,\beta)\Gamma})$. In other words, up to lower-order terms, the same lower bound (5) applies also to a protocol Π' as above. By Yao's minimax theorem [37], any randomized protocol that succeeds with probability at least q on *every* input of the CorruptedMail problem requires at least as many rounds as required by a deterministic protocol Π' that succeeds with the same probability on the uniform distribution of inputs. Hence, the lower bound (5) applies to randomized protocols as well. □

3.3. Reduction to the approximate *MST* problem. In this section we describe the reduction from the CorruptedMail(α, β) problem on the family $\mathcal{G}$ of unweighted graphs to the $\frac{\beta}{\alpha}$-approximate *MST* problem on the family $\mathcal{G}^\omega$ of weighted graphs (see the beginning of section 3 for the definition of this family).

The protocol Π_{Corr} for the CorruptedMail(α, β) problem proceeds in the following way. Given an instance (G, χ), $G \in \mathcal{G}$, $\chi \in \mathcal{A}$, of the CorruptedMail problem, the vertex s computes the weights of edges $(s, v_j^{(0)})$, $j \in \{1, 2, \ldots, \Gamma\}$, in the following way. If $\chi_j = 0$, then the weight of the edge $(s, v_j^{(0)})$ is set to zero; otherwise it is set to infinity. All the other weights of edges are set by their endpoints to the values that are determined by the definition of the family $\mathcal{G}^\omega$. (Recall that there is no freedom in setting other weights, and the only difference between two distinct n-vertex graphs from $\mathcal{G}^\omega$ is the setting of the weights of the edges of the star S_0.) This setting of weights is performed locally by every vertex and requires no distributed computation.

Next, the vertices invoke a $\frac{\beta}{\alpha}$-approximation protocol Π for the obtained instance $G(\chi)$ of the *MST* problem. Upon the termination of the protocol, each vertex v knows which edges among the edges that are incident to v belong to the approximate *MST* tree τ_0 for $G(\chi)$ that was constructed by the protocol. The vertex r calculates the output bit string $\chi' \in \{0,1\}^\Gamma$ in the following way. For each index $j \in \{1, 2, \ldots, \Gamma\}$, if the edge $(r, v_j^{(m)})$ belongs to the tree τ_0, the vertex r sets $\chi'_j = 1$. Otherwise, it sets $\chi'_j = 0$. Finally, the vertex r returns the bit string χ'.

Observe that whenever the construction of the approximate *MST* tree τ_0 is completed, the computation of the bit string χ' is performed locally by the vertex r and requires no distributed computation. It follows that the running time of the obtained protocol Π_{Corr} for the CorruptedMail(α, β) problem is precisely equal to the running time of the $\frac{\beta}{\alpha}$-approximation protocol Π for the *MST* problem.

We next argue that the reduction is correct, i.e., that the protocol Π_{Corr} solves correctly the CorruptedMail(α, β) problem, assuming that the protocol Π is a $\frac{\beta}{\alpha}$-approximation protocol for the *MST* problem.

Lemma 3.7. *For each $\chi \in \mathcal{A}$, if τ_0 is a $\frac{\beta}{\alpha}$-approximate MST for the graph $G(\chi)$, then the bit string χ' that is returned by the protocol Π_{Corr} has Hamming weight at most $\beta \cdot \Gamma$, and, furthermore, the bit string χ' dominates the input bit string χ.*

Proof. Consider a bit string $\chi \in \{0,1\}^\Gamma$ of Hamming weight $||\chi|| = \alpha\Gamma$. By construction, precisely $\alpha\Gamma$ edges of the star S_0 have weight ∞ in $G(\chi)$. Therefore, the exact *MST* contains all the edges of the paths $P_1, P_2, \ldots, P_\Gamma$ and of the tree τ, as all of them have weight zero, and it contains $(1-\alpha)\Gamma$ edges of weight zero that belong

to the star S_0. In addition, for each index $j \in \{1, 2, \ldots, \Gamma\}$ such that $\omega(s, v_j^{(0)}) = \infty$, it contains the edge $(r, v_j^{(m)})$. Recall that the latter edge has unit weight. Also, note that $\omega(s, v_j^{(0)}) = \infty$ implies that $\chi_j = 1$. As $||\chi|| = \alpha\Gamma$, it follows that exactly $\alpha\Gamma$ edges of the star S_0 have weight ∞. Hence, exactly $\alpha\Gamma$ edges of the star S_m belong to the *MST*, implying that its weight is $\alpha\Gamma$.

Consider a $\frac{\beta}{\alpha}$-approximate *MST* τ_0. By definition of approximate *MST*, its weight is at most $\beta\Gamma$. Hence, in particular, it contains at most $\beta\Gamma$ edges of the star S_m, implying that $||\chi'|| \leq \beta\Gamma$. Consider an index $j \in \{1, 2, \ldots, \Gamma\}$ such that $\chi_j = 1$. It follows that the weight of the edge $(s, v_j^{(0)})$ in $G(\chi)$ is ∞. As no edge with infinite weight may belong to τ_0 (as its weight is at most $\beta\Gamma$), it follows that neither $(s, v_j^{(0)})$ nor $(z_i, v_j^{(i)})$ for some $i \in \{1, 2, \ldots, m-1\}$ may belong to the tree τ_0. It follows that the edge $(r, v_j^{(m)})$ belongs to τ_0, as otherwise the vertex s would not be connected in τ_0 to the vertices of the path P_j. The latter would imply that τ_0 is not a spanning tree of the graph $G(\chi)$, contradicting the assumption that it is an approximate *MST* for $G(\chi)$. Hence, the edge $(r, v_j^{(m)})$ belongs to the tree τ_0. Hence, the bit χ'_j is set to 1 by the reduction. It follows that the output bit string χ' dominates the input bit string χ and that $||\chi'|| \leq \beta\Gamma$. □

Therefore, if Π is a $\frac{\beta}{\alpha}$-approximation protocol for *MST* on the family $\mathcal{G}^\omega$ of weighted graphs, then Π_{Corr} is a protocol for the CorruptedMail(α, β) problem on the family $\mathcal{G}$ of unweighted graphs, with the same running time. Recall that any (deterministic or randomized) protocol for the CorruptedMail(α, β) problem on the family $\mathcal{G}$ of graphs requires $t = \Omega\big(\big(\frac{n \cdot \alpha}{p \cdot B}\big)^{\frac{1}{2} - \frac{1}{2(2p+1)}}\big)$ rounds. Observe that all the graphs in the family $\mathcal{G}^\omega$ have the same unweighted diameter $2p+2$. The next theorem follows.

Theorem 3.8. *Any randomized H-approximation protocol for the MST problem on graphs of diameter at most Λ for $\Lambda \in \{4, 6, 8, \ldots\}$ requires $T = \Omega\big(\big(\frac{n}{H \cdot \Lambda \cdot B}\big)^{\frac{1}{2} - \frac{1}{2(\Lambda-1)}}\big)$ rounds of distributed computation; i.e., $T^{2 + \frac{2}{\Lambda-2}} \cdot H = \Omega\left(\frac{n}{\Lambda \cdot B}\right)$.*

In particular, for any $\epsilon > 0$, approximation of *MST* on graphs with constant diameter $\Lambda \geq 4$ within a factor of $\big(\frac{n}{B}\big)^{1-\epsilon}$ requires at least $\Omega(\big(\frac{n}{B}\big)^{\epsilon(\frac{1}{2} - \frac{1}{2\Lambda-2})})$ rounds of distributed computation.

We remark that in addition to being the first result on the hardness of distributed approximation, Theorem 3.8 also implies a lower bound of $\Omega((n/B)^{1/2-\epsilon})$ on the time complexity of the exact computation (or even approximation within any constant factor) of the *MST* problem restricted to graphs with constant diameter $O(1/\epsilon)$. Previously, the best-known lower bound on exact computation of *MST* on graphs with constant diameter was $\Omega(n^{1/3}/B)$ due to [28]. However, the lower bound of [28] is stronger in the sense that it requires the protocol to work only on graphs of diameter at most 4. Substituting $\Lambda = 4$ in our result yields an improvement, by a factor of $B^{2/3}$, in the result of [28].

By substituting $\Lambda = \log \frac{n}{H \cdot B}$, we get the following.

Corollary 3.9. *Any randomized H-approximation protocol for the MST problem on graphs of diameter $\Lambda = O(\log n)$ requires $T = \Omega(\sqrt{\frac{n}{H \cdot B \cdot \log n}})$ rounds of distributed computation; i.e., $T^2 \cdot H = \Omega(\frac{n}{B \cdot \log n})$.*

A lower bound of $\Omega(\frac{\sqrt{n}}{B \log n})$ on the running time of a protocol that computes the *MST* exactly on graphs of diameter at most $O(\log n)$ was shown in [32]. Our

lower bound (Corollary 3.9) shows, in particular, a stronger (by a factor of $\sqrt{B \log n}$) lower bound on the running time of a protocol that computes the *MST* exactly or *approximates* it within *any constant factor* on graphs with diameters in the same range.

4. Dependence on the diameter. In this section we show that if the diameter is $\Omega(n^\epsilon)$ for some constant $\epsilon > 0$, then the lower bound of Corollary 3.9 can be strengthened by a factor of $\sqrt{\log n}$.

To this end, consider the following family $\tilde{\mathcal{G}}$ of unweighted graphs that contains one n-vertex graph $\tilde{G}_n$ for infinitely many positive integers n. Fix some n such that $\tilde{G}_n \in \tilde{\mathcal{G}}$, and denote $\tilde{G} = \tilde{G}_n$. The vertex set $\tilde{V}$ of the graph $\tilde{G}$ consists of Γ paths $P_1, P_2, \ldots, P_\Gamma$, each of length m^2 (as in the definition of the family $\mathcal{G}$, Γ and m are sufficiently large positive integers that will be fixed later on); i.e., for $j \in \{1, 2, \ldots, \Gamma\}$, $P_j = (v_j^{(0)}, v_j^{(1)}, \ldots, v_j^{(m^2)})$. In addition, the vertex set $\tilde{V}$ contains a path P^τ (in the family $\tilde{\mathcal{G}}$ the path P^τ plays a role that is analogous to the one that the tree τ plays in the family $\mathcal{G}$) of length m; i.e., $P^\tau = (s = z_0, z_1, \ldots, z_m = r)$. The edge set $\tilde{E}$ of the graph $\tilde{G}$ contains, in addition to the paths $P_1, P_2, \ldots, P_\Gamma$ and the path P^τ, the stars $S_0, S_1, \ldots, S_m$, where $S_i = \{(z_i, v_j^{(i \cdot m)}) \mid j \in \{1, 2, \ldots, \Gamma\}\}$ for $i \in \{0, 1, \ldots, m\}$. This completes the description of the family $\tilde{\mathcal{G}}$ of unweighted graphs. This family is a slight generalization of the family $\mathcal{G}'$ of unweighted graphs that was introduced in [32]. Specifically, in [32], $\Gamma = m^2$. Observe that the the diameter of the graph G is $\Omega(m)$.

Consider the CorruptedMail(α, β) problem on family $\tilde{\mathcal{G}}$ of unweighted graphs. We next argue that it requires $\Omega(\min\{m, \Gamma/B\})$ rounds of distributed computation. The proof proceeds similarly to the proof of the analogous fact for the family $\mathcal{G}$ of unweighted graphs (see section 3.2), but it is simpler because the graphs of the family $\tilde{\mathcal{G}}$ have a simpler structure than the graphs of family $\mathcal{G}$. The *tails* sets are $T_0 = \tilde{V} \setminus \{s\}$, $T_i = \{v_j^{(i)}, v_j^{(i+1)}, \ldots, v_j^{(m^2)} \mid j \in \{1, 2, \ldots, \Gamma\}\} \cup \{z_{i'} \mid i' \in \{\lfloor i/m \rfloor, \lfloor i/m \rfloor + 1, \ldots, m\}\}$, $i \in \{1, 2, \ldots, m\}$. As in Lemma 3.4, $T_0 \supseteq T_1 \supseteq \cdots \supseteq T_m$, and let $\tilde{E}_{i,i+1}$ be the set of edges that connect one of the vertices of $\tilde{V} \setminus T_i$ with one of the vertices of T_{i+1}. It is easy to see that $|E_{i,i+1}| \leq 1$ for $i \in \{0, 1, \ldots, m^2 - 1\}$. (Recall that the proof of an analogous upper bound, $|E_{i,i+1}| \leq p \cdot m^{1/p}$, for a graph $G \in \mathcal{G}$ is somewhat more complicated.) Now, a lemma that is analogous to Lemma 3.4 follows.

Lemma 4.1. *For $t \in \{0, 1, \ldots, m^2 - 1\}$, $\rho(T_t, t) \leq (2^{B+1} - 1)^t$.*

Exactly the same argument as the one that works for family $\mathcal{G}$ shows that any deterministic protocol for the CorruptedMail(α, β) problem on family $\tilde{\mathcal{G}}$ either runs for at least $t \geq m^2 - 1$ rounds or has a running time t that satisfies the inequality $\rho(\{r\}, t) \geq |\Upsilon'| = \Omega(2^{l(\alpha,\beta)\Gamma})$ (see inequality (3)). Hence,

$$(2^{B+1} - 1)^t \geq \rho(T_t, t) \geq \rho(\{r\}, t) = \Omega(2^{l(\alpha,\beta)\Gamma}) \ .$$

It follows that $t = \Omega(\min\{m^2, l(\alpha, \beta)\Gamma/B\})$. An argument analogous to the one that was presented above for the CorruptedMail(α, β) problem on the family $\mathcal{G}$ shows that the same (up to a constant factor) lower bound applies to any *randomized* protocol for the CorruptedMail(α, β) problem on the family $\tilde{\mathcal{G}}$ as well.

Finally, for the reduction to the *MST* problem, one needs to introduce the family $\tilde{\mathcal{G}}^\omega$ of weighted graphs. The choice of weights of edges is analogous to the choice of weights for family $\mathcal{G}^\omega$, and is different from the one suggested in [32], since it is geared toward guaranteeing that the delivery will suffer from only a *one-sided error*, as was

discussed above. Specifically, the edges of the paths $P_1, P_2, \ldots, P_\Gamma$, as well as the edges of the path P^τ, all have weight zero. The edges of the stars $S_1, S_2, \ldots, S_{m-1}$ all have weight ∞. The edges of the star S_m all have unit weights, and, finally, each edge of the star S_0 may weigh either 0 or ∞, so that exactly $\alpha\Gamma$ of them have weight ∞.

The reduction from the CorruptedMail(α, β) problem on family $\tilde{\mathcal{G}}$ to the $\frac{\beta}{\alpha}$-approximate *MST* problem on family $\tilde{\mathcal{G}}^\omega$ is identical to the reduction between these problems on families of graphs $\mathcal{G}$ and $\mathcal{G}^\omega$, respectively. The latter reduction was described in section 3.3. The analysis of the reduction, described in section 3.3, applies here as well. It follows that any randomized protocol for the *MST* problem on the family $\tilde{\mathcal{G}}^\omega$ of weighted graphs requires $t = \Omega(\min\{m^2, l(\alpha,\beta)\Gamma/B\})$ rounds. By (1), $t = \Omega(\min\{m^2, \alpha\cdot\Gamma/B\})$. Substitute $m = \sqrt{\alpha\cdot\Gamma/B}$, set $H = 1/\alpha$, and observe that the number of vertices n in $\tilde{G}$ is $O(\Gamma\cdot m^2)$. The diameter Λ of the graphs from the family $\mathcal{G}$ is $m+2 = O\big(\big(\frac{n}{HB}\big)^{\frac{1}{4}}\big)$. It is easy to see that by replacing the path P^τ with k paths $P_1^\tau, P_2^\tau, \ldots, P_k^\tau$ of decreasing lengths $m, m^{1/2}, m^{1/4}, \ldots, m^{1/2^k}$ we obtain a similar lower bound for graphs of diameter $\Lambda = O\big(\big(\frac{n}{HB}\big)^{\frac{1}{2^{k+2}}}\big)$. We conclude with the following theorem.

Theorem 4.2. *For any constant $\delta > 0$, any randomized H-approximation protocol for the MST problem on graphs of diameter at most $O(n^\delta)$ requires $T = \Omega(\sqrt{\frac{n}{H\cdot B}})$ rounds of distributed computation; i.e., $T^2\cdot H = \Omega(\sqrt{\frac{n}{B}})$. In particular, for any $\epsilon > 0$, an $\big(\frac{n}{B}\big)^{1-\epsilon}$-approximation requires $\Omega(\big(\frac{n}{B}\big)^{\frac{\epsilon}{2}})$ rounds.*

In particular, this improves the result of [32] by a factor of $\sqrt{B}$. Specifically, [32] has shown a lower bound of $\Omega(\sqrt{n}/B)$ on the number of rounds required for the exact computation of the *MST*, while our result gives a stronger (by a factor of $\sqrt{B}$) lower bound on the number of rounds required to approximate the *MST* within any constant approximation factor.

Similarly, our technique enables us to get a lower bound on the time-approximation trade-off for the *MST* problem on graphs of diameter 3. This is done by replacing the tree τ in the family $\mathcal{G}$ of unweighted graphs with a clique of size $m+1$. The clique consists of original "leaves" $s = z_0, z_1, \ldots, z_m = r$. The obtained family generalizes the family $\mathcal{G}''$ due to [28]. In [28] the family $\mathcal{G}''$ proved a lower bound of $\Omega(n^{1/4}/B)$ on the number of rounds required for exact computation of the *MST* on graphs of diameter 3. In construction of [28], $\Gamma = m^3$.

The tail sets are defined exactly as for the family $\mathcal{G}$, except that $\tau(i) = \{z_i, z_{i+1}, \ldots, z_m\}$ for $i \in \{1, 2, \ldots, m\}$. It is easy to see that the edge sets $E_{i,i+1}$ are of cardinality $O(m^2)$ for each index i, implying that $\rho(T_t, t) \le (2^{B+1}-1)^{t\cdot m^2}$, for each $t \in \{0, 1, \ldots, m-1\}$. The rest of the proof is identical to that described above, and it yields a lower bound of $\Omega(\min\{m, \frac{\Gamma\cdot\alpha}{m^2\cdot B}\})$ on the number of rounds required for any randomized protocol for a $\frac{\beta}{\alpha}$-approximate *MST* on graphs of diameter 3. Substitute $m = (n\cdot\alpha/B)^{1/4}$, $H = 1/\alpha$, and observe that the number of vertices n is $O(\Gamma\cdot m)$. The next theorem follows.

Theorem 4.3. *Any randomized H-approximation protocol for the MST problem on graphs of diameter at most 3 requires $T = \Omega(\big(\frac{n}{H\cdot B}\big)^{\frac{1}{4}})$ rounds of distributed computation; i.e., $T^4\cdot H = \Omega(\frac{n}{B})$. In particular, for any $\epsilon > 0$, $\big(\frac{n}{B}\big)^{1-\epsilon}$-approximation requires $\Omega(\big(\frac{n}{B}\big)^{\frac{\epsilon}{4}})$ rounds.*

In particular, this improves the lower bound of [28] for exact computation of the *MST* on graphs of diameter 3 by a factor of $B^{1/4}$.

Observe that our time-approximation trade-off cannot be generalized further to

graphs of diameter at most 2, as there is a protocol for exact computation of the *MST* in $O(\log n)$ rounds on graphs of diameters 1 and 2 (due to [36, 28, 29]).

5. Dependence on ω_{max}: Asymmetric two-sided error. Observe that in the lower bounds that we described in the previous sections, the ratio ω_{max} between the maximum weight of an edge in a graph and the minimum weight is infinity. Actually, the weights ∞ and 0 were introduced for clarity of presentation, and one can carry on the proof if $\omega_{max} = O(n^2)$. However, allowing ω_{max} to get even smaller than that disables the reduction from the CorruptedMail problem to the *MST* problem. Specifically, it can happen that a protocol for the *MST* problem will produce a $\frac{\beta}{\alpha}$-approximate *MST* τ_0, and this tree τ_0 will induce (through the reduction) an output bit string χ' for the CorruptedMail problem that will not dominate the input bit string χ. This is due to the fact that now some edges of "infinite" weight can be taken into the approximate *MST* τ_0, while still keeping a relatively small total weight of τ_0. However, fortunately, if ω_{max} is at least $\frac{\sqrt{n}}{\alpha^{3/2}}$, the number of errors of the second type (i.e., indices j such that $\chi_j = 1$ and $\chi'_j = 0$) is small. This enables us to prove a time-approximation trade-off similar to that of Theorem 3.8 for ω_{max} that is much smaller than n^2 (specifically, $\Omega(\frac{\sqrt{n}}{\alpha^{3/2}})$). As was already mentioned, due to our upper bound on the distributed approximation of *MST* (section 6), such a trade-off becomes impossible for $\omega_{max} = o(\sqrt{n \cdot H/B}/\log^* n) = o(\sqrt{n/(B \cdot \alpha)} \cdot \log^* n)$.

The proof utilizes the following observation that may be interesting in its own right. Consider a bit string χ of length Γ of Hamming weight $\alpha\Gamma$. Suppose that an adversary is allowed to introduce into χ a *huge* number of corruptions of the first type (i.e., to change from 0 to 1 at most $(\beta-\alpha-\epsilon)\Gamma$ positions for some constant arbitrarily small positive α and ϵ and constant $\beta < 1$ that is arbitrarily close to 1), and it is allowed to introduce a reasonable number of corruptions of the second type (i.e., to change from 1 to 0 at most a constant fraction of $\varrho \cdot \alpha\Gamma$ of positions for some universal constant $0 < \varrho < 1 - 2\epsilon$). Then the obtained bit string χ' still carries on a significant fraction of the entropy that was carried on by the original bit string χ!

To capture this intuition, we introduce the AsymMail (asymmetrically corrupted mail) problem. This problem generalizes the CorruptedMail problem that was discussed in the previous sections. The AsymMail(α, β) problem is defined on the same family $\mathcal{G}$ of unweighted graphs as the CorruptedMail(α, β) problem. The vertex s accepts as input a bit string χ of Hamming weight $\alpha\Gamma$, and the vertex r outputs a bit string χ' of Hamming weight at most $\beta\Gamma$. It is required that for each index $j \in \{1, 2, \ldots, \Gamma\}$ such that $\chi_j = 1$, $\chi'_j = 1$ as well, *except for* $\frac{\alpha}{10}\Gamma$ *indices* j. For a pair of bit strings $\chi, \chi' \in \{0,1\}^\Gamma$ that satisfy the above condition, we will say that the bit string χ' *almost dominates* the bit string χ.

Consider a bipartite graphs $(\mathcal{A}, \Upsilon, E(\mathcal{A}, \Upsilon))$, with $E(\mathcal{A}, \Upsilon) = \{(\chi, \chi') \mid \chi \in \mathcal{A}, \chi' \in \Upsilon, \chi' \text{ almost dominates } \chi\}$. Fix some bit string $\chi' \in \Upsilon$ with Hamming weight $\lambda\Gamma$, $\frac{9}{10}\alpha \le \lambda \le \beta$. The number of the bit strings $\chi \in \mathcal{A}$ that are almost dominated by χ' is at most

$$D = \sum_y \binom{\lambda\Gamma}{(\alpha - y)\Gamma} \cdot \binom{(1-\lambda)\Gamma}{y\Gamma}, \tag{6}$$

where the index y runs over $y \in \{0, 1/\Gamma, 2/\Gamma, \ldots, \alpha/10\}$ (the last index y of this sum is the biggest multiple of $1/\Gamma$ that is smaller than or equal to $\alpha/10$).

The expression $\binom{\lambda\Gamma}{(\alpha-y)\Gamma}\binom{(1-\lambda)\Gamma}{y\Gamma}$ is a monotone increasing function of y whenever

$0 \leq y \leq \alpha(1-\lambda)$. To ensure that y is in this range in (6), it is sufficient to fix $\beta = 3/4$ (or, actually, any other constant smaller than $9/10$). For this choice of β,

$$D \;\leq\; (\alpha/10)\Gamma \cdot \binom{\lambda\Gamma}{(9\alpha/10)\Gamma}\binom{(1-\lambda)\Gamma}{(\alpha/10)\Gamma}\;. \tag{7}$$

Let $g(\lambda)$ denote the logarithm of base 2 of the right-hand side expression in (7). This function has a positive derivative for $\lambda < 9/10$. Hence, setting $\lambda = \beta < 9/10$ maximizes this expression. It follows that

$$D \;\leq\; (\alpha/10)\Gamma \cdot \binom{\beta\Gamma}{(9\alpha/10)\Gamma}\binom{(1-\beta)\Gamma}{(\alpha/10)\Gamma}\;.$$

A subset $\Upsilon' \subseteq \Upsilon$ is said to *almost dominate* the set $\mathcal{A}$ if for each bit string $\chi \in \mathcal{A}$ there exists a bit string $\chi' \in \Upsilon'$ that almost dominates χ.

It follows that any subset $\Upsilon' \subseteq \Upsilon$ that almost dominates $\mathcal{A}$ has cardinality at least

$$|\Upsilon'| \;\geq\; (|\mathcal{A}|/D) \;\geq\; \frac{\binom{\Gamma}{\alpha\Gamma}}{(\alpha/10)\Gamma\binom{\beta\Gamma}{(9\alpha/10)\Gamma}\binom{(1-\beta)\Gamma}{(\alpha/10)\Gamma}}\;.$$

The factor $(\alpha/10)\Gamma$ affects only lower-order terms of the lower bound, and, therefore, can be ignored. Using the Stirling formula, and analyzing the logarithm $\hat{l}(\alpha,\beta)$ of the right-hand side expression, we get $|\Upsilon'| \geq 2^{\hat{l}(\alpha,\beta)\Gamma}$, with $\hat{l}(\alpha,\beta) = \hat{l}(0,\beta) + \frac{\partial \hat{l}}{\partial \alpha}(\alpha,\beta)\cdot\alpha + o(\alpha) = \alpha(\frac{9}{10}\log(\frac{1}{\beta}\cdot\frac{9}{10}) + \frac{1}{10}\log(\frac{1}{1-\beta}\cdot\frac{1}{10})) + o(\alpha)$.

Consider $f(q) = q\cdot\log(\frac{1}{\beta}\cdot q) + (1-q)\cdot\log\frac{1-q}{1-\beta}$. Note that $f(1) = \lim_{q\to 1} f(q) = \log\frac{1}{\beta}$, and that $f'(q) = c\cdot\log\left(\frac{q}{1-q}\cdot\frac{1-\beta}{\beta}\right)$ for some universal positive constant $c > 0$. Hence, $f'(q) > 0$ whenever $q > \beta$. As we set $q = 9/10$ and $\beta = 3/4$, it follows that $f(9/10) < \log 1/\beta$. Recall that $l(\alpha,\beta) = \alpha\cdot\log 1/\beta + o(\alpha)$ and that $\hat{l}(\alpha,\beta) < l(\alpha,\beta)$ is consistent with the intuition that the entropy of the corrupted bit string when two-sided errors are allowed is smaller than the entropy of the corrupted bit string when only one-sided errors are allowed.

Now, exactly the same argument that counts the number of possible configurations of the vertices of the graph $G \in \mathcal{G}$ shows the following lemma.

LEMMA 5.1. *Any protocol for the* AsymMail(α,β) *problem on a family $\mathcal{G}$ of graphs requires* $\Omega(\min\{m, \hat{l}(\alpha,\beta)\frac{\Gamma}{p\cdot m^{1/p}\cdot B}\})$ *rounds.*

Recalling that $n = O(\Gamma\cdot m)$ and setting $m = (\frac{n\cdot\alpha}{p\cdot B})^{\frac{1}{2}-\frac{1}{2(2p+1)}}$, we see that a lower bound of $\Omega(m) = \Omega((\frac{n\cdot\alpha}{p\cdot B})^{\frac{1}{2}-\frac{1}{2(2p+1)}})$ follows (this lower bound is only by a constant factor smaller than the analogous lower bound (5) for the CorruptedMail problem).

We next describe the reduction from the AsymMail problem to the approximate *MST* problem with "small" ω_{max}. Consider the family $\mathcal{G}^{\omega m}$ of weighted graphs. This family is constructed out of the family $\mathcal{G}^{\omega}$ of weighted graphs by the following mapping ψ. Given a graph $G = (V,E) \in \mathcal{G}^{\omega}$, the mapping ψ constructs a graph in $\mathcal{G}^{\omega m}$ with the same vertex set and the same edge set, but with slightly different edge weights. Let ω_1, $\omega_2 > 1$ be two positive integer parameters to be fixed later. For an edge $e \in E$, the new weight $\omega'(e)$ is determined by

$$\omega'(e) = \begin{cases} 1, & \omega(e) = 0, \\ \omega_1, & \omega(e) = 1, \\ \omega_1\cdot\omega_2, & \omega(e) = \infty. \end{cases} \tag{8}$$

Consider the same reduction that was described in section 3.3 that, given an instance of the CorruptedMail problem, produces an instance of the *MST* problem. Consider a composition of this reduction with the mapping ψ. Let $\alpha > 0$ be at most a small positive constant, and let β' satisfy $\alpha < \beta' < \frac{100}{101}(\beta - \alpha)$. We obtain a reduction from the AsymMail(α, β) problem to the $\frac{\beta'}{\alpha}$-approximate *MST* problem on graphs with $\omega_{max} = \omega_1 \cdot \omega_2$.

Consider a weighted graph $G(\chi) \in \mathcal{G}^{\omega m}$ that is formed by the reduction. Recall that χ is the input bit string of the AsymMail problem, and its Hamming weight is $\alpha \cdot \Gamma$. It follows that precisely $\alpha \cdot \Gamma$ edges of the star S_0 have weight $\omega_1 \cdot \omega_2$ (this is the analogue of ∞), and all remaining $(1-\alpha) \cdot \Gamma$ edges of this star have unit weights. It is easy to see that the *MST* of $G(\chi)$ contains all the edges of unit weight, and, in addition, precisely $\alpha \cdot \Gamma$ edges of weight ω_1 of the star S_m.

It follows that the weight of the *MST*, $\omega'(MST) = \alpha\Gamma \cdot \omega_1 + (n-1-\alpha\Gamma)$. Consider the $\frac{\beta'}{\alpha}$-approximate *MST* τ_0 that is returned by the oracle approximation protocol for *MST* (its existence is assumed by the reduction). By definition, $\omega'(MST) \leq \omega'(\tau_0) \leq \frac{\beta'}{\alpha} \cdot (\alpha\Gamma \cdot \omega_1 + (n-1-\alpha\Gamma))$. Consider some index $j \in \{1, 2, \ldots, \Gamma\}$ such that $\chi_j = 1$, and suppose that $\chi'_j = 0$. As $\chi_j = 1$, the weight of the edge $(s, v_j^{(0)})$ is $\omega_1 \cdot \omega_2$. As $\chi'_j = 0$, the edge $(r, v_j^{(m)})$ does not belong to the tree τ_0. As τ_0 is a spanning tree of $G(\chi)$, it follows that at least one edge from the set $HE_j = \{(z_i, v_j^{(i)}) \mid i \in \{0, 1, \ldots, m-1\}\}$ belongs to the tree τ_0 (*HE* stands for "heavy edges"). Observe that for each edge $e \in HE_j$, $\omega'(e) = \omega_1 \cdot \omega_2$. Note also that for two distinct indices $j \neq j'$, $j, j' \in \{1, 2, \ldots, \Gamma\}$, $HE_j \cap HE_{j'} = \emptyset$. It follows that if there are q indices j such that $\chi_j = 1$ and $\chi'_j = 0$, then at least q edges of weight $\omega_1 \cdot \omega_2$ belong to approximate *MST* τ_0; i.e., $q \cdot \omega_1\omega_2 \leq \beta'\Gamma \cdot \omega_1 + \frac{\beta'}{\alpha}(n-1-\alpha\Gamma)$. As we allow at most $\alpha\Gamma/10$ errors of the second type, we impose the following condition on ω_1 and ω_2:

$$\left(\frac{\alpha}{10}\Gamma + 1\right)\omega_1 \cdot \omega_2 > \beta' \cdot \Gamma \cdot \omega_1 + \frac{\beta'}{\alpha}(n-1-\alpha\Gamma)\ . \tag{9}$$

Consider some index $j \in \{1, 2, \ldots, \Gamma\}$ such that $\chi_j = 0$, and suppose that $\chi'_j = 1$. As $\chi_j = 0$, $\omega'(s, v_j^{(0)}) = 1$. As $\chi'_j = 1$, the edge $(r, v_j^{(m)})$ belongs to the tree τ_0. Recall that $\omega'((r, v_j^{(m)})) = \omega_1$. It follows that $\omega'(\tau_0) \geq \omega_1 \cdot (||\chi'|| - ||\chi||) = \omega_1 \cdot (||\chi'|| - \alpha\Gamma)$; i.e., $\omega_1 \cdot (||\chi'|| - \alpha\Gamma) \leq \omega'(\tau_0) \leq \beta'\Gamma \cdot \omega_1 + \frac{\beta'}{\alpha}(n-1-\alpha\Gamma)$. We next impose the following condition on ω_1:

$$\Gamma \cdot \omega_1 \geq 100\frac{n}{\alpha}\ . \tag{10}$$

Then $\beta' \cdot (\Gamma\omega_1 + \frac{n-1-\alpha\Gamma}{\alpha}) \leq \beta' \cdot (\Gamma\omega_1 + \frac{n}{\alpha}) \leq \frac{101}{100}\beta'\Gamma \cdot \omega_1$. It follows that $||\chi'|| - \alpha\Gamma \leq \frac{101}{100}\beta'\Gamma$. Recall that $\beta' < \frac{100}{101}(\beta - \alpha)$. Hence, $||\chi'|| \leq (\frac{101}{100}\beta' + \alpha)\Gamma < \beta\Gamma$. Hence, under conditions (9) and (10), the output bit string χ' almost dominates the input bit string χ. It follows that under these conditions, the AsymMail(α, β) problem reduces to a $\frac{\beta'}{\alpha}$-approximate *MST* problem for any $\beta' < \frac{100}{101}(\beta - \alpha)$.

Conditions (9) and (10) determine how large ω_1 and ω_2 should be for this reduction to work. Inequality (10) means that $\omega_1 \geq 100\frac{n}{\Gamma \cdot \alpha}$, and to satisfy (9), it is enough to ensure that $\frac{\alpha}{10}\Gamma \cdot \omega_1 \cdot \omega_2 > \beta'\frac{101}{100}\Gamma\omega_1$. The latter implies that $\omega_2 > \frac{\beta'}{\alpha} \cdot \frac{101}{10}$. Hence, $\omega_{max} = \omega_1 \cdot \omega_2 > 1010 \cdot \beta' \cdot \frac{n}{\Gamma \cdot \alpha^2}$. As β' is a fixed constant ($\beta' < 1$), it follows that $\omega_{max} = \Theta(\frac{n}{\Gamma \cdot \alpha^2})$ is enough. Combining this with the lower bound on the AsymMail(α, β) problem (Lemma 5.1) implies that approximating *MST*

within a ratio of $\Theta(\frac{1}{\alpha})$ on graphs of diameter $2p+2$ with $\omega_{max} = \Theta(\frac{n}{\Gamma \cdot \alpha^2})$ requires $\Omega(\min\{m, \hat{l}(\alpha,\beta)\frac{\Gamma}{p \cdot m^{1/p} \cdot B}\})$ rounds for some constant $\beta < 1$.

Recall that $n = O(\Gamma \cdot m)$. Set $H = \frac{1}{\alpha}$, and $m = \left(\frac{n \cdot \alpha}{p \cdot B}\right)^{\frac{1}{2} - \frac{1}{2(2p+1)}}$. We conclude with the following.

THEOREM 5.2. *Any randomized H-approximation protocol for the MST problem on graphs of diameter at most Λ for $\Lambda \in \{4, 6, \dots\}$ with $\omega_{max} = \Theta\left(\left(\frac{n}{\Lambda \cdot B}\right)^{\frac{1}{2} - \frac{1}{2(\Lambda-1)}} \cdot H^{\frac{3}{2} + \frac{1}{2(\Lambda-1)}}\right)$ requires at least $T = \Omega\left(\left(\frac{n}{\Lambda \cdot H \cdot B}\right)^{\frac{1}{2} - \frac{1}{2(\Lambda-1)}}\right)$ rounds; i.e., $T^{2 + \frac{2}{\Lambda-2}} \cdot H = \Omega(\frac{n}{\Lambda \cdot B})$. In particular, for $\Lambda = \Theta(\log n)$, $\omega_{max} = \Theta\left(\left(\frac{n}{B \cdot \log n}\right)^{\frac{1}{2}} \cdot H^{3/2}\right)$, and $T^2 \cdot H = \Omega\left(\frac{n}{B \cdot \log n}\right)$.*

By setting m to be smaller than $\left(\frac{n}{HpB}\right)^{\frac{1}{2} - \frac{1}{2(2p+1)}}$, one can get a smaller lower bound on T but with $\omega_{max} = \Theta(mH^2)$. Also, as for the case of unbounded ω_{max}, the factor $\log n$ in the denominator can be eliminated if one allows a bigger diameter (specifically, $O(n^\delta)$ for any constant positive $\delta > 0$). Also, this way one can get a lower bound for approximate *MST* on graphs with diameter 3 (Theorem 5.2 applies as is) by considering the ASYMMAIL(α, β) problem on the family $\mathcal{G}''$ of graphs (see Theorem 4.3 and the discussion that precedes it).

We remark that one cannot expect to get $\omega_{max} = o\left(\sqrt{\frac{n \cdot H}{B}} / \log^* n\right)$ in Theorem 5.2. This is because by Theorem 6.4, there exists an $O(\frac{\omega_{max}}{T'})$-approximation protocol for the *MST* problem with running time $O(\Lambda(G) + T' \log^* n)$. For ω_{max} as above, the approximation ratio is $H = O(\frac{\omega_{max}}{T'}) = o(\sqrt{\frac{n}{B}} \frac{1}{\log^* n} \cdot \frac{\sqrt{H}}{T'})$; i.e., $T' = o(\sqrt{\frac{n}{BH}} \frac{1}{\log^* n})$, and the running time of the protocol is $T = O(\Lambda(G)) + o(\sqrt{\frac{n}{BH}}) = o(\sqrt{\frac{n}{BH}})$ (assuming that $\Lambda(G) = o(\sqrt{\frac{n}{BH}})$; indeed, the lower bounds apply for a small diameter, and the upper bound apply for an arbitrarily large diameter). This protocol provides an H-approximation for the *MST* problem. Hence, $T^2 \cdot H = o(\sqrt{n/B})$, contradicting the stronger (by a factor of $\log n$) version of Theorem 5.2 (that applies for $\Lambda(G) = O(n^\delta)$ for any constant $\delta > 0$).

6. An upper bound. In this section we devise a distributed protocol for the approximate *MST* problem. Our protocol runs in $O(\Lambda(G) + n^\epsilon \cdot \log^* n)$ rounds and constructs an $O(\frac{\omega_{max}}{n^\epsilon})$-approximate *MST*, where ω_{max} is the ratio between the weights of the heaviest and the lightest edges in the graph G, and ϵ is any fixed number between zero and one. Note that the approximation ratio of $O(\omega_{max})$ is trivial.

Throughout this section we assume that $B = \log n$. Hence, the protocol can be used whenever $B = \Omega(\log n)$, and its running time will be the same. Also, obviously it can be adapted to the case $B = o(\log n)$ incurring an overhead of $O(\frac{\log n}{B}) = O(\log n)$. Also, we assume that the weights of edges are scaled between 1 and ω_{max}.

Consider an *MST* τ_0 of the graph G. A connected subtree of τ_0 is called a *fragment* of τ_0. When the *MST* τ_0 can be understood from the context, such a fragment will be called an *MST fragment*. A *k-MST forest* $\mathcal{F}$ of a graph $G = (V, E)$ is a collection of vertex-disjoint trees that satisfy the following properties:

1. $\bigcup_{T \in \mathcal{F}} V(T) = V$, $\bigcup_{T \in \mathcal{F}} E(T) \subseteq E$.
2. $|V(T)| = \Omega(k)$, $\Lambda(T) = O(k)$.
3. There exists an *MST* τ_0 for the graph G, such that each tree $T \in \mathcal{F}$ is its fragment.

The notion of a *k-MST* forest is related to the notion of the (σ, ρ) spanning forest of [23]. Trees $T \in \mathcal{F}$ of a (σ, ρ) spanning forest $\mathcal{F}$ have to satisfy properties 1 and 2 with $|V(T)| \geq \sigma$ and $\Lambda(T) \leq \rho$, but may not satisfy property 3. It was demonstrated

in [23] that a k-*MST* forest of an n-vertex graph G can be constructed in $O(k \cdot \log^* n)$ rounds of distributed computation.

The first step of our n^ϵ-approximation protocol for the *MST* problem is to construct an n^ϵ-*MST* forest $\mathcal{F}$. This requires $O(n^\epsilon \cdot \log^* n)$ rounds. The second step is to construct a breath first search (BFS) spanning tree τ of the entire graph, rooted at some arbitrary vertex $rt = rt(\tau)$. This requires $O(\Lambda(G))$ rounds. After the construction of the tree τ, each vertex v in the graph knows its unweighted distance to the root rt, $dist_\tau(rt, v)$. At the third step of the protocol, convergecasts are conducted in parallel over the spanning trees of the fragments $T \in \mathcal{F}$ of the n^ϵ-*MST* forest $\mathcal{F}$. Throughout the convergecast over a fragment T, the root of the fragment T learns the identity of the vertex $v \in V(T)$ that is closest in τ to the root rt of τ. The draws are broken arbitrarily. The fourth step involves broadcasts over the spanning trees of the fragments of the identities of these chosen vertices. Both convergecasts and broadcasts are done in parallel (recall that the fragments are vertex disjoint) and require $O(\max\{\Lambda(T) \mid T \in \mathcal{F}\})$ rounds. As $\mathcal{F}$ is an n^ϵ-*MST* forest, it follows that $\Lambda(T) = O(n^\epsilon)$ for every tree $T \in \mathcal{F}$. Hence, these steps require $O(n^\epsilon)$ rounds. After the broadcasts are done, the fifth step occurs. On the fifth step in each fragment $T \in \mathcal{F}$, the chosen vertex $v \in V(T)$ inserts the edge $e_v = (par_{(\tau, rt(\tau))}(v), v)$ into the tree τ_0 that the protocol constructs. It also informs the parent of the vertex v in τ, $par_{(\tau, rt(\tau))}(v)$, that it was chosen, and the parent inserts the edge e_v into τ_0 as well. In addition, each vertex w in the graph inserts into the tree τ_0 the edges of the k-*MST* forest $\mathcal{F}$ that are incident to w. Observe that the fifth step requires only one round of distributed computation (for sending the messages by the chosen vertices to their parents in τ).

This completes the description of the protocol. It follows from our discussion that its running time is $O(\Lambda(G) + n^\epsilon \cdot \log^* n)$. We next argue that it is indeed an $O(\frac{\omega_{max}}{n^\epsilon})$-approximation protocol for the *MST* problem.

LEMMA 6.1. *The subgraph τ_0 that is constructed by the protocol is acyclic.*

Proof. Suppose for contradiction that there is a cycle $((u_0, w_0), P_0, (u_1, w_1), P_1, \ldots, (u_{t-1}, w_{t-1}), P_{t-1})$, where P_i is a path in some fragment T_i between w_i and $u_{((i+1) \bmod t)}$, and $(u_i, w_i) \in E(\tau)$, $i \in \{0, 1, \ldots, t-1\}$ (where τ is the BFS spanning tree of the graph). Fragments may appear more than once. (Observe that the cycle cannot be contained entirely in one fragment, because for each fragment $T \in \mathcal{F}$, the edges of τ_0 with both endpoints in T all belong to the spanning tree of T.) It follows that $|dist_\tau(rt, u_i) - dist_\tau(rt, w_i)| = 1$ for $i \in \{0, 1, \ldots, t-1\}$.

Assume, without loss of generality that $dist_\tau(rt, w_0) - dist_\tau(rt, u_0) = 1$; i.e., $u_0 = par_{(\tau, rt)}(w_0)$. Then $u_1 = par_{(\tau, rt)}(w_1)$, because otherwise both edges $(u_0, w_0), (u_1, w_1)$ connect vertices of the same fragment (the vertices w_0 and u_1) to their parents in the BFS tree τ. However, at most one such edge is inserted into the tree τ_0 for each fragment in the approximation protocol. It follows that $u_i = par_{(\tau, rt)}(w_i)$ for $i \in \{0, 1, \ldots, t-1\}$. Hence,

$$dist_\tau(rt, u_0) > dist_\tau(rt, w_0) \geq dist_\tau(rt, u_1) > \cdots \geq \cdots > dist_\tau(rt, w_{t-1}) \geq dist_\tau(rt, u_0) .$$

This is a contradiction, implying that the subgraph τ_0 is acyclic. □

LEMMA 6.2. *The subgraph τ_0 is connected, and it is spanning all the vertices of the graph G.*

Proof. The second assertion follows directly from the observation that the edge set of the tree τ_0 contains the edge set of the n^ϵ-*MST* forest $\mathcal{F}$, and from the definition of a k-*MST* forest.

For the first assertion, consider some pair of vertices u and w in V. Let T_u and T_w be the pair of fragments of the n^ϵ-*MST* forest $\mathcal{F}$ such that $u \in T_u$, $w \in T_w$. If $T_u = T_w$, then u and w are connected in τ_0, because the subgraph τ_0 contains a spanning tree of each fragment T of the n^ϵ-*MST* forest $\mathcal{F}$, and, in particular, of $T_u = T_w$. Otherwise, let $u_0 \in T_u$ (resp., $w_0 \in T_w$) be the closest vertex in T_u (resp., T_w) to $rt = rt(\tau)$ (in terms of the unweighted distance). To prove that there is a path between u and w in τ_0, it suffices to prove that there is a path between u_0 and w_0. We prove this by induction on $dist_\tau(rt, u_0) + dist_\tau(rt, w_0)$.

The induction base is the case when the sum is 0, i.e., $u_0 = w_0 = rt$, and then the assertion is obvious.

For the induction step, consider the parent $v = par_{(\tau,rt)}(u_0)$ of the vertex u_0 in the BFS tree τ. Observe that $dist_\tau(rt, v) = dist_\tau(rt, u_0) - 1 < dist_\tau(rt, u_0)$, and that the edge $e = (u_0, v)$ belongs to τ_0. Thus, it suffices to prove that there is a path between v and w_0 in τ_0. Let T_v be the fragment of $\mathcal{F}$ that contains the vertex v, and let v_0 be the vertex that was chosen by the convergecast on this fragment. It follows that $dist_\tau(rt, v_0) \leq dist_\tau(rt, v) < dist_\tau(rt, u_0)$, and, thus, the induction hypothesis is applicable to the pair of vertices, v_0 and w_0. As the fragment T_v is connected, the lemma follows. □

It follows from Lemmas 6.1 and 6.2 that τ_0 is a spanning tree of the graph G.

LEMMA 6.3. *The tree τ_0 is a $(1 + O(\frac{\omega_{max}}{n^\epsilon}))$-approximate MST of the graph G.*

Proof. Let $\omega : E \to R^+$ denote the weight function that is associated with the graph G, and let $\omega(MST)$ denote the weight of the *MST*. Observe that the weight of the n^ϵ-*MST* forest $\mathcal{F}$ is at most $\omega(MST)$, and that the convergecast and broadcast procedures insert into τ_0 at most $O(n^{1-\epsilon})$ additional edges. Hence, $\omega(\tau_0) = O(n^{1-\epsilon} \cdot \omega_{max}) + \omega(MST)$. Thus,

$$\frac{\omega(\tau_0)}{\omega(MST)} = 1 + O\left(\frac{n^{1-\epsilon} \cdot \omega_{max}}{\omega(MST)}\right) .$$

Recall that by our assumption, all the weights are scaled between 1 and ω_{max}. Hence, $\omega(MST) \geq n - 1$. It follows that $\omega(\tau_0)/\omega(MST) = O(\omega_{max}/n^\epsilon)$. □

To conclude, we have the following.

THEOREM 6.4. *For any $0 < \epsilon < 1$, there exists a protocol that constructs an H-approximate MST for an n-vertex weighted graph (G, ω) in $O(\Lambda(G) + n^\epsilon \cdot \log^* n)$ rounds with $H = 1 + O(\frac{\omega_{max}}{n^\epsilon})$. Let $T = n^\epsilon \cdot \log^* n$. Then $T \cdot H = O(\omega_{max} \cdot \log^* n)$.*

Note that for graphs with small ω_{max} (e.g., constant, or polylogarithmic in n) this protocol provides an approximation ratio that is arbitrarily close to 1, and the running time of the protocol is arbitrarily close to $O(\Lambda(G))$.

7. Discussion. In this paper we presented a lower bound on the time-approximation trade-off of the distributed *MST* problem. While this is one of the most fundamental problems in the area of distributed computing, there are many other important distributed problems whose approximation behavior is yet to be explored. In our opinion, it would be of particular interest to establish lower bounds on time-approximation trade-offs of such problems as maximum dominating set and maximum matching. Interrelating the complexities of these problems via distributed reductions appears to be an even greater challenge.

Acknowledgments. The author is grateful to Michael Langberg, Zvika Lotker, Alessandro Panconesi, David Peleg, Alexander Razborov, Oded Regev, Vitaly Rubinovich, and Avi Wigderson for helpful discussions. The author also wishes to thank

the anonymous referees for their suggestions and remarks that helped to improve the presentation in this paper.

REFERENCES

[1] B. Awerbuch, *Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems*, in Proceedings of the 19th ACM Symposium on Theory of Computing, ACM, New York, 1987, pp. 230–240.

[2] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg, *Near-linear cost sequential and distributed constructions of sparse neighborhood covers*, in Proceedings of the 34th IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 1993, pp. 638–647.

[3] B. Awerbuch and G. Gallager, *A new distributed algorithm to find breadth first search trees*, IEEE Trans. Inform. Theory, IT-33 (1987), pp. 315–322.

[4] Y. Bartal, J. W. Byers, and D. Raz, *Global optimization using local information with applications to flow control*, in Proceedings of the 38th IEEE Symposium on the Foundations of Computer Science, IEEE, Los Alamitos, CA, 1997, pp. 303–312.

[5] A. Czygrinow, M. Hanckowiak, and M. Karonski, *Distributed O(Delta log n)-edge-coloring algorithm*, in Proceedings of the 9th Annual European Symposium on Algorithms, Lecture Notes on Comput. Sci. 2161, Springer, Berlin, 2001, pp. 345–355.

[6] D. Bertsekas and R. G. Gallager, *Data Networks*, 2nd ed., Prentice–Hall International, London, 1992.

[7] A. Czygrinow, M. Hanckowiak, and E. Szymanska, *Distributed algorithm for approximating the maximum matching*, Discrete Appl. Math., 143 (2004), pp. 62–71.

[8] F. Chin and H. F. Ting, *An almost linear time and $O(n \log n + e)$ messages distributed algorithm for minimum-weight spanning trees*, in Proceedings of the 26th IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 1985, pp. 257–266.

[9] D. Dubhashi, A. Mei, A. Panconesi, J. Radhakrishnan, and A. Srinivisan, *Fast distributed algorithm for (weakly) connected dominating sets and linear-size skeletons*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2003, pp. 717–724.

[10] M. Elkin, *Computing almost shortest paths*, in Proceedings of the 20th ACM Symposium on Principles of Distributed Computing, ACM, New York, 2001, pp. 53–62.

[11] M. Elkin, *A faster distributed protocol for constructing a minimum spanning tree*, in Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (New Orleans, LA), ACM, New York, SIAM, Philadelphia, 2004, pp. 352–361.

[12] M. Elkin, *Unconditional lower bounds on the time-approximation tradeoffs for the distributed minimum spanning tree problem*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing (Chicago, IL), ACM, New York, 2004, pp. 331–340.

[13] M. Elkin, *An overview of distributed approximation: A survey*, ACM SIGACT News, 35 (2004), pp. 40–57.

[14] M. Elkin and J. Zhang, *Efficient algorithms for constructing $(1+\epsilon, \beta)$-spanners in the distributed and streaming models*, in Proceedings of the 23rd ACM Symposium on Principles of Distributed Computing, ACM, New York, 2004, pp. 160–168.

[15] T. Fischer, A. Goldberg, D. J. Haglin, and S. Plotkin, *Approximating matchings in parallel*, Inform. Process. Lett., 46 (1993), pp. 115–118.

[16] R. G. Gallager, P. A. Humblet, and P. M. Spira, *A distributed algorithm for minimum-weight spanning trees*, ACM Trans. Programming Lang. Syst., 5 (1983), pp. 66–77.

[17] J. A. Garay, S. Kutten, and D. Peleg, *A sublinear time distributed algorithm for minimum-weight spanning trees*, SIAM J. Comput., 27 (1998), pp. 302–316.

[18] E. Gafni, *Improvements in the time complexity of two message-optimal election algorithms*, in Proceedings of the 4th Symposium on Principles of Distributed Computing, ACM, New York, 1985, pp. 175–185.

[19] F. Grandoni, K. Konemann, A. Panconesi, and M. Sozio, *Primal-dual based distributed algorithms for vertex cover with semi-hard capacities*, in Proceedings of the 24th ACM Symposium on Principles of Distributed Computing (Las Vegas, NV), ACM, New York, 2005, pp. 118–125.

[20] D. Grable and A. Panconesi, *Nearly optimal distributed edge colouring in O(log log n) rounds*, Random Structures Algorithms, 10 (1997), pp. 385–405.

[21] M. Hańćkowiak, M. Karoński, and A. Panconessi, *On the distributed complexity of computing maximal matchings*, SIAM J. Discrete Math., 15 (2001), pp. 41–57.

[22] L. Jia, R. Rajaraman, and R. Suel, *An efficient distributed algorithm for constructing small dominating sets*, in Proceedings of the 20th ACM Symposium on Principles of Distributed Computing, ACM, New York, 2001, pp. 33–42.
[23] S. Kutten and D. Peleg, *Fast distributed construction of k-dominating sets and applications*, J. Algorithms, 28 (1998), pp. 40–66.
[24] F. Kuhn, T. Moscibroda, and R. Wattenhofer, *What cannot be computed locally!*, in Proceedings of the 23rd Symposium on the Principles of Distributed Computing (St. John's, Newfoundland, Canada), ACM, New York, 2004, pp. 300–309.
[25] F. Kuhn and R. Wattenhofer, *Constant-time distributed dominating set approximation*, in Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing (Boston, MA), ACM, New York, 2003, pp. 25–32.
[26] F. Kuhn and R. Wattenhofer, *Distributed Combinatorial Optimization*, Technical Report 426, Department of Computer Science, ETH, Zurich, 2004. Available online at http://www.inf.ethz.ch/research/disstechreps/techreports/techreports?range=400
[27] N. Linial, *Locality in distributed graph algorithms*, SIAM J. Comput., 21 (1992), pp. 193–201.
[28] Z. Lotker, B. Patt-Shamir, and D. Peleg, *Distributed MST for constant diameter graphs*, in Proceedings of the 20th ACM Symposium on Principles of Distributed Computing (Newport, RI), ACM, New York, 2001, pp. 63–72.
[29] Z. Lotker, B. Patt-Shamir, E. Pavlov, and D. Peleg, *Minimum-weight spanning tree construction in $O(\log \log n)$ communication rounds*, SIAM J. Comput., 35 (2005), pp. 120–131.
[30] T. Moscibroda and R. Wattenhofer, *Facility location: Distributed approximation*, in Proceedings of the 24th ACM Symposium on Principles of Distributed Computing (Las Vegas, NV), ACM, New York, 2005, pp. 108–117.
[31] D. Peleg, *Distributed Computing: A Locality-Sensitive Approach*, SIAM, Philadelphia, PA, 2000.
[32] D. Peleg and V. Rubinovich, *A near-tight lower bound on the time complexity of distributed MST construction*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 1999, pp. 253–261.
[33] A. Panconesi and A. Srinivasan, *Randomized distributed edge coloring via an extension of the Chernoff–Hoeffding bounds*, SIAM J. Comput., 26 (1997), pp. 350–368.
[34] C. Papadimitriou and M. Yannakakis, *Linear programming without matrix*, in Proceedings of the 25th ACM Symposium on Theory of Computing, ACM, New York, 1993, pp. 121–129.
[35] S. Rajagopalan and V. V. Vazirani, *Primal-dual RNC approximation algorithms for set cover and covering integer programs*, SIAM J. Comput., 28 (1998), pp. 525–540.
[36] V. Rubinovich, *Distributed Minimum Spanning Tree Construction*, M.Sc. Thesis, Bar-Ilan University, Ramat-Gan, Israel, 1999.
[37] A. Yao, *Probabilistic computations: Towards a unified measure of complexity*, in Proceedings of the 17th IEEE Symposium on Foundations of Computer Science (Providence, RI), IEEE, Los Alamitos, CA, 1977, pp. 222–227.

SIAM J. COMPUT.
Vol. 36, No. 2, pp. 457–497

TOWARD A TOPOLOGICAL CHARACTERIZATION OF ASYNCHRONOUS COMPLEXITY*

GUNNAR HOEST† AND NIR SHAVIT‡

Abstract. This paper introduces the use of topological models and methods, formerly used to analyze computability, as tools for the quantification and classification of *asynchronous complexity*. We present the first *asynchronous complexity theorem*, applied to decision tasks in the iterated immediate snapshot (IIS) model of Borowsky and Gafni. We do so by introducing a novel form of topological tool called the *nonuniform chromatic subdivision*. Building on the framework of Herlihy and Shavit's topological computability model, our theorem states that the time complexity of any asynchronous algorithm is directly proportional to the level of nonuniform chromatic subdivisions necessary to allow a simplicial map from a task's input complex to its output complex. To show the power of our theorem, we use it to derive a new tight bound on the time to achieve n process approximate agreement in the IIS model: $\lceil \log_d \frac{\max\_input - \min\_input}{\epsilon} \rceil$, where $d = 3$ for two processes and $d = 2$ for three or more. This closes an intriguing gap between the known upper and lower bounds implied by the work of Aspnes and Herlihy. More than the new bounds themselves, the importance of our asynchronous complexity theorem is that the algorithms and lower bounds it allows us to derive are intuitive and simple, with topological proofs that require no mention of concurrency at all.

Key words. shared memory, asynchronous systems, topology, immediate snapshots, approximate agreement, simplicial complexes, subdivisions

AMS subject classifications. 68Q10, 68Q17, 68Q60, 68Q85

DOI. 10.1137/S0097539701397412

1. Introduction.

In the 21st century, computers are progressively being used as coordination devices in asynchronous, distributed systems. Unfortunately, the standard, Turing notions of computability and complexity are not sufficient for evaluating the behavior of such systems. In the last few years, techniques of modeling and analysis based on classical algebraic topology [3, 12, 14, 18, 20, 21, 22, 24, 25, 26, 32] in conjunction with distributed simulation methods [9, 10, 11, 12] have brought about significant progress in our understanding of computability problems in an asynchronous distributed setting. We feel the time is ripe to extend these techniques to address *asynchronous complexity*.

This paper studies asynchronous shared memory solutions to the class of problems called *decision tasks*, input/output problems in which N processes start with input values and, after communicating, halt with private output values. We focus on the *iterated immediate snapshot* (IIS) memory model introduced by Borowsky and Gafni [12] as part of their new simplified proof of the asynchronous computability theorem [26]. The model is a restriction of atomic snapshot memory that guarantees that processes' scan operations return views that contain nondecreasing sets of the

*Received by the editors November 5, 2001; accepted for publication (in revised form) June 20, 2005; published electronically July 31, 2006. A preliminary version of this paper appeared in *Proceedings of the Sixteenth Annual ACM Symposium on the Principles of Distributed Computing*, 1997, pp. 199–208.

http://www.siam.org/journals/sicomp/36-2/39741.html

†Laboratory for Computer Science, MIT, Cambridge, MA 02139 (gwhoest@mit.edu).

‡Computer Science Department, Tel-Aviv University, Ramat Aviv, 69978, Israel (shanir@cs.tau.ac.il). Current address: Sun Microsystems Laboratories, Burlington, MA 01803. Most of this work was performed while this author was at MIT under Israel Science Foundation grant 03610882 and NSF grant CCR-9520298.

participating processes' inputs. Though it is not a realistic computation model (no machine supports such operations) we believe it is a good first candidate for topological modeling since it has a particularly nice geometric representation and hence easily lends itself to topological analysis.

1.1. Historical background and related work. Let us begin by giving a brief account of previous work on computability problems in fault-prone, asynchronous, distributed systems, applications of algebraic topology to asynchronous computability problems, simulation techniques, and also on characterizing the *Approximate Agreement* task.

In 1985, a fundamental paper by Fischer, Lynch, and Paterson [17] demonstrated that traditional Turing computability theory is not sufficient for analyzing computability problems in asynchronous, distributed systems. In particular, it showed that the well-known *Consensus* task, in which each participating process has a private input value drawn from some set S, and every nonfaulty process must decide on the same output value equal to the input of some process, cannot be solved in a message passing system even if only one process may fail by halting. Later, it was also shown that the message passing and shared memory models are equivalent [4], so this result carries over to shared memory systems as well. This fundamental discovery led to the creation of a highly active research area, which is surveyed in a recent book by Lynch [29].

In 1988, Biran, Moran, and Zaks [7] provided a breakthrough result by introducing a graph-theoretic framework that allows a complete characterization of the types of tasks that can be solved in a message passing or shared memory system in the presence of a single failure. However, this framework proved hard to extend to more than one failure, and even the problem of characterizing the solvability of specific tasks such as *Renaming* [5] and *Set Agreement* [13] for any number of processes remains unsolved.

In 1993, three research teams working independently—Borowsky and Gafni [10], Saks and Zaharoglou [32], and Herlihy and Shavit [24]—derived impossibility results for solving the Set Agreement task in the read-write shared memory model. The paper of Borowsky and Gafni introduced a powerful new simulation technique for proving solvability and unsolvability results in asynchronous, distributed systems. The technique allows N-process protocols to be executed by fewer processes in a resilient way and has been proven correct by Borowsky, Gafni, Lynch, and Rajsbaum [9]. The paper by Saks and Zaharoglou [32] constructed an elegant topological structure that captures the knowledge of the processors of the state of the system, allowing them to prove the impossibility of wait-free k-set agreement using point-set topology. The proof exposes an interesting relation between set agreement and the Brouwer fixed point theorem for the k-dimensional ball.

The paper of Herlihy and Shavit [24, 26] introduced a new formalism based on tools from classical, algebraic topology for reasoning about computations in asynchronous, distributed systems in which any number of processes may fail. Their framework consisted of modeling tasks and protocols using algebraic structures called *simplicial complexes* and then applying standard homology theory to reason about them. Herlihy and Shavit extended this framework by providing the *asynchronous computability theorem*, which states a condition that is necessary and sufficient for a task to be solvable by a wait-free protocol in shared memory [26], and showed applications of this theorem to tasks such as Set Agreement and Renaming. Borowsky [8] generalized this solvability condition to a model consisting of regular shared memory augmented with set-consensus objects, under more general resiliency requirements.

In 1993, Chaudhuri, Herlihy, Lynch, and Tuttle [14] also used topological and geometric arguments to prove tight bounds on solving the Set Agreement problem in the *synchronous* message passing model where an arbitrary number of processes may fail.

In 1994, Herlihy and Rajsbaum derived further impossibility results for Set Agreement by applying classical homology theory [20]. Moreover, in a unifying paper in 1995, Herlihy and Rajsbaum provided a common, general framework for describing a wide collection of impossibility results by using chain maps and chain complexes [22]. At the same time, Attiya and Rajsbaum reproved several impossibility results using purely combinatorial tools [3].

In 1995, Gafni and Koutsopias presented a reduction from the classical contractibility problem of algebraic topology to show that it is undecidable whether a certain class of 3-process tasks is wait-free solvable in the shared memory model [18]. This work was then generalized by Herlihy and Rajsbaum to arbitrary numbers of processes and failures in a variety of computational models [21]. More recently, Havlicek showed that, while undecidability holds in the general case, the problem of solvability is in fact decidable for a relatively large class of tasks [19]. Another recent paper by Herlihy, Rajsbaum, and Tuttle [23] introduces the use of pseudospheres as a means for unifying the synchronous, semisynchronous, and asynchronous message passing computation models.

The immediate snapshot (IS) object was introduced by Borowsky and Gafni in 1993 [11]. It is the basic building block of the iterated immediate snapshot (IIS) model, first implicitly used by Herlihy and Shavit [24, 25] and more recently formulated as a computation model by Borowsky and Gafni [12] as part of their new, simplified proof of the asynchronous computability theorem of Herlihy and Shavit [26]. This work also shows that the IIS model is computationally equivalent to standard shared memory models by providing a wait-free implementation of IIS from shared memory, and vice versa. It is not clear, however, whether these implementations are optimal from a complexity-theoretic viewpoint.

The *Approximate Agreement* problem is a weakening of the *Consensus* problem in which each process has a real valued input, and in any execution, nonfaulty processes with inputs in a range $[\min\_input, \max\_input]$ (the range changes from one execution to the next based on the input set of participating processes) must agree on output values within that range that are at most $\epsilon > 0$ apart. The problem was first introduced in 1986 by Dolev, Lynch, Pinter, Stark, and Weihl [15] in a paper showing that this task can be solved in both the synchronous and asynchronous message passing models even when assuming a Byzantine failure model (in which processes may exhibit arbitrary, even malicious, behavior). The paper also provided matching upper and lower bounds for solving the task in these settings. These results were extended to various failure models by Fekete [16], who also showed optimality in terms of the number of rounds of communication used.

In 1994, Attiya, Lynch, and Shavit published a paper giving an $\Omega(\log N)$ step complexity lower bound, together with a matching $O(\log N)$ upper bound, for solving in a wait-free manner N-process Approximate Agreement in "normal" (synchronous and failure-free) executions using single-writer, multireader shared memory [6]. These results were part of a proof that, in certain settings, wait-free algorithms are inherently *slower* than non–wait-free algorithms.

This work was extended by Schenk [33], who showed matching upper and lower bounds for solving the task in the asynchronous single-writer, multireader shared memory model where the magnitudes of the inputs are bounded from above.

Finally, in 1994, Aspnes and Herlihy [2] showed a $\lceil \log_3 \frac{\max\_input - \min\_input}{\epsilon} \rceil$ lower bound, together with a $\lceil \log_2 \frac{\max\_input - \min\_input}{\epsilon} \rceil$ upper bound on the time complexity (the number of steps taken by a process) for solving Approximate Agreement using wait-free protocols in the asynchronous single-writer, multireader shared memory model.

1.2. The asynchronous complexity theorem. This paper introduces a new theorem that for the first time provides a topological characterization of complexity for asynchronous computation. We introduce the *nonuniform iterated immediate snapshot (NIIS) model*, a refinement of the IIS model that allows better modeling of complexity. Keeping in style with Herlihy and Shavit's topological computability framework [26], our theorem states that the worst case time complexity for solving a decision task in the NIIS model is equivalent to the minimal number of nonuniform chromatic subdivisions of the task's input complex necessary to allow a simplicial map from the subdivided input complex to the output complex. The theorem implies an algorithm if one is given a subdivision and a mapping.

The nonuniform chromatic subdivisions we introduce (see Figure 13) are a looser and more general form of standard chromatic subdivisions [26]. Unlike the iterated standard chromatic subdivisions used in the computability work of [26, 12], they allow individual simplexes in a complex to be subdivided a different number of times, while ensuring that the subdivision of the complex as a whole remains consistent. Nonuniformity is a necessary property when analyzing complexity since it allows the level of subdivision of input simplexes to differ from one simplex to the next. This allows one to model a world in which different numbers of steps are taken on different input sets. If one used only uniform subdivisions, one could talk only about the complexity of the most highly subdivided simplex. This would make the complexity theorem useless, since, for example, for the Approximate Agreement problem, Aspnes and Herlihy [2] show that for any k one can find a set of inputs that will require time k in the worst case.

The power of our theorem lies in its ability to allow one to reason about the complexity of problems in a purely geometric setting. As we show, the subdivisions of a complex are a clean and higher level way of thinking about the multitude of different length executions of a concurrent protocol. We found this geometric representation helpful and believe that it will prove to be an invaluable tool for designing and analyzing concurrent algorithms. For technical reasons, in order to avoid the need to deal with infinite size complexes, we restrict our problem space to decision tasks with finite (yet not necessarily bounded) input and output domains.

We provide an example application of Theorem 4.2. In section 5, we use our topological framework to show tight upper and lower bounds on the time to solve the Approximate Agreement problem in a wait-free manner in the NIIS model. We close the gap implied by the work of Aspnes and Herlihy [2], proving matching upper and lower bounds of $\lceil \log_d \frac{\max\_input - \min\_input}{\epsilon} \rceil$, where $d = 3$ for two processes and $d = 2$ for three or more.

Apart from the theorem itself, its proof provides two additional contributions to the asynchronous computability literature.

- The upper bound proof of Herlihy and Shavit's asynchronous computability theorem [26] and related papers by Borowsky and Gafni [11, 12] all rely on the fact that the standard chromatic subdivision [25, 26] is indeed a subdivision in the topological sense. We provide the first formal proof of this fact.
- In 1997, Borowsky and Gafni provided a simulation of atomic snapshot mem-

ory from IIS memory [12]. They showed that based on this simulation, if one is given a proof of an asynchronous computability theorem for the IIS model (which they called Proposition 3.1), it will imply one for the general read-write model. The hope was that the proof of their Proposition 3.1 would be constructive and therefore significantly simpler than the nonconstructive proof in [24, 26]. The proof of our asynchronous complexity theorem in section 4 provides a constructive proof of computability for the NIIS model, and since IIS is a subset of NIIS, it provides the first known proof of Proposition 3.1 of [12].

1.3. Organization. The paper is organized as follows. Section 2 provides a formal definition of decision tasks. It also contains a thorough description of our model of computation, together with the complexity measures we use for analyzing protocols in this model. Section 3 contains a collection of necessary definitions and results from algebraic topology, as well as a description of how we model decision tasks and NIIS protocols topologically. It also contains definitions of the standard chromatic subdivision and the nonuniform chromatic subdivision. Section 4 contains a statement and proof of our main theorem. Section 5 contains an application of our asynchronous complexity theorem to the Approximate Agreement task. Finally, section 6 summarizes our results and also gives some directions for further research.

2. Model. In order to develop a useful and applicable complexity theory for asynchronous, distributed computer systems, we need to define some reasonable model of such systems. This model must be detailed enough so as to accurately and faithfully capture the inherent complexity of solving tasks in real distributed systems yet be simple enough so as to easily lend itself to some practical form of complexity analysis. The model we consider in this paper consists of a class of one-shot distributed problems, called *decision tasks*, together with a novel model of computation, a type of shared memory called the *nonuniform iterated immediate snapshot (NIIS) model*. This section contains a detailed description of these fundamental concepts. It also contains the complexity measures that will be used to analyze the complexity of solving decision tasks in the NIIS model.

2.1. Informal synopsis. We begin with an informal synopsis of our model, which largely follows that of Herlihy and Shavit [24, 25, 26]. Some fixed number $N = n+1$ of sequential threads of control, called *processes*, communicate by asynchronously accessing shared memory in order to solve *decision tasks*. In such a task, each process starts with a private *input* value and halts with a private *output* value. For example, in the well-known *Binary Consensus* task, the processes have binary inputs and must agree on some process's input [17]. A *protocol* is a distributed program that solves a decision task in such a system. A protocol is *wait-free* if it guarantees that every nonfaulty process will halt in a finite number of steps, independent of the progress of the other processes. The *time complexity* of solving a decision task in this model on a given input set is the supremum of the number of accesses to shared memory made by any process on that input set.

2.2. Decision tasks. In this section, we define decision tasks more precisely. This class of tasks is intended to provide a simplified model of reactive systems, such as databases, file systems, or automated teller machines. An input value represents information entering the system from the surrounding environment, such as a character typed at a keyboard, a message from another computer, or a signal from a sensor. An output value models an effect on the outside world, such as an irrevocable

decision to commit a transaction, to dispense cash, or to launch a missile. Informally speaking, a decision task is a relation between vectors of input values and vectors of output values. We define this more precisely below.

Let D_I and D_O be two finite data types, possibly identical, called the *input data type* and the *output data type*, respectively. We first define the concept of an input vector.

DEFINITION 2.1. *An $n+1$-process* input vector $\vec{I}$ *is an $n+1$-dimensional vector, indexed by $\{0,\ldots,n\}$, each component of which is either an object of type D_I or the distinguished value $\perp$, with the additional requirement that at least one component of $\vec{I}$ must be different from $\perp$.*

The definition of output vectors is similar to that of input vectors.

DEFINITION 2.2. *An $n+1$-process* output vector $\vec{O}$ *is an $n+1$-dimensional vector, indexed by $\{0,\ldots,n\}$, each component of which is either an object of type D_O or the distinguished value $\perp$.*

When it is clear from the context, we omit mentioning the number of processes in specifying input and output vectors. We denote the ith component of an input vector $\vec{I}$ by $\vec{I}[i]$, and, similarly, we denote the ith component of an output vector $\vec{O}$ by $\vec{O}[i]$. In the remainder of the paper, unless stated otherwise, we will assume that i and j are index values in the set $\{0,\ldots,n\}$. These index values will be used both for specifying vector elements and also for indexing processes. We will use the terms "one-dimensional array" ("array" for short) and "vector" interchangeably.

We are often concerned with executions that are prefixes of a given execution.

DEFINITION 2.3. *Vector $\vec{U}$ is a* prefix *of $\vec{V}$ if, for $0 \le i \le n$, either $\vec{U}[i] = \vec{V}[i]$ or $\vec{U}[i] = \perp$.*

If a prefix has an entry distinct from $\perp$, then it agrees with the corresponding entry in the original.

DEFINITION 2.4. *A set V of vectors is* prefix-closed *if for all $\vec{V} \in V$, every prefix $\vec{U}$ of $\vec{V}$ is in V.*

In this paper, we will consider only sets of input and output vectors that are finite and prefix-closed.

DEFINITION 2.5. *An* input set *is a finite, prefix-closed set of input vectors. An* output set *is a finite, prefix-closed set of output vectors.*

Next, we define the notion of a task specification map, which maps each element of the input set to a subset of the output set. Our definition is similar to that of Havlicek [19].

DEFINITION 2.6. *Let I and O be input and output sets, respectively. A* task specification map *relating the two sets is a relation $\gamma \subseteq I \times O$ such that the following conditions hold:*

- *For all $\vec{I} \in I$, there exists a vector $\vec{O} \in O$ such that $(\vec{I}, \vec{O}) \in \gamma$.*
- *For all $(\vec{I}, \vec{O}) \in \gamma$, and for all i, $\vec{I}[i] = \perp$ if and only if $\vec{O}[i] = \perp$.*

Note that the above definition requires that in defining the task, every participating process must have a defined output. This is the specification of the task and does not model its solvability in any given computation model.

As a convenient notation, we denote the set of vectors $\vec{O}$ in O such that $(\vec{I}, \vec{O}) \in \gamma$ by $\gamma(\vec{I})$. For a given input vector $\vec{I}$, the set of vectors $\gamma(\vec{I})$ simply represents the set of legitimate output vectors for the set of inputs specified by $\vec{I}$. This set will generally contain more than one allowable output vector.

DEFINITION 2.7. *A* decision task *$\mathcal{D} = \langle I, O, \gamma \rangle$ is a tuple consisting of a set I of input vectors, a set O of output vectors, and a task specification map γ relating these two sets.*

We note that, by definition, decision tasks are inherently *one-shot* in the sense that all processes have a single input and must decide on a single output exactly once. Not all entries in a given input vector need contain an input value; some may contain the special value $\perp$, indicating that some processes do not receive an input value. We formalize this notion of participation in the definition below.

DEFINITION 2.8. *For any input vector $\vec{I}$, if the ith component is not $\perp$, then i* participates *in $\vec{I}$. Otherwise, we say that i does not participate in $\vec{I}$. Moreover, we define the* participating set *in $\vec{I}$ to be the set of participating indexes.*

As noted by Herlihy and Shavit [24, 25, 26], the reason for incorporating an explicit notion of participating indexes in our formalism for decision tasks is that it is convenient for capturing the intuitive notion of "order of actions in time" through the use of participating processes. For example, it allows us to distinguish between tasks such as *Unique-Id* and *Fetch-And-Increment*, which have the same sets of input and output vectors, have the same $\gamma(\mathcal{I})$ when all processes participate, but have quite different task specification maps when subsets of participating processes are taken into account.

$\vec{I}$	$\gamma(\vec{I})$
$(0,\perp,\perp)$	$(0,\perp,\perp),(1,\perp,\perp),(2,\perp,\perp)$
$(\perp,0,\perp)$	$(\perp,0,\perp),(\perp,1,\perp),(\perp,2,\perp)$
$(\perp,\perp,0)$	$(\perp,\perp,0),(\perp,\perp,1),(\perp,\perp,2)$
$(0,0,\perp)$	$(0,1,\perp),(1,0,\perp),(0,2,\perp),(0,2,\perp),(2,1,\perp),(1,2,\perp)$
$(0,\perp,0)$	$(0,\perp,1),(1,\perp,0),(0,\perp,2),(0,\perp,2),(2,\perp,1),(1,\perp,2)$
$(\perp,0,0)$	$(\perp,0,1),(\perp,1,0),(\perp,0,2),(\perp,0,2),(\perp,2,1),(\perp,1,2)$
$(0,0,0)$	$(0,1,2),(0,2,1),(1,0,2),(1,2,0),(2,0,1),(2,1,0)$

FIG. 1. *The Unique-Id task.*

$\vec{I}$	$\gamma(\vec{I})$
$(0,\perp,\perp)$	$(0,\perp,\perp)$
$(\perp,0,\perp)$	$(\perp,0,\perp)$
$(\perp,\perp,0)$	$(\perp,\perp,0)$
$(0,0,\perp)$	$(0,1,\perp),(1,0,\perp)$
$(0,\perp,0)$	$(0,\perp,1),(1,\perp,0)$
$(\perp,0,0)$	$(\perp,0,1),(\perp,1,0)$
$(0,0,0)$	$(0,1,2),(0,2,1),(1,0,2),(1,2,0),(2,0,1),(2,1,0)$

FIG. 2. *The Fetch-And-Increment task.*

EXAMPLE 2.9. *The $n+1$-process* Unique-Id *task is defined as follows: each participating process $i \in \{0,\ldots,n\}$ has an input $x_i = 0$ and chooses an output $y_i \in \{0,\ldots,n\}$ such that for any pair of processes $i \neq j$, $y_i \neq y_j$.*

EXAMPLE 2.10. *In the* Fetch-And-Increment *problem, each participating process $i \in \{0,\ldots,n\}$ has an input $x_i = 0$ and chooses a unique output $y_i \in \{0,\ldots,n\}$ such that* (1) *for some participating process i, $y_i = 0$, and* (2) *for $1 \leq k \leq n$, if $y_i = k$, then for some $j \neq i$, $y_j = k-1$.*

The tables in Figures 1 and 2, taken from [26], show the task specifications for *Unique-Id* and *Fetch-And-Increment* for three processes. Notice that *Unique-Id* allows identifiers to be assigned statically, while *Fetch-And-Increment* effectively requires

that they be assigned dynamically in increasing order. The first task has a trivial wait-free solution: statically preassign the values 0, 1, and 2 to the three processes. The second has no solution in read-write memory if one or more processes can fail.

2.3. Modeling objects, processes, and protocols. We formally model objects, processes, and protocols using a simplified form of the Input/Output (I/O) automaton formalism of Lynch and Tuttle [28]. An *I/O automaton* is a nondeterministic automaton with a finite or infinite set of *states*, a set of *input actions*, a set of *output actions*, and a transition relation given by a set of *steps*, each defining a state transition following a given action. An *execution* of an I/O automaton is an alternating sequence of states and enabled actions, starting from some initial state. An *execution fragment* is a subsequence of consecutive states and actions occurring in an execution. For simplicity we will use the term execution to mean either execution or execution fragment, the appropriate term being clear from the context. An automaton *history* is the subsequence of actions occurring in an execution. Automata can be composed by identifying input and output actions in the natural way (details can be found in [28]).

An *object* X is an automaton with input action $call(i, v, X, D)$ and output action $return(i, v, X, D)$, where i is a process id, v is a value, X an object, and D a data type. An action on object X by process i is said to occur on X's ith "port." A *process* i is an automaton with output actions $call(i, v, X, D)$, and $decide(i, v)$ and input actions $return(i, v, X, D)$ and $start(i, v)$. An *operation* is a *matching* pair of *call* and *return* actions, that is, having the same type, name, and process id. From here on we will abuse this notation for the sake of clarity by dropping unnecessary parameters and denoting others using subscripts.

A *protocol* $\mathcal{P} = \{0, \ldots, n; M\}$ is the automaton composed by identifying in the obvious way the actions for processes $0, \ldots, n$ and the memory M. A process i is said to *participate* in an execution of a protocol if the execution contains a $start(v)_i$ action. The set of participating processes is called the execution's *participating set*. Note that this definition of the participating set matches our earlier definition of a participating set of input vectors. To capture the notion that a process represents a single thread of control, a protocol execution is *well formed* if every process history (the projection of the history onto the actions of i) has a unique *start* action (generated externally to the protocol) which precedes any *call* or *return* actions, alternates matching *call* and *return* actions, and has at most one *decide* action. We restrict our attention to well-formed executions.

2.4. Solvability. We are interested in solvability in the face of arbitrary fail-stop failures [17] (such failures also model processes being arbitrarily delayed or halted). To capture the notion of processes having fail-stop failures, we add to the process automaton a unique $fail(i)$ event. A process's execution is thus a sequence of actions ending in either a *decide* or a *fail* action. If the execution ended in a *fail* action the process is said to be *faulty*. An execution is *t-faulty* if up to t processes become faulty.

DEFINITION 2.11. *A protocol* solves *a decision task in an execution if the following condition holds. Let* $\{i | i \in U\}$ *be the processes that have start actions, and let* $\{u_i | i \in U\}$ *be their arguments. Let* $\{j | j \in V\}$, $V \subseteq U$, *be the processes that execute decide actions, and let* $\{v_j | j \in V\}$ *be their output values. Let* $\vec{I}$ *be the input vector with* u_i *in component* i, *and* $\perp$ *elsewhere, and let* $\vec{O}$ *be the corresponding output vector for the* v_j. *We require that*

1. *no process takes an infinite number of steps without a decide or fail action, and*
2. $\vec{O}$ *is a prefix of some vector in* $\Delta(\vec{I})$.

Informally, the second condition implies that if a protocol solves a task in an execution, the outputs of the nonfaulty processes in any prefix of the execution are consistent with the allowable outputs of the possibly larger set of inputs to the execution as a whole. A protocol for N processes *wait-free solves* a decision task if it solves it in every t-faulty execution where $0 \leq t < N$. We will call such a protocol *wait-free* and henceforth use the term *solves* to mean *wait-free solves.*[1]

2.5. One-shot IS. Our memory model is based on Borowsky and Gafni's *IS object* [11], a model that has proven to be a useful building block for the construction and analysis of protocols in many asynchronous, distributed systems [11, 12, 24, 25, 26, 31].

Informally, an $n+1$-process IS object consists of a shared $n+1$-dimensional memory array and supports a single type of operation, called *writeread*. Each *writeread* operation writes a value to a single shared memory array cell, and returns a "snapshot" view of the entire array in the state immediately following the write—hence the name "immediate snapshot." A *writeread* operation by process i writes its value to the ith cell of the memory array.

Formally, we can specify IS objects as I/O automata [28]. Let D be any data type, and define $\vartheta(D)$ to be the data type $(D \cup \{\perp\})^{n+1}$, the set of all $n+1$-arrays each of whose cells contains either an element of D or $\perp$. We index the elements of $\vartheta(D)$ using the numbers in $\{0, \ldots, n\}$ and define $\vartheta^k(D)$ for $k \geq 2$ as $\vartheta(\vartheta^{k-1}(D))$, that is, a recursively growing vector of vectors. An IS automaton for $n+1$-processes and data type D called IS_x is defined as in Figure 3. We refer to such an object as an IS_D^{n+1} object. When the data type and number of processes are clear from the context, we usually omit the subscripts and superscripts above.

In Figure 3, the operation *writeread*$(v, S)_{i,x}$ by process i on IS_x writes the value v to the ith cell of *memory* and subsequently returns a snapshot S. The idea of the automaton specification is to capture the notion of an update of a memory array location followed immediately by a snapshot view of the entire array. Using a style similar to that of the atomic snapshot memory specification of [1], we record the history of invocations and responses using interface variables and allow the combined "write and snapshot" operation itself to occur via an internal automaton transition at some point between the invocation and associated response. Figure 4 shows a stylized diagram of the IS object IS_x.

For all i, the *inv_writeread*$(v)_{i,x}$ action simply writes the input value v to the ith cell of the *input_value* array of IS_x. This array provides temporary storage for inputs to the IS_D^{n+1} object. At the same time, the flag *"inv"* is written to the ith cell of the *interface* array, which indicates an input by process i. The *update*$(\mathcal{U})$ action is the internal transition that periodically copies a set of values corresponding to the indexes in $\mathcal{U}$ from the *input_value* array to the *memory* array. The set $\mathcal{U}$ must be a subset of the indexes i with the property that *interface*$[i] = inv$; in other words, these are operations that have been invoked by participating processes and have not yet been updated in memory. Additionally, a copy of the *memory* array is written to the ith cell of the *return_value* array for each $i \in \mathcal{U}$. This corresponds to an IS view being collected. Finally, the flag *"ret"* is written to the ith cell of the *interface* array for each $i \in \mathcal{U}$, indicating that a response value to the invocation by process i is available.

[1] Note that we use the standard notion of *wait-free* protocols [27] and not the more restrictive notion of *bounded wait-free* protocols [27] even though, given that we consider deterministic algorithms and that in our discussion the input space is finite, one could actually place a bound on the length of any wait-free execution.

IS_D^{n+1} *object named* IS_x

Signature
Inputs:
$inv\_writeread(v)_{i,x}$, $v \in D$, $i \in \{0, \ldots, n\}$.
Internals:
$update(\mathcal{U})$, $\mathcal{U} \subseteq \{0, \ldots, n\}$.
Outputs:
$ret\_writeread(S)_{i,x}$, $S \in \vartheta(D)$, $i \in \{0, \ldots, n\}$.
State
memory $\in \vartheta(D)$, initially $(\perp, \ldots, \perp)$.
input_value $\in \vartheta(D)$, initially $(\perp, \ldots, \perp)$.
return_value $\in \vartheta^2(D)$, initially $(\perp, \ldots, \perp)$.
interface $\in \{inv, ret, \perp\}^{n+1}$, initially $(\perp, \ldots, \perp)$.

Transitions

input: $inv\_writeread(v)_{i,x}$
Eff: $input\_value[i] := v$
$interface[i] := inv$

internal: $update(\mathcal{U})$
Pre: $\mathcal{U} \subseteq \{i \mid interface[i] = inv\}$
Eff: For all i in $\mathcal{U}$ do
$memory[i] := input\_value[i]$
For all i in $\mathcal{U}$ do
$return\_value[i] := memory$
For all i in $\mathcal{U}$ do
$interface[i] := ret$

output: $ret\_writeread(S)_{i,x}$
Pre: $interface[i] = ret$
$return\_value[i] = S$
Eff: $interface[i] := \perp$

FIG. 3. *I/O automaton for an* IS_D^{n+1} *object with name* IS_x.

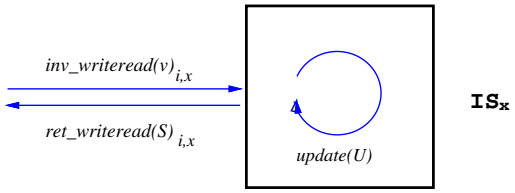


FIG. 4. *Diagram of* IS_x.

The $ret\_writeread(S)_{i,x}$ output action provides a response to a previous invocation by process i. Its only effect on the IS object is to reset the value $interface[i]$ to $\perp$, thereby preventing more than one response to an invocation. The $ret\_writeread(S)_{i,x}$ action can occur only after a return value has been written to the ith cell of the *return_value* array and the flag *"ret"* has been written to the corresponding cell in the *interface* array.

In the remainder of this section, we will state and prove a few basic properties about IS objects. These properties will be useful later, when we prove the correctness of a topological framework for analyzing the complexity of protocols in models of computation that include multiple IS objects.

In this paper we will consider only a restricted class of executions of IS_x, called *one-shot executions*, in which each object has at most one invocation and at most one response by any process.

LEMMA 2.12. *For any two distinct actions* $update(\mathcal{U})$ *and* $update(\mathcal{U}')$ *in a one-shot execution* α *of* IS_x, *the index sets* $\mathcal{U}$ *and* $\mathcal{U}'$ *are disjoint.*

Proof. Suppose without loss of generality that $\mathcal{U}$ occurs before $\mathcal{U}'$, and suppose $i \in \mathcal{U}$. Immediately after the action *update*($\mathcal{U}$), *interface[i]* is equal to *"ret."* Since we are considering a one-shot execution, *interface[i]* will not return to the value *"inv"* for the remainder of the execution. Hence, the precondition of the *update*($\mathcal{U}'$) action guarantees that $i \notin \mathcal{U}'$. □

We can now define what we mean by concurrent operations of IS_x.

DEFINITION 2.13. *Two operations writeread*$(v_i, S_i)_{i,x}$ *and writeread*$(v_j, S_j)_{j,x}$ *in a one-shot execution* α *of* IS_x *are concurrent if there exists an action update*($\mathcal{U}$) *in* α *such that* $i, j \in \mathcal{U}$.

The proof of the following lemma is immediate by construction.

LEMMA 2.14. *Consider any operation writeread*$(v_i, S_i)_{i,x}$ *in a one-shot execution* α *of* IS_x. *Then* $S_i[i] = v_i$.

The value returned by a *ret_writeread*$(S)_{i,x}$ action is a one-dimensional array of type $\vartheta(D)$. In the following lemma, we prove that IS_x exhibits the property that the set of snapshots returned in a one-shot execution can be totally ordered by the prefix relation defined in Definition 2.3.

LEMMA 2.15. *Consider any two writeread operations in a one-shot execution* α, *writeread*$(v_i, S_i)_{i,x}$ *and writeread*$(v_j, S_j)_{j,x}$. *Either* S_i *is a prefix of* S_j *or* S_j *is a prefix of* S_i.

Proof. Suppose the values v_i and v_j are written to *memory* by the actions *update*($\mathcal{U}_i$) and *update*($\mathcal{U}_j$), respectively.

If these actions are the same, that is, if $\mathcal{U}_i = \mathcal{U}_j$, the two operations *writeread*$(v_i, S_i)_{i,x}$ and *writeread*$(v_j, S_j)_{j,x}$ are concurrent. In this case, the value of *memory* that is copied to *return_value[i]* is identical to the value copied to *return_value[j]*, since both are copied by the same *update*($\mathcal{U}_i$) action. It follows that $S_i = S_j$. Now suppose $\mathcal{U}_i \neq \mathcal{U}_j$, and suppose *update*($\mathcal{U}_i$) occurs after *update*($\mathcal{U}_j$). Since no *memory* cells are ever reset, it follows that the *memory* version that is written to *return_value[j]* during *update*($\mathcal{U}_j$) is a prefix of the version that is written to *return_value[i]* during *update*($\mathcal{U}_i$). Hence S_j is a prefix of S_i. The case where *update*($\mathcal{U}_j$) occurs after *update*($\mathcal{U}_i$) is similar, and in this case we have that S_i is a prefix of S_j. The lemma follows. □

The next lemma concerns what is referred to in [11] as the *immediacy* property of *IS* objects. If a value written to *memory* by an invocation by process j is contained in a snapshot of an operation by process i, then the snapshot returned to process j is a prefix of that returned to process i. This corresponds to the informal notion of a *writeread* operation by j happening *before* a writeread operation by i.

LEMMA 2.16. *Consider any two writeread operations in a one-shot execution* α, *writeread*$(v_i, S_i)_{i,x}$ *and writeread*$(v_j, S_j)_{j,x}$. *If* $S_i[j] \neq \perp$, *then* S_j *is a prefix of* S_i.

Proof. Suppose the values v_i and v_j are written to *memory* by the actions *update*($\mathcal{U}_i$) and *update*($\mathcal{U}_j$), respectively, and suppose $S_i[j] \neq \perp$. This implies that either v_j was written to *memory* during *update*($\mathcal{U}_i$), in which case $\mathcal{U}_i = \mathcal{U}_j$, or the action *update*($\mathcal{U}_j$) occurred before *update*($\mathcal{U}_i$). In either case, we have that S_j must be a prefix of S_i. □

2.6. The NIIS model. Our *nonuniform iterated immediate snapshot (NIIS) model* is a variant of the iterated immediate snapshot (IIS) model, first used implicitly by Herlihy and Shavit [25, 26], and later formulated as a computation model by Borowsky and Gafni [12].

The IIS model assumes a bounded sequence $IS_D^{n+1}, IS_{\vartheta(D)}^{n+1}, IS_{\vartheta^2(D)}^{n+1}, \ldots, IS_{\vartheta^{k-1}(D)}^{n+1}$ of IS objects, denoted by $IS_1, IS_2, IS_3, \ldots, IS_k$, where $k > 0$. On a high level, the IIS

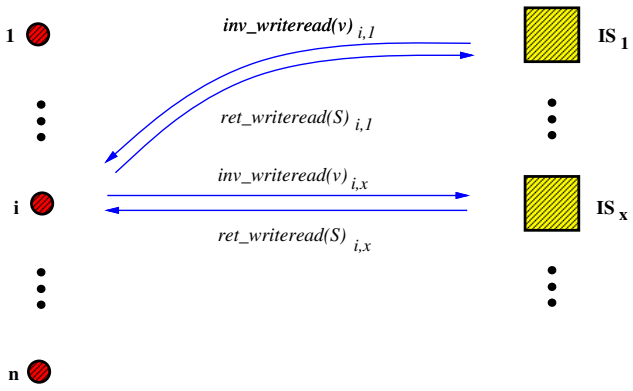


FIG. 5. *Diagram of the NIIS model.*

model has each participating process proceed in accessing IS objects in ascending order in the sequence. On each object it executes a *writeread(v)*$_{i,1}$ action, with v equal to its *local_state*. The response from each object, consisting of an IS of its shared memory vector, is provided as input to the next object accessed in the sequence, until k objects have been accessed. Once a process has received an output from the kth IS object, it applies a decision map δ to this value to create its returned decision value. We note that there is no loss of generality in assuming a "full information model" where the input to one *writeread* operation is the response by the previous one.

We generalize the IIS model by introducing the *NIIS model*. Unlike IIS, the NIIS model assumes an unbounded sequence $IS_D^{n+1}, IS_{\vartheta(D)}^{n+1}, IS_{\vartheta^2(D)}^{n+1}, \ldots$ of IS objects, denoted $IS_1, IS_2, IS_3, \ldots$. A stylized interconnection diagram of the k-shot IIS model is given in Figure 5. The number of IS objects accessed by any two distinct processes in a given execution need not be the same, and, moreover, the number of objects accessed by any fixed process may vary from execution to execution. The motivation behind this is to be able to model complexity more accurately. In a given execution it may be the case that the necessary amount of computation will vary from process to process, from input value to input value, and indeed from execution to execution. This cannot be captured by a uniform model such as IIS in which every execution of a given protocol will involve the same number of steps.

The only significant difference between a protocol $\mathcal{P}_{(n,\tau,\delta)}$ in the NIIS model and a protocol in the IIS model is that after each complete *writeread* operation, each process checks whether it has reached a final state by applying the predicate τ to the *local_state* variable. If τ returns `true`, the process executes a *decide(S)*$_i$ action and halts. Otherwise, it accesses the next IS object as in the IIS model, and so on. In fact, any protocol in the IIS model is equivalent to a protocol $\mathcal{P}_{(n,\tau,\delta)}$ in the NIIS model, in which the predicate τ simply checks whether the *local_state* variable is of type $\vartheta^k(D)$.

Notice that, unlike the IIS model, the NIIS model permits unbounded length executions (assuming an unbounded number of IS objects) for some choices of the termination predicate map τ. However, we will consider only protocols for which τ is chosen such that the entire system does not have any infinite executions.

Each protocol in the NIIS model is fully characterized by the maximum number $n+1$ of processes that can participate, a predicate function $\tau : \bigcup_{l=0}^{\infty} \vartheta^l(D) \to \{\texttt{true}, \texttt{false}\}$, which each process applies to its *local_state* variable after each com-

Process i

Signature
Inputs:
$ret\_writeread(S)_{i,x}$, $S \in \bigcup_{l=1}^{\infty} \vartheta^l(D)$.
$start(v)_i$, $v \in D$.
$fail_i$
Internals:

Outputs:
$inv\_writeread(v)_{i,x}$, $v \in \bigcup_{l=0}^{\infty} \vartheta^l(D)$.
$decide(v)_i$, $v \in D_O$.
State
local_state $\in (\bigcup_{l=0}^{\infty} \vartheta^l(D)) \cup \{\perp\}$, initially in the fresh state $\perp$.
status $\in \{ready, waiting, decided, failed\}$, initially *ready*.
Transitions

input: $ret\_writeread(S)_{i,x}$
Eff: $local\_state := S$
If $status = waiting$ do
$status := ready$

output: $inv\_writeread(v)_{i,x}$
Pre: $\tau(local\_state) =$ `false`
$status = ready$
$local\_state = v \neq \perp$
Eff: $status := waiting$

input: $fail_i$
Eff: $status := failed$

input: $start(v)_i$
Eff: $local\_state := v$

output: $decide(v)_i$
Pre: $\tau(local\_state) =$ `true`
$\delta(local\_state) = v \in D_O$
$status = ready$
Eff: $status := decided$

FIG. 6. *I/O automaton for process i running NIIS protocol.*

plete *writeread* operation to determine whether or not to decide, and a decision map $\delta : \bigcup_{l=0}^{\infty} \vartheta^l(D) \to D_O$, where D_O is an arbitrary data type, which we call the protocol's *output data type*. We refer to the protocol obtained by fixing these parameters as $\mathcal{P}_{(n,\tau,\delta)}$.

We specify each IS object as in Figure 3, and each process i as in Figure 6. The protocol can then be specified by composing the automata for the processes and the automata for IS objects by matching up invocations and responses from consecutive IS objects in the natural way. The resulting protocol automaton is denoted by $\mathcal{P}_{(n,\tau,\delta)} = \{0, 1, \ldots, n; IS_1, IS_2, \ldots\}$.

For any execution α of $\mathcal{P}_{(n,\tau,\delta)}$, the processes' input values can conveniently be represented using an $n+1$-dimensional *input vector* $\vec{I}$, as specified in the previous section, with input data type D. The ith entry of $\vec{I}$ is the input of process i. Similarly, the processes' output values in α can be represented using an $n+1$-dimensional *output vector* $\vec{O}$. The ith entry of $\vec{O}$ is the output of process i.

It should be noted that the notions of participating processes and sets defined for executions and input vectors are consistent; a process i participates in an execution α if and only if the index i participates in the input vector $\vec{I}$ corresponding to α. Therefore, when the meaning is clear from the context, we usually omit qualifying a participating set with an execution or input vector.

We note that we have added $fail_i$ actions to the protocol to achieve the stopping failure property. We note that by construction if α is an execution of $\mathcal{P}_{(n,\tau,\delta)}$ that contains a $fail_i$ action, then α contains no actions locally controlled by i ($inv\_writeread(v)_{i,x}$ or $decide(S)_i$) after the $fail_i$ action.

2.7. Complexity measures for the IIS and NIIS models. We now define the complexity measures to be used for analyzing the performance of protocols in the NIIS model. Since the IIS model is equivalent to a special case of the NIIS model, these measures also apply directly to the IIS model.

Let $\mathcal{P}_{(n,\tau,\delta)}$ be a protocol in the NIIS model solving a given decision task $\mathcal{D}$, let $\vec{I}$ be an input vector, and let α be any execution of $\mathcal{P}_{(n,\tau,\delta)}$ that corresponds to $\vec{I}$. For all i, let t_i be the number of IS objects accessed by process i in α. We first define the time complexity of the execution α.

DEFINITION 2.17. *The time complexity of α, denoted t_α, is $\max_i t_i$, the maximum number of IS objects accessed by any process.*

We note that t_α is well defined, since the number of processes $n+1$ is finite. Moreover, by definition of the max function, t_α is an integer value. We use the definition given above to define the time complexity of the protocol $\mathcal{P}$ on the input vector $\vec{I}$.

DEFINITION 2.18. *The time complexity of $\mathcal{P}_{(n,\tau,\delta)}$ on $\vec{I}$, denoted $t_{\vec{I}}$, is the supremum of the set $\{t_\alpha \mid t_\alpha$ is an execution corresponding to $\vec{I}\}$.*

Finally, we define the complexity of a protocol $\mathcal{P}_{(n,\tau,\delta)}$ on an input set I.

DEFINITION 2.19. *The time complexity of $\mathcal{P}_{(n,\tau,\delta)}$ on I, denoted t_I, is the supremum of the set $\{t_{\vec{I}} \mid \vec{I} \in I\}$.*

The reason for preferring these simple, discrete complexity measures over other, more elaborate, measures such as real time, for instance, is the highly regular structure of the IIS and NIIS models. We make the assumption that each access to an IS object takes the same amount of time and do not worry about breaking up the time required to complete each access to an IS object into subparts. Instead, we group the time spent on invocation, response, and on local computation at the IS object. This assumption is somewhat strong, as the presence of asynchrony in our model will tend to introduce varying delays for each access to an object. However, we believe that, as a first step toward a complexity theory, this assumption is justifiable, as it allows for complexity measures that are simple and easy to apply, and that have a particularly nice topological representation, as we will see in section 4.

3. A topological framework. In this section we first introduce some known tools from the field of algebraic topology and show how they may be used to model decision tasks and protocols in the NIIS model of computation. We then introduce a new tool for analyzing complexity in this setting, called the nonuniform iterated chromatic subdivision.

3.1. Basic topological definitions and concepts. This section introduces the basic topological definitions and concepts that we shall need for modeling decision tasks and wait-free protocols in the NIIS model. Some of these definitions are fairly standard and are mainly taken from popular textbooks on algebraic topology [30, 34], while others are due to Herlihy and Shavit [24, 25, 26]. The statements and proofs related to subdivisions are novel to this work. Some of the figures used in this section are also adopted from Herlihy and Shavit's work [24, 25, 26].

A *vertex* $\vec{v}$ is a point in a Euclidean space $\mathbb{R}^l$. A set $\{\vec{v}_0, \ldots, \vec{v}_n\}$ of vertexes is *geometrically independent* if and only if the set of vectors $\{\vec{v}_i - \vec{v}_0\}_{i=1}^n$ is linearly independent. Clearly, for a set of $n+1$ vertexes to be geometrically independent, $l \geq n$. We can now define the concept of a geometric simplex, or simplex for short.

DEFINITION 3.1. *Let $\{\vec{v}_0, \ldots, \vec{v}_n\}$ be a geometrically independent set of vertexes in $\mathbb{R}^l$. We define the n-simplex S spanned by $\vec{v}_0, \ldots, \vec{v}_n$ to be the set of all points x*

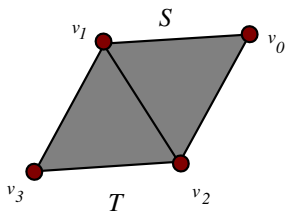


FIG. 7. *Example of a pure, two-dimensional simplicial complex.*

such that $x = \sum_{i=0}^{n} t_i \vec{v}_i$, *where* $\sum_{i=0}^{n} t_i = 1$ *and* $t_i \geq 0$ *for all* i.

For example, a 0-simplex is a vertex, a 1-simplex a line segment, a 2-simplex a solid triangle, and a 3-simplex a solid tetrahedron. For simplicity, we often denote the simplex spanned by a set $\{\vec{v}_0, \ldots, \vec{v}_n\}$ of geometrically independent vertexes as $(\vec{v}_0, \ldots, \vec{v}_n)$. The number n is called the *dimension* of the simplex S and is often denoted by $\dim(S)$. For clarity, we will sometimes include the number n as an explicit superscript when referring to a simplex; that is, we will write S^n to refer to the simplex spanned by the vertexes in $\{\vec{v}_0, \ldots, \vec{v}_n\}$.

Any simplex T spanned by a subset of $\{\vec{v}_0, \ldots, \vec{v}_n\}$ is called a *face* of S. The faces of S different from S itself are called the *proper* faces of S. The simplex spanned by the vertexes $\{\vec{v}_0, \vec{v}_1\}$ is a proper face of the 2-simplex S spanned by $\{\vec{v}_0, \vec{v}_1, \vec{v}_2\}$ in Figure 7.

The union of the proper faces of S is called the *boundary* of S and is denoted $Bd(S)$. The interior of S, denoted $\mathrm{Int}(S)$, is defined by the set equation $\mathrm{Int}(S) = S - Bd(S)$. For any set of points, the point

$$\vec{b} = \sum_{i=0}^{n} (\vec{v}_i/(n+1))$$

is their *barycenter*. The *barycenter* of a simplex S is the barycenter of its vertexes. In particular, if S is a vertex, then $\vec{b} = S$.

We will use a vertex to model the state of a single process, and a simplex to model consistent states of all the processes involved in solving a decision task or in running a protocol in the NIIS model. To model a collection of such states we need the concept of a geometric, simplicial complex, or complex for short, which is defined below.

DEFINITION 3.2. *A geometric simplicial complex* $\mathcal{K}$ *in the Euclidean space* $\mathbb{R}^l$ *is a collection of geometric simplexes in* $\mathbb{R}^l$ *such that*

- *every face* T *of every simplex* S *in* $\mathcal{K}$ *is contained in* $\mathcal{K}$, *and*
- *the intersection* U *of any two simplexes* S, T *in* K *is contained in* $\mathcal{K}$.

In this paper we will consider only finite complexes. The *dimension* of a complex $\mathcal{K}$, often denoted by $\dim(\mathcal{K})$, is the highest dimension of any of its simplexes and is also sometimes indicated explicitly by a superscript. An n-dimensional complex (or n-complex) is *pure* if every simplex is a face of some n-simplex. All complexes considered in this paper are pure unless stated otherwise. A simplex S in $\mathcal{K}$ with dimension $\dim(S) = \dim(\mathcal{K})$ is called a *maximal* simplex.

Given a simplex S, let $\mathcal{S}$ denote the complex of all faces of S, and let $\dot{\mathcal{S}}$ denote the complex consisting of all proper faces of S. We note that, since $\dot{\mathcal{S}}$ contains all faces

of S except S itself, $\dim(\dot{\mathcal{S}}) = \dim(S) - 1$. An example of a pure, two-dimensional simplicial complex, which we call $\mathcal{K}$, is shown in Figure 7. This complex equals the union of $\mathcal{S}$ and $\mathcal{T}$, where S is the 2-simplex spanned by $\{\vec{v}_0, \vec{v}_1, \vec{v}_2\}$, and T is the 2-simplex spanned by $\{\vec{v}_0, \vec{v}_1, \vec{v}_3\}$. Both S and T are maximal simplexes in this example.

If $\mathcal{L}$ is a subcollection of simplexes in $\mathcal{K}$ that is closed under containment and intersection, where $\dim(\mathcal{L}) \leq \dim(\mathcal{K})$, then $\mathcal{L}$ is a complex in its own right. It is called a *subcomplex* of $\mathcal{K}$. For example, the complex $\dot{\mathcal{S}}$ of faces of S is a subcomplex of $\mathcal{K}$ in Figure 7.

One subcomplex of a complex $\mathcal{K}$ of particular interest is the subcomplex of all simplexes in $\mathcal{K}$ of dimension at most p, where p is some integer between 0 and $\dim(\mathcal{K})$. We call this subcomplex the pth *skeleton* of a $\mathcal{K}$, denoted $skel^p(\mathcal{K})$. The elements of the collection $skel^0(\mathcal{K})$ are called the 0-*simplexes* of $\mathcal{K}$. The 0-skeleton of the complex $\mathcal{K}$ in Figure 7 is the collection of 0-simplexes $\{(\vec{v}_0), (\vec{v}_1), (\vec{v}_2), (\vec{v}_3)\}$. Similarly, the 1-skeleton of $\mathcal{K}$ is the union of the 0-skeleton described above and the collection $\{(\vec{v}_0, \vec{v}_1), (\vec{v}_0, \vec{v}_2), (\vec{v}_1, \vec{v}_2), (\vec{v}_1, \vec{v}_3), (\vec{v}_2, \vec{v}_3)\}$.

Let $|\mathcal{K}|$ be the subset $\bigcup_{S \in \mathcal{K}} S$ of $\mathbb{R}^l$ that is the union of the simplexes of $\mathcal{K}$. Giving each simplex its natural topology as a subspace of $\mathbb{R}^l$, we topologize $|\mathcal{K}|$ by declaring a subset A of $|\mathcal{K}|$ to be closed if and only if $A \cap S$ is closed for all $S \in \mathcal{K}$. This space is called the *polytope* of $\mathcal{K}$. Conversely, $\mathcal{K}$ is called a *triangulation* of $|\mathcal{K}|$.

In practice, the geometric representations we have given for simplexes and complexes are not always convenient, since the analytic geometry involved can get quite involved. Therefore, we introduce the notions of *abstract simplexes* and *abstract complexes*.

DEFINITION 3.3. *An abstract simplex S is a finite, nonempty set.*

The *dimension* of S is its cardinality. Each nonempty subset T of S is called a *face* of S. Each element of S is called a *vertex* of S. There is a close relationship between geometric simplexes and abstract simplexes. Any geometrically independent set of vectors $\{\vec{v}_0, \ldots, \vec{v}_n\}$ not only spans a geometric simplex; it also forms an abstract simplex.

DEFINITION 3.4. *An abstract complex $\mathcal{K}_a$ is a collection of abstract simplexes, such that if S is in $\mathcal{K}_a$, so is any face of S.*

Most concepts defined for geometric complexes immediately carry over to abstract complexes; the *dimension* of $\mathcal{K}_a$, often denoted by $\dim(\mathcal{K}_a)$, is the highest dimension of any of its simplexes. An n-dimensional abstract complex (or n-complex) is *pure* if every simplex is a face of some n-simplex. If $\mathcal{L}_a$ is a subcollection of $\mathcal{K}_a$ that is itself an abstract complex, then $\mathcal{L}_a$ is called a *subcomplex* of $\mathcal{K}_a$.

DEFINITION 3.5. *Let $\mathcal{K}$ be a geometric complex, and let V be the vertex set of $\mathcal{K}$. Let $\mathcal{K}_a$ be the abstract complex of all subsets S of V such that S spans a simplex in $\mathcal{K}$. Then $\mathcal{K}_a$ is called the* vertex scheme *of $\mathcal{K}$.*

DEFINITION 3.6. *Two abstract complexes $\mathcal{K}_a$ and $\mathcal{L}_a$ are* isomorphic *if there is a bijective correspondence ψ between their vertex sets such that a set S of vertexes is in $\mathcal{K}_a$ if and only if $\psi(S) \in \mathcal{L}_a$. The bijective correspondence ψ is called an* isomorphism.

THEOREM 3.7. *Every abstract complex $\mathcal{K}_a$ is isomorphic to the vertex scheme of some geometric complex $\mathcal{K}$ in $\mathbb{R}^{2\dim(\mathcal{K}_a)+1}$.*

We will not prove this theorem here. For a proof, see any standard textbook on algebraic topology [30, 34]. In the rest of this paper, for convenience, we will often use abstract and geometric representations of simplexes and complexes interchangeably.

We now define a way of "adding" simplexes, known as *starring*.

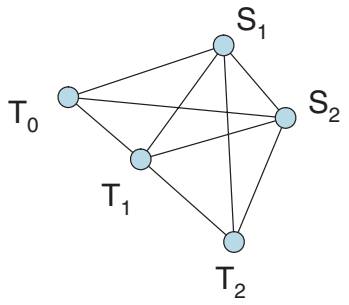


FIG. 8. *The star of two complexes S and T.*

DEFINITION 3.8. *Let $S = (s_0, \ldots, s_p)$ and $T = (t_0, \ldots, t_q)$ be simplexes whose combined sets of vertexes are affinely independent. Then the* star *of S and T, denoted $S \star T$, is the simplex $(s_0, \ldots, s_p, t_0, \ldots, t_q)$.*

We may extend the notion of starring to complexes as well.

DEFINITION 3.9. *Let $\mathcal{K}$ and $\mathcal{L}$ be simplicial complexes, not necessarily of the same dimension. Then the* star *of $\mathcal{K}$ and $\mathcal{L}$, denoted $\mathcal{K} \star \mathcal{L}$, is the collection of simplexes $\mathcal{K} \cup \mathcal{L} \cup \{S \star T \mid S \in \mathcal{K}, T \in \mathcal{L}\}$.*

The star of two complexes $\mathcal{K}$ and $\mathcal{L}$ is a complex in its own right [30]. Figure 8 shows a complex consisting of two 3-simplexes and all their faces resulting from starring the complex S which includes the 1-simplex (s_0, s_1) and all its faces, and the complex T consisting of the two 1-simplexes (t_0, t_1) and (t_1, t_2) and all their faces.

The remainder of this section, however, which introduces a number of important topological concepts, such as simplicial maps, subdivisions, and carriers, is set in the context of geometric complexes.

We first define the notions of simplicial vertex maps and simplicial maps from one complex into another.

DEFINITION 3.10. *Let $\mathcal{K}$ and $\mathcal{L}$ be complexes, possibly of different dimensions, and let $\mu : skel^0(\mathcal{K}) \to skel^0(\mathcal{L})$ be a function mapping vertexes to vertexes. Suppose that whenever the vertexes $\vec{v}_0, \ldots, \vec{v}_n$ of $\mathcal{K}$ span a simplex of $\mathcal{K}$, the vertexes $\mu(\vec{v}_0), \ldots, \mu(\vec{v}_n)$ span a simplex of $\mathcal{L}$. Then μ is called a simplicial vertex map from $\mathcal{K}$ to $\mathcal{L}$. μ can be extended to a continuous map $\mu_* : |\mathcal{K}| \to |\mathcal{L}|$ such that*

$$x = \sum_{i=0}^{n} t_i \vec{v}_i \Rightarrow \mu_*(x) = \sum_{i=0}^{n} t_i \mu_*(\vec{v}_i).$$

This continuous extension is called a simplicial map from $\mathcal{K}$ to $\mathcal{L}$.

For simplicity, we henceforth refer to the simplicial vertex map μ as the simplicial map, without actual reference to the continuous extension μ_*, which is less relevant for our purposes. As a further abuse of notation, we usually write $\mu : \mathcal{K} \to \mathcal{L}$ when we refer to the simplicial vertex map, glossing over the fact that this map is in fact defined only on the vertexes of $\mathcal{K}$, and that the image of the map is a subset of the vertex set of $\mathcal{L}$. Henceforth, unless stated otherwise, all maps between complexes are assumed to be simplicial. An example of a simplicial map is given in Figure 9.

We note that a simplex and its image under a simplicial map need not have the same dimension. A simplicial map $\mu : \mathcal{K} \to \mathcal{L}$ is *noncollapsing* if it preserves dimension; that is, for all $S \in \mathcal{K}$: $\dim(\mu(S)) = \dim(S)$.

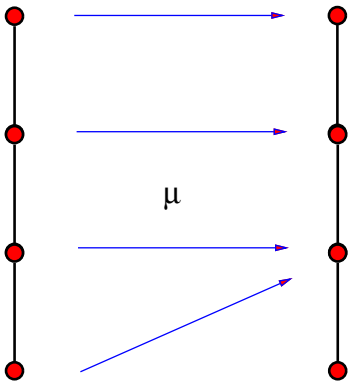


FIG. 9. *Example of a simplicial map between two complexes.*

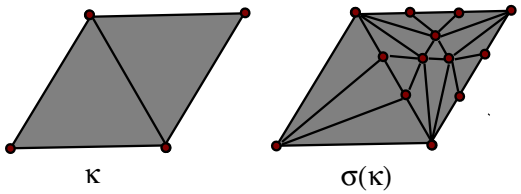


FIG. 10. *Example of a pure, two-dimensional simplicial complex and a subdivision of it.*

DEFINITION 3.11. *A* coloring *of an n-dimensional complex $\mathcal{K}$ is a noncollapsing simplicial map $\chi : \mathcal{K} \to \mathcal{S}$, where S is an n-simplex.*

Intuitively, a coloring corresponds to a labeling of the vertexes of the complex such that no two neighboring vertexes (connected by a 1-simplex) have the same label. A *chromatic complex* $(\mathcal{K}, \chi)$ is a complex $\mathcal{K}$ together with a coloring χ of $\mathcal{K}$. When it is clear from the context, we specify the chromatic complex $(\mathcal{K}, \chi)$ simply as the complex $\mathcal{K}$, omitting explicit mention of the coloring χ.

DEFINITION 3.12. *Let $(\mathcal{K}, \chi_{\mathcal{K}})$ and $(\mathcal{L}, \chi_{\mathcal{L}})$ be chromatic complexes, and let $\mu : \mathcal{K} \to \mathcal{L}$ be a simplicial map. We say that μ is* chromatic *if, for every vertex $\vec{v} \in \mathcal{K}$, $\chi_{\mathcal{K}}(\vec{v}) = \chi_{\mathcal{L}}(\mu(\vec{v}))$.*

In other words, μ is chromatic if it maps each vertex in $\mathcal{K}$ to a vertex in $\mathcal{L}$ of the same color. All the simplicial maps we consider in this paper are chromatic. We can now define the concepts of a *subdivision* of a complex and the *carrier* of a simplex in a subdivision.

DEFINITION 3.13. *Let $\mathcal{K}$ be a complex in $\mathbb{R}^l$. A complex $\sigma(\mathcal{K})$ is said to be a* subdivision *of $\mathcal{K}$ if the following two conditions hold:*

- *Each simplex in $\sigma(\mathcal{K})$ is contained in a simplex in $\mathcal{K}$.*
- *Each simplex of $\mathcal{K}$ equals the union of finitely many simplexes in $\sigma(\mathcal{K})$.*

An example of a complex and its subdivision is given in Figure 10.

DEFINITION 3.14. *If S is a simplex of $\sigma(\mathcal{K})$, the* carrier *of S, denoted carrier(S), is the unique smallest $T \in \mathcal{K}$ such that $S \subset T$.*

The concept of a carrier of a simplex is illustrated in Figure 11. The original

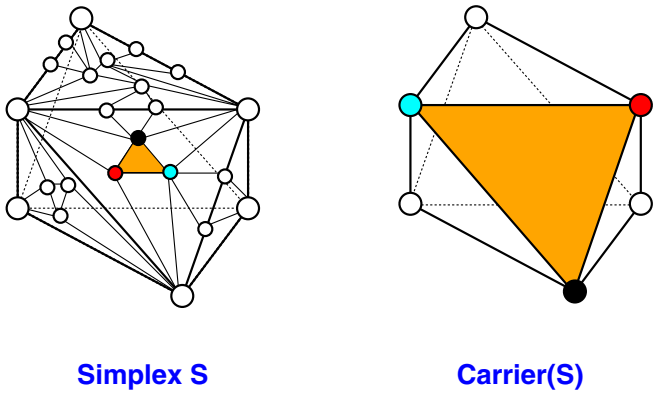


FIG. 11. *The carrier of a simplex.*

complex is shown on the right, and the subdivided complex is shown on the left. A simplex S in the subdivision and the corresponding carrier $carrier(S)$ in the original complex are highlighted in the figure.

A *chromatic subdivision* of $(\mathcal{K}, \chi_{\mathcal{K}})$ is a chromatic complex $(\sigma(\mathcal{K}), \chi_{\sigma(\mathcal{K})})$ such that $\sigma(\mathcal{K})$ is a subdivision of $\mathcal{K}$, and for all S in $\sigma(\mathcal{K})$, $\chi_{\sigma(\mathcal{K})}(S) \subseteq \chi_{\mathcal{K}}(carrier(S))$. A simplicial map $\mu : \sigma_1(\mathcal{K}) \to \sigma_2(\mathcal{K})$ between chromatic subdivisions of $\mathcal{K}$ is *carrier preserving* if for all $S \in \sigma_1(\mathcal{K})$, $carrier(S) = carrier(\mu(S))$. All subdivisions we consider in this paper will be chromatic, unless explicitly stated otherwise.

3.2. Topological modeling of decision tasks. Earlier in this section, we defined the notion of a decision task in terms of input and output vectors. That definition was intended to help the reader understand what a decision task is, but it lacks the mathematical structure necessary to prove interesting results. We now reformulate this definition in terms of simplicial complexes. To illustrate our constructions, we will first explain on a high level how to topologically model tasks, specifically the well-known Unique-Id task of Example 2.9. We will then provide detailed topological definitions of decision tasks.

We represent all possible input vectors to a task as a simplicial complex. In the case of the *Unique-Id* task, there is a (unique) $n+1$-dimensional input vector $\vec{I} = [0, \ldots, 0]$ represented as a simplex S, with dimension $0 \leq \dim(S) \leq n$. The dimension of S equals the number of non-$\perp$ elements in the vector.

From here on, each vertex $\vec{v}$ in a simplex S will be labeled with a process id and an input value. We will use $ids(S)$ to denote a simplex S's set of process ids (similarly for a complex), and $vals(S)$ to denote the multiset of values in S (similarly for a complex). If $\vec{J}$ is a prefix of $\vec{I}$, then the simplex corresponding to $\vec{J}$ is a face of S. The set I of input vectors is thus modeled as a complex $\mathcal{I}$ of input simplexes, called the *input complex*. For the *Unique-Id* task each vertex in $\vec{I}$ is labeled $\langle i, v_i \rangle$, where $\vec{I}[i] = v_i = 0$.

Similarly, the set O of output vectors is modeled as a complex $\mathcal{O}$ of output simplexes, called the *output complex*. In the *Unique-Id* task we represent each $n+1$-dimensional output vector $\vec{O} = [x_1, \ldots, x_n]$, where for all i, j, either $x_i = \perp$ or $0 \leq x_i \leq n$ and $(x_i = x_j) \Rightarrow (x_i = \perp)$, as a simplex T, with dimension $0 \leq \dim(T) \leq n$. Each vertex $\vec{v}$ in T is labeled with a process id and an output value $\langle i, v_i \rangle$, where $\vec{O}[i] = v_i$. As before, if $\vec{P}$ is a prefix of $\vec{O}$, then the simplex corresponding to $\vec{P}$ is a face of T.

A *topological task specification map* Γ maps the input complex to the output complex in a way that captures the input/output vector relation γ of a given task. The *Unique-Id* task induces a topological task specification map Γ in the natural way, mapping each input simplex $S \in \mathcal{I}$ to a set $\Gamma(S)$ of output simplexes in $\mathcal{O}$, with the property that for all $T \in \Gamma(S)$, the set $vals(T)$ contains no non-$\perp$ duplicates.

We can now give an alternative, topological representation of the *Unique-Id* decision task by simply specifying it as a tuple $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$ consisting of an input complex $\mathcal{I}$, an output complex $\mathcal{O}$, and a topological task specification map Γ.

This topological representation gives an alternative interpretation of the notion of "similar" system states. The processes corresponding to vertexes on the common boundary of the two simplexes cannot distinguish between the two global output sets based on their own output values. Unlike graph-theoretic models (e.g., [7]), simplicial complexes capture in a natural way the notion of the *degree* of similarity between the two global output sets: it is the dimension of the intersection of the two 2-simplexes.

We now give a formal procedure for how to specify a given decision task $\mathcal{D} = \langle I, O, \gamma \rangle$ topologically. We first construct a representation using abstract simplexes and complexes. It then follows from Theorem 3.7 that there exists a representation using geometric simplexes and complexes for which the vertex scheme is isomorphic to the abstract representation. There are standard ways of constructing such geometric complexes [30], but we choose not to get into the details of these constructions in this paper.

DEFINITION 3.15. *Let $\vec{I} \in I$ be an input vector. The* input simplex *corresponding to $\vec{I}$, denoted $S(\vec{I})$, is the abstract simplex $(\langle i_0, v_{i_0} \rangle, \ldots, \langle i_m, v_{i_m} \rangle)$, where $v_{i_j} = \vec{I}[i_j]$, and where for all i_j, $i_0 \leq i_j \leq i_m$, $i_j \in \{0, \ldots, n\} \wedge \vec{I}[i_j] \neq \perp$, and for all i, $(\vec{I}[i] = \perp) \Rightarrow (i \notin \{i_0, \ldots, i_m\})$.*

DEFINITION 3.16. *Let $\vec{O} \in O$ be an output vector. The* output simplex *corresponding to $\vec{O}$, denoted $T(\vec{O})$, is the abstract simplex $(\langle i_0, v_{i_1} \rangle, \ldots, \langle i_m, v_{i_m} \rangle)$, where $v_{i_j} = \vec{O}[i_j]$, and where for all i_j, $i_0 \leq i_j \leq i_m$, $i_j \in \{0, \ldots, n\} \wedge \vec{O}[i_j] \neq \perp$, and for all i, $(\vec{O}[i] = \perp) \Rightarrow (i \notin \{i_0, \ldots, i_m\})$.*

In other words, the vertexes in an input/output simplex correspond exactly to the non-$\perp$ values of the input/output vectors. Having defined input and output simplexes, we can define input and output complexes.

DEFINITION 3.17. *The* input complex *corresponding to I, denoted $\mathcal{I}$, is the collection of input simplexes $S(\vec{I})$ corresponding to the input vectors of I.*

DEFINITION 3.18. *The* output complex *corresponding to O, denoted $\mathcal{O}$, is the collection of output simplexes $T(\vec{O})$ corresponding to the output vectors of O.*

Definitions 3.17 and 3.18 make sense topologically due to the following lemma which follows from the fact that the sets of input and output vectors we consider are prefix-closed (see Definition 2.5).

LEMMA 3.19. *Given a set I of input vectors (alternatively a set O of output vectors), the corresponding input complex $\mathcal{I}$ (output complex $\mathcal{O}$), as defined in Definition* 3.17*, is an abstract, chromatic complex.*

Given a pair of (abstract) input and output complexes, we may apply Theorem 3.7 to construct a corresponding pair of geometric chromatic input and output complexes by embedding the abstract complexes in $\mathbb{R}^{2n+1}$. As discussed in section 3.1, we will

thus work with both interchangeably in the remainder of this paper.

We now construct a topological equivalent of the task specification map $\gamma \subseteq I \times O$.

DEFINITION 3.20. *The* topological task specification map *corresponding to* γ*, denoted* $\Gamma \subseteq \mathcal{I} \times \mathcal{O}$*, is defined as follows:*

$$(S(\vec{I}), T(\vec{O})) \in \Gamma \iff (\vec{I}, \vec{O}) \in \gamma.$$

As a convenient notation, for all $S(\vec{I}) \in \mathcal{I}$, we denote the set of simplexes $T(\vec{O})$ in $\mathcal{O}$ such that $(S(\vec{I}), T(\vec{O})) \in \Gamma$ by $\Gamma(S(\vec{I}))$. Usually, we simply refer to a topological task specification map as a "task specification map." We now prove that task specifications are id-preserving; if a process i has an input value, it must also have an output value, and vice versa.

LEMMA 3.21. *For all* $S(\vec{I}) \in \mathcal{I}$*, and all* $T(\vec{O}) \in \Gamma(S(\vec{I}))$*,* $ids(T) = ids(S)$.

Proof. Let $S(\vec{I})$ be any simplex in $\mathcal{I}$, and let $T(\vec{O}) \in \Gamma(S(\vec{I}))$. Then $\vec{O} \in \gamma(\vec{I})$ by Definition 3.20. Suppose $i \notin ids(S(\vec{I}))$. Then $\vec{I}[i] = \perp$ by Definition 3.15, and hence by Definition 2.6, $\vec{O}[i] = \perp$. It follows from Definition 3.16 that $i \notin ids(T(\vec{O}))$. Now suppose $i \notin ids(T(\vec{O}))$. Then $\vec{O}[i] = \perp$ by Definition 3.16, and hence by Definition 2.6, $\vec{I}[i] = \perp$. It follows from Definition 3.15 that $i \notin ids(S(\vec{I}))$. ∎

A schematic illustration of a topological decision task specification is given in Figure 12.

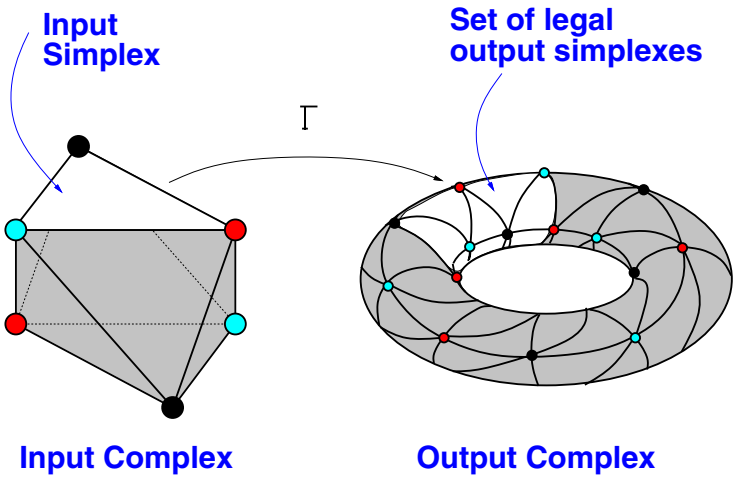


FIG. 12. *A decision task.*

DEFINITION 3.22. *Given a decision task* $\mathcal{D} = \langle I, O, \gamma \rangle$*, the corresponding* topological representation *of the task, denoted* $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$*, consists of an input complex* $\mathcal{I}$ *corresponding to* I*, an output complex* $\mathcal{O}$ *corresponding to* O*, and a task specification map* Γ *corresponding to* γ.

In the remainder of this paper, we will specify decision tasks using Definitions 2.7 and 3.22 interchangeably. A set of inputs or outputs may thus be specified as either a vector or a simplex, the vertexes of which are labeled with process ids and values.

3.3. Topological modeling of NIIS protocols. We model protocols in the NIIS model in much the same way that we model decision tasks. As discussed in section 2, the sets of inputs and outputs for any execution α of a protocol $\mathcal{P}_{(n,\tau,\delta)}$

in the NIIS model can be modeled using $n + 1$ process input and output vectors. We denote the sets of input vectors and output vectors of a protocol by I and O, respectively. We are interested only in protocols that solve decision tasks, so we may assume that the set I of possible input vectors to a protocol is prefix-closed. The following lemma states that for any protocol in the NIIS model, the set O of possible output vectors from all executions of the protocol must necessarily be prefix-closed. Recall from section 2.6 that we are considering only the set of fair (and hence finite) executions of a protocol here.

LEMMA 3.23. *Let O be the set of possible output vectors of a protocol $\mathcal{P}_{(n,\tau,\delta)}$ in the NIIS model, with a corresponding set of input vectors I. Then O is prefix-closed.*

Proof. Let $\vec{O}$ be an output vector produced by the execution α_O, and let $\vec{P}$ be a prefix of $\vec{O}$. We construct an execution α_P as follows: For each i such that $\vec{O}[i] = v_i \neq \perp = \vec{P}[i]$, replace the action *decide(S)*$_i$ (S is the output value returned by that action) in α_O with a *fail*$_i$ action, meaning that process i fail-stopped before deciding. Clearly, the execution thus obtained is a possible execution of $\mathcal{P}_{(n,\tau,\delta)}$, and its output vector is $\vec{P}$. Hence $\vec{P}$ is in O, and O is prefix-closed. □

Given that both the set of input vectors I and the set of output vectors O associated with a protocol $\mathcal{P}_{(n,\tau,\delta)}$ are prefix-closed sets of vectors, we can construct corresponding input and output complexes, denoted $\mathcal{I}$ and $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{I})$, respectively. These complexes are constructed in the same way as the complexes corresponding to input and output sets of vectors for decision tasks, and the proofs that they are indeed chromatic complexes are also identical. The output complex $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{I})$ is called a *protocol complex*.

Let $\mathcal{J}$ be a subcomplex of the input complex $\mathcal{I}$. The set of possible outputs when the protocol is given inputs corresponding to simplexes in $\mathcal{J}$ is denoted $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{J})$.

LEMMA 3.24. *Let $\mathcal{J}$ be a subcomplex of $\mathcal{I}$. Then $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{J})$ is a subcomplex of $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{I})$.*

Proof. It suffices to show that $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{J})$ is a complex, since $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{J})$ is clearly a subset of $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{I})$. Consider the set of vectors J corresponding to the subcomplex $\mathcal{J}$ as the set of input vectors to the protocol $\mathcal{P}_{(n,\tau,\delta)}$. This set is prefix-closed since $\mathcal{J}$ is a complex and hence is closed under containment. Hence the set P of output vectors given input vectors in J is prefix-closed by Lemma 3.23. It follows that the complex corresponding to $\mathcal{P}_{(n,\tau,\delta)}$ with input complex $\mathcal{J}$, denoted $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{J})$, is by construction a complex and hence a subcomplex of $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{I})$. □

In the remainder of this paper, we will specify protocols in NIIS using both its formal specification from section 2.6 as well as protocol complexes as described in this section interchangeably. A set of inputs or outputs may thus be specified as either a vector or a simplex, the vertexes of which are labeled with process ids and values.

3.4. Subdivisions. The standard chromatic subdivision was introduced by Herlihy and Shavit as part of their work on asynchronous computability [24, 25, 26]. It is essentially a chromatic generalization of the standard barycentric subdivision from classical algebraic topology [30, 34]. In this section, we will present a complete, formal definition of the standard chromatic subdivision, together with a proof that it is, in the topological sense, a chromatic subdivision of a given complex. As noted earlier, such a proof also provides the necessary formal basis for the use of the standard chromatic subdivision in [12, 26]. We note that our definition is somewhat different from that of Herlihy and Shavit [24, 25, 26], as it is based on an explicit, inductive, geometric construction. We also introduce the concept of a *nonuniform chromatic subdivision*, a generalization of the standard chromatic subdivision, in which the different simplexes

of a complex are not necessarily subdivided the same number of times. Informally, a nonuniform chromatic subdivision of level 1 of a complex $\mathcal{K}$, denoted by $\widetilde{\mathcal{X}}^1(\mathcal{K})$, is constructed by choosing, for each n-simplex in $\mathcal{K}$, a *single* face of the simplex (a face can be of any dimension and can also be the whole simplex) to which we apply the standard chromatic subdivision. We then induce the subdivision onto the rest of the simplex. The subdivisions of any two intersecting simplexes must be such that they agree on their shared face. Examples can be seen in Figure 13. Its right-hand side shows a valid nonuniform chromatic subdivision of a complex where, for example, the simplex (b, c, d)'s subdivision is the result of subdividing the 1-face (c, d) once and then inducing this subdivision onto the rest of the simplex. The left-hand side structure is not a legal subdivision, since the subdivision of the simplex (b, c, d) does not agree with that of the simplex (a, b, d) on the shared face (b, d). This structure is not even a simplicial complex, since it contains an object that is not a simplex (the cross-hatched region in Figure 13). A kth level nonuniform chromatic subdivision of a complex $\mathcal{K}$, denoted by $\widetilde{\mathcal{X}}^k(\mathcal{K})$, is generated by repeating this process k times, where only simplexes in faces that were subdivided in round $k-1$ can be subdivided in phase k. The complex on the right-hand side of Figure 13 is an example of a nonuniform chromatic subdivision of level 2, since the face (a, d) is subdivided twice.

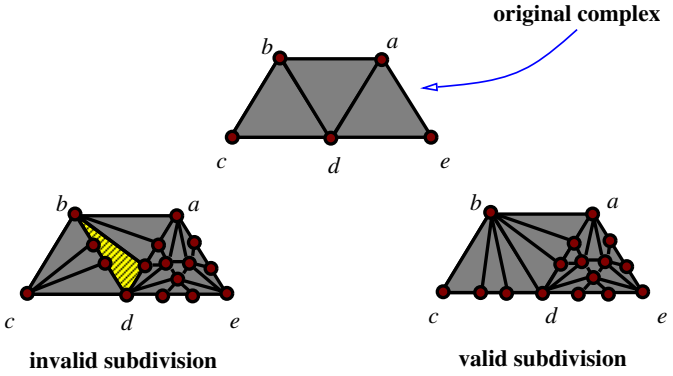


FIG. 13. *Valid and invalid nonuniform subdivisions.*

Later in this section we will show that the nonuniform chromatic subdivisions correspond in a natural way to the set of protocol complexes in the NIIS model of computation. As an execution in the NIIS model unfolds, some processes continue to step through IIS objects while others fail or decide. For a given input simplex, each transition through an IIS object by a subset of processes will correspond to a subdivision of the face corresponding to their respective vertexes. The other faces of the simplex, ones corresponding to the remaining processes, are not subdivided further, corresponding to the idea that the respective processes have either failed or decided. In the input complex, input simplexes sharing the same face have compatible subdivisions. As it turns out, each nonuniform standard chromatic subdivision is equal to some NIIS protocol complex (up to isomorphism).

3.4.1. The standard chromatic subdivision. In this section we provide our definition of the *standard chromatic subdivision* and prove that this definition does indeed specify a chromatic subdivision of a given complex.

Let $\mathcal{K}$ be a pure, n-dimensional, chromatic geometric complex, where the colors are the numbers in $0, \ldots, n$. Label each vertex $\vec{v}$ in $\mathcal{K}$ with $\langle i, v_i \rangle$, where i is the color

(process id) of $\vec{v}$ denoted also as $id(\vec{v})$, and v_i is a value (denoted also as $val(\vec{v})$) in some set D_I chosen such that no two vertexes in $\mathcal{K}$ have the same label. Note that two adjacent vertexes may have the same values if they have different colors. We define the standard chromatic subdivision of $\mathcal{K}$ by inductively defining a sequence of subdivisions $\mathcal{L}_p$ of the skeletons of $\mathcal{K}$, $0 \leq p \leq n$, as follows. We begin with the following auxiliary definition of the *containment conditions* among labeled vertexes.

DEFINITION 3.25. *For any set of simplexes $T = (\vec{t_0}, \ldots, \vec{t_r})$ in a complex $\mathcal{K}$ with labels $\langle i, S_i \rangle$, where S_i is the vertex scheme of some subset of simplexes in $\mathcal{K}$, define the* containment conditions *on the labels of T for any $1 \leq i, j \leq r$, $i \neq j$, as follows:*

C1. $id(\vec{t_i}) \neq id(\vec{t_j})$.

C2. $id(\vec{t_i}) \in ids(val(\vec{t_i}))$.

C3. $val(\vec{t_i})$ *is a face of* $val(\vec{t_j})$ *or vice versa.*

C4. $id(\vec{t_j}) \in ids(val(\vec{t_i})) \Rightarrow val(\vec{t_j})$ *is a face of* $val(\vec{t_i})$.

Condition C1 simply states that the given simplex is chromatic; that is, vertexes are colored with different ids. The remaining three conditions, which we will elaborate on shortly, will be used to capture the relation among the values written and read by a collection of *writeread* operations. In a nutshell, if one thinks of the $id(\vec{t_i})$ as the value written and the $ids(val(\vec{t_i}))$ as the IS value returned, then conditions C2, C3, and C4 correspond to the properties in Lemmas 2.14, 2.15, and 2.16.

We inductively define $\mathcal{L}_p$. Let $\mathcal{L}_0 = skel^0(\mathcal{K})$. Inductively assume that $\mathcal{L}_{p-1}$ is a chromatic subdivision of the $p-1$-skeleton of $\mathcal{K}$ where each vertex $\vec{v}$ in $\mathcal{L}_{p-1}$ is labeled $\langle i, S_i \rangle$, and where S_i is the vertex scheme of some simplex in $skel^{p-1}(\mathcal{K})$. We further assume that the labels $\langle i, S_i \rangle$ are such that any $T = (\vec{t_0}, \ldots, \vec{t_r})$, where $r \leq p-1$, is a simplex in $\mathcal{L}_{p-1}$ if and only if $ids(T) \subseteq ids(carrier(T))$ and for all $1 \leq i, j \leq r$, $i \neq j$, the labels of T meet the containment conditions of Definition 3.25.

Figure 14 describes a simplex (S_0, S_1, S_2) whose $\mathcal{L}_0$ subdivision includes the black vertexes. It has been subdivided by $\mathcal{L}_1$, causing each simplex in $\mathcal{L}_0$ to be split in three by the two new vertexes in grey. Note that each of these pairs of vertexes has a different *id* but the same value field which represents the vertex scheme of its carrier $\mathcal{L}_0$ simplex. The reader can check that the labels of these vertexes meet the four containment conditions of Definition 3.25. If we think of the value fields of vertexes as representations of the return value of an IS *writeread* operation, then the four conditions capture the nature of two process executions in the IIS model. For any two processes, say, 0 and 1, there are three possible outcomes of passing through an IIS object, represented by the three simplexes of the subdivided $(\langle 0, 0 \rangle, \langle 1, 1 \rangle)$ simplex: 0 reads only itself and 1 reads both, 1 reads only itself and 0 reads both, or they both read each other.

Based on $\mathcal{L}_{p-1}$ we can now complete the definition of $\mathcal{L}_p$. Let $S = (\vec{s_0}, \ldots, \vec{s_p})$ be a p-simplex in $\mathcal{K}$. The set $Bd(S)$ is the polytope of a subcomplex of the $p-1$-skeleton of $\mathcal{K}$, and hence of a subcomplex of $\mathcal{L}_{p-1}$, which we denote $\mathcal{L}_{Bd(S)}$. Let $\vec{b}$ be the barycenter of S, and let δ be some positive real number such that $0 < \delta < 1/p$. For each $1 \leq i \leq p$, define $\vec{m_i}$ to be the point $(1+\delta)\vec{b} - \delta\vec{s_i}$. These points are called the *midpoints* of S. Figure 14 shows the barycenter and midpoints of a 2-simplex. We can now label $\vec{m_i}$ with $\langle i, S \rangle$, S here being the vertex scheme of the geometric simplex S. Let M_S be the set of midpoints of S. We define $\mathcal{L}_S$ to be the union of $\mathcal{L}_{Bd(S)}$ and all the faces of all chromatic p-simplexes $T = (\vec{t_0}, \ldots, \vec{t_p})$, such that for all $1 \leq i, j \leq p : i \neq j$, $\vec{t_i} \in skel^0(\mathcal{L}_{Bd(S)}) \cup M_S$, and the four containment conditions of Definition 3.25 hold. $\mathcal{L}_p$ is thus the complex consisting of the union of the complexes $\mathcal{L}_S$, as S ranges over all the p-simplexes of $\mathcal{K}$.

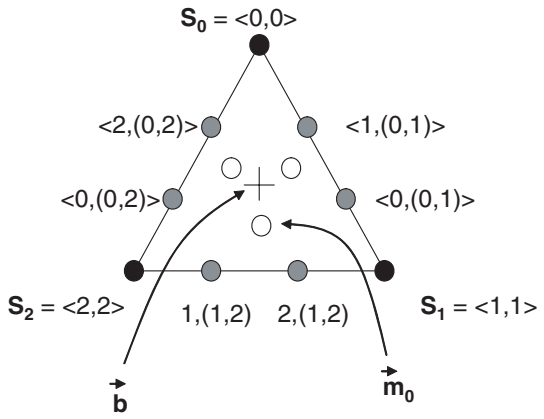


FIG. 14. *Example of the inductive step in the construction of the standard chromatic subdivision (for the sake of brevity, labels of simplexes slightly abuse notation).*

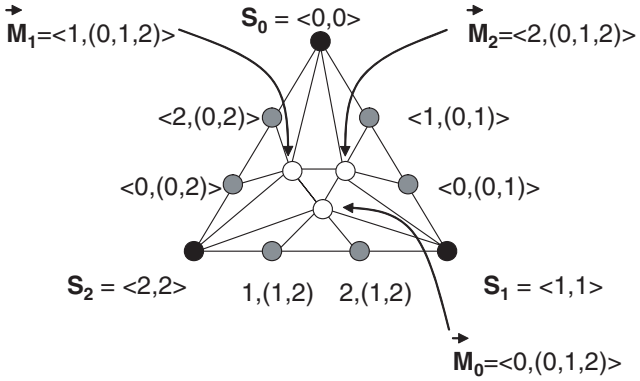


FIG. 15. *Example of the standard chromatic subdivision of a* 2*-simplex.*

Figure 15 describes a single subdivision $\mathcal{L}_p$ of a 2-simplex. Think of the value fields of vertexes as representations of the return values of an IS *writeread* operation by any one of three processes 1, 2, or 3. The complex captures all the possible outputs of executions in the IIS model. Each 2-simplex represents one such execution. A simplex with a face belonging to the boundary, such as the simplex $(\langle 1,1\rangle, \langle 2,(1,2)\rangle, \langle 0,(0,1,2)\rangle)$ in the lower right-hand corner of Figure 15, represents an execution where process 1 executed a *writeread* reading of only itself, process 2 performed a *writeread* and read itself, and process 1, and finally process 0 performed a *writeread* and read all three. The middle simplex (m_0, m_1, m_2) corresponds to an execution where all three processes were concurrent and read each other's written value. The three containment conditions C2, C3, and C4 on the labels of the vertexes guarantee the properties of the IIS model as they are captured in Lemmas 2.14, 2.15, and 2.16: *writeread* operations are by different processes, $writeread_i$ returns i's written value, the returned snapshot by one process must include the values returned by the other, and finally, if i read j, then the value returned by j cannot contain an input that i did not read.

We now prove that this structure makes sense mathematically, that is, that it is in fact a subdivision of the p-skeleton of $\mathcal{K}$.

LEMMA 3.26. *For all $0 \leq p \leq n$, $\mathcal{L}_p$ is a chromatic subdivision of $skel^p(\mathcal{K})$.*

Proof. We argue by induction. The case $p = 0$ is trivial. So suppose $p > 0$, and suppose the claim holds for $\mathcal{L}_0, \ldots, \mathcal{L}_{p-1}$. We will first prove that $\mathcal{L}_p$ is a chromatic simplicial complex. To that end, we prove the following auxiliary lemma.

LEMMA 3.27. *For all p-simplexes S in $\mathcal{K}$, $\mathcal{L}_S$ is a chromatic complex.*

Proof. We must show that $\mathcal{L}_S$ is closed under containment and intersection. Let U be a simplex in $\mathcal{L}_S$, and let V be a face of U, where $0 \leq \dim(V) \leq \dim(U) < p$. If U is in $\mathcal{L}_{Bd(S)}$, then so is V, since $\mathcal{L}_{Bd(S)}$ is a complex (since $\mathcal{L}_p$ is a subdivision and hence a complex by assumption). Hence V is in $\mathcal{L}_S$. Suppose U is not contained in $\mathcal{L}_{Bd(S)}$. Then U must be the face of a p-simplex T as described above. By definition of $\mathcal{L}_S$, all the faces of T, and hence all faces of U, must be in $\mathcal{L}_S$. It follows that $\mathcal{L}_S$ is closed under containment.

Let U, V be simplexes in $\mathcal{L}_S$, and suppose their intersection, denoted by W, is nonempty. If U, V are both in $\mathcal{L}_{Bd(S)}$, it follows immediately that V^r is in $\mathcal{L}_{Bd(S)}$ and hence in L_S. Similarly, if U is in $\mathcal{L}_{Bd(S)}$ but V is not, then $W = U \cap V = U \cap (V \cap |\mathcal{L}_{Bd(S)}|)$. Note that $V \cap |\mathcal{L}_{Bd(S)}|$ is a simplex in $\mathcal{L}_{Bd(S)}$, since all the containment conditions of Definition 3.25 are satisfied. Hence it follows that W is in $\mathcal{L}_{Bd(S)}$, and hence in $\mathcal{L}_S$. If neither U nor V is in $\mathcal{L}_{Bd(S)}$, then since all faces of U and V are in $\mathcal{L}_S$, then so is W. It follows that $\mathcal{L}_S$ is closed under intersection, and hence is a complex. That $\mathcal{L}_S$ is chromatic follows from the fact that we include only chromatic simplexes in $\mathcal{L}_S$ in our construction (note that $\mathcal{L}_{p-1}$ and hence $\mathcal{L}_{Bd(S)}$ are chromatic by assumption). □

Notice that for all distinct p-simplexes S, T we have that $|\mathcal{L}_S| \cap |\mathcal{L}_T| = S \cap T$, which is a simplex in $skel^{p-1}(\mathcal{K})$ and hence is the polytope of a subcomplex of $\mathcal{L}_{p-1}$ and hence of both $\mathcal{L}_S$ and $\mathcal{L}_T$. It follows that $\mathcal{L}_p$ is a simplicial complex [30]. It remains to show that $\mathcal{L}_p$ is a chromatic subdivision. To this end, we must first show that every simplex in $\mathcal{L}_p$ is contained in some simplex in $skel^p(\mathcal{K})$ and that every simplex in $skel^p(\mathcal{K})$ is the union of finitely many simplexes in $\mathcal{L}_p$. Now, it is clear from our construction that any simplex T_q in $\mathcal{L}_p$ is contained in some simplex S in $skel^p(\mathcal{K})$. Also, since for all simplexes S in $skel^p(\mathcal{K})$, the set of midpoints is finite, and $\mathcal{L}_{p-1}$ is a subdivision of $skel^{p-1}(\mathcal{K})$ by assumption, it follows that S is the union of finitely many simplexes in $\mathcal{L}_p$. Hence $\mathcal{L}_p$ is a subdivision. This subdivision is chromatic, since $\mathcal{L}_{p-1}$ is chromatic by assumption, since the colors used to color the midpoints of any simplex S are exactly the colors used to color S, and since any simplex in $\mathcal{L}_p$ including midpoints must satisfy the requirement that no two vertexes have the same color (id). □

We are now ready to give our definition of the standard chromatic subdivision of a complex $\mathcal{K}$.

DEFINITION 3.28. *The* standard chromatic subdivision *of $\mathcal{K}$, denoted $\mathcal{X}(\mathcal{K})$, is the complex $\mathcal{L}_n$.*

An example of a complex and its standard chromatic subdivision is given in Figure 16. As one can see, the subdivision of a simplex as seen in Figure 15 is applied to all the simplexes in the complex $\mathcal{K}$. Applying the standard chromatic subdivision k times, where $k > 1$, yields a subdivision $\mathcal{X}^k(\mathcal{K}) = \mathcal{X}^{k-1}(\mathcal{X}(\mathcal{K}))$, which we call the kth iterated standard chromatic subdivision [24, 25, 26]. Since the standard chromatic subdivision of a complex is again a complex, and a chromatic subdivision of a chromatic subdivision of $\mathcal{K}$ is itself a chromatic subdivision of $\mathcal{K}$, $\mathcal{X}^k(\mathcal{K})$ is a chromatic subdivision of $\mathcal{K}$. The number k is called the *level* of the subdivision.

The following is the vertex scheme representation of the standard chromatic subdivision. This particularly compact formulation of the standard chromatic subdivision

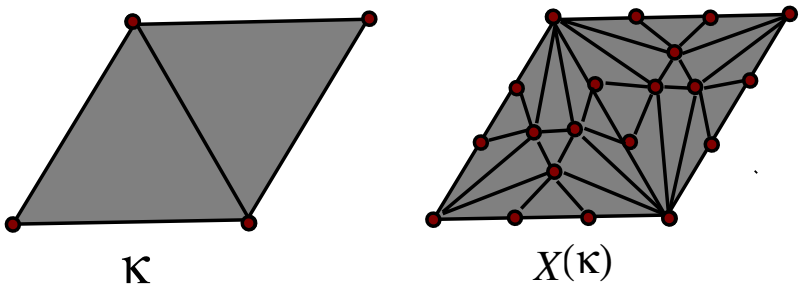


FIG. 16. *Example of a two-dimensional complex and its standard chromatic subdivision.*

is equivalent to the definition of Herlihy and Shavit [24, 25, 26].

LEMMA 3.29. *Let $\mathcal{K}$ be a pure, chromatic complex of dimension n. The vertex scheme of $\mathcal{X}(\mathcal{K})$ is the closure under containment of the set of all n-simplexes of the form $S = (\langle 0, S_0\rangle, \ldots, \langle n, S_n\rangle)$, where for all i, S_i is the vertex scheme of some face of a simplex S in $\mathcal{K}$, and the following conditions hold for all $i \neq j$:*

- *$i \in ids(S_i)$.*
- *S_i is a face of S_j or vice versa.*
- *If $j \in ids(S_i)$, then S_j is a face of S_i.*

Furthermore, any abstract complex $\mathcal{L}$ with a vertex scheme meeting the above criteria has a realization as a geometric standard chromatic subdivision $\mathcal{X}(\mathcal{K}) = \mathcal{L}$ of the complex $\mathcal{K}$ induced by all vertexes $\langle i, S_i\rangle$ where $\dim(S_i) = 0$.

As before, the above three conditions have the exact same role as containment conditions C2, C3, and C4 of Definition 3.25.

Proof. We argue that the vertex scheme of $\mathcal{X}(\mathcal{K})$ meets the above properties by induction on k, where $0 \leq k \leq n$. It is immediate that the simplexes of $\mathcal{X}(\mathcal{K})$ lying in the subdivision $\mathcal{L}_0$ of $skel^0(\mathcal{K})$ are of this form (each such simplex is a vertex of $\mathcal{K}$ labeled with a process id and a value), and the three requirements of the lemma are all satisfied trivially.

Now suppose the claim holds for $0, \ldots, k-1$. Consider a simplex T lying in the subdivision $\mathcal{L}_k$ of $skel^k(\mathcal{K})$ and not in $\mathcal{L}_{k-1}$. Then $T = U \star V$, where U is a simplex in $\mathcal{L}_{k-1}$, and V is a simplex, each vertex of which is one of the midpoints in M_S, where $S = carrier(T)$. By assumption, V is nontrivial, meaning that there is at least one vertex in V. However, U may be trivial. For each vertex $\vec{v}$ in V, $val(\vec{v}) = S$, the vertex scheme of S. Hence, for i, j in $ids(V)$, since $ids(V) \subseteq ids(S)$, all the containment conditions of Definition 3.25 are met. For i, j in $ids(U)$, the conditions are satisfied by induction. Now suppose i is in $ids(U)$, while j is in $ids(V)$. Notice that $S_i = carrier(U)$, and $S_j = carrier(V) = S$.

The first condition follows by induction (for i) and since $ids(V) \subseteq ids(S) = vals(V)$ (for j). Since $carrier(U)$ is a face of S, it follows that S_i is a proper face of S, and hence of S_j, which equals the vertex scheme of S, and so the second condition is satisfied. It is clear that i is in $ids(S_j)$, since S_j equals the vertex scheme of S, and i is in $ids(carrier(U))$, which is a subset of $ids(S)$. That S_i is a face of S_j has already been established. It follows that, since $\mathcal{X}(\mathcal{K})$ is chromatic, $ids(U) \cap ids(V) = \emptyset$,

$ids(S_i) \subseteq ids(carrier(U))$, and j is not in $ids(carrier(U))$, j cannot be in $ids(S_i)$. It follows that the third condition is satisfied.

We now show that any abstract complex $\mathcal{L}$ with a vertex scheme meeting the above criteria has a realization as a geometric standard chromatic subdivision $\mathcal{X}(\mathcal{K}) = \mathcal{L}$ of the complex $\mathcal{K}$ induced by all vertexes with 0-dimensional labels, that is, $\langle i, S_i \rangle$ where $\dim(S_i) = 0$. We argue by induction on k, $0 \leq k \leq n$, the size of the set S_i in the label of any vertex $\langle i, S_i \rangle$ in $\mathcal{L}$. It is immediate that the vertexes with zero-dimensional labels meet the criteria since the first condition of Lemma 3.29 implies condition C2 and all other conditions of Definition 3.28 are satisfied trivially. These vertexes form $skel^0(\mathcal{K})$.

Now suppose the claim holds for $0, \ldots, k-1$. Consider the set of k-simplexes T_j, all having vertexes $\langle i, S_i \rangle$ out of the same subset of k ids in $\mathcal{L}$. By definition each simplex has at least one vertex whose label $\langle i, S_i \rangle$ has $\dim(S_i) = k$. By the induction hypothesis the realization of this set includes k complexes, each with a proper subset using $k-1$ of these ids, and each a geometric complex meeting the requirements of the geometric standard chromatic subdivision. These complexes which inductively form $skel^{k-1}$ meet each other at $k-2$-dimensional boundaries, and their union (topological sum [30]) is a complex that is a $k-1$-dimensional sphere (for example, three one-dimensional complexes, each a subdivision using two unique ids, form a one-dimensional sphere, i.e., a circle). Let $\vec{b}$ be the barycenter of this sphere, and let v_i, $0 \leq i \leq k$, be the set of vertexes in the sphere with zero-dimensional labels. Now, in the set of simplexes T_j there are by the definition of Lemma 3.29 k vertexes with k-dimensional labels. If we choose each of them as a midpoint $\vec{m_i}$ at some distance $(1+\delta)\vec{b} - \delta\vec{v_i}$, from the barycenter $\vec{b}$ where $0 \leq \delta \leq 1/k$, then the simplexes T_j defined by the conditions of Lemma 3.29 form the interior of a simplex bounded by the $k-1$-dimensional sphere and fit the conditions of Lemma 3.26, implying that the above realization of $\mathcal{L}$ is a geometric standard chromatic subdivision of the complex $\mathcal{K}$ induced by the vertexes with zero-dimensional labels in $\mathcal{L}$. □

In the remainder of this paper, we will usually work with this description of the standard chromatic subdivision, and we refer to it as $\mathcal{X}(\mathcal{K})$. Whenever the distinction between the geometric and abstract representations of $\mathcal{X}(\mathcal{K})$ is significant, it will be mentioned explicitly.

3.4.2. The nonuniform chromatic subdivision. In this section, we define the *nonuniform chromatic subdivision* and prove that this definition does indeed specify a chromatic subdivision of a given complex. We will give a recursive definition of the *nonuniform chromatic subdivision* which we denote as $\widetilde{\mathcal{X}}^k(\mathcal{K})$, $k \geq 0$. We note that unlike $\mathcal{X}^k(\mathcal{K})$, $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is a procedure and not a function, and so $\widetilde{\mathcal{X}}^k(\widetilde{\mathcal{X}}(\mathcal{K})) \neq \widetilde{\mathcal{X}}(\widetilde{\mathcal{X}}^k(\mathcal{K}))$.

DEFINITION 3.30. *Let $\mathcal{K}$ be a pure n-dimensional chromatic complex, where the colors are the numbers in $0, \ldots, n$. A k-level nonuniform chromatic subdivision of a complex $\mathcal{K}$ by $\widetilde{\mathcal{X}}^k(\mathcal{K})$ for $k \geq 0$ is defined as follows.*

If $\dim(\mathcal{K}) = 0$, then for all $k \geq 0$, $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is $\mathcal{K}$ itself. Now suppose $\dim(\mathcal{K}) > 0$. Then $\widetilde{\mathcal{X}}^0(\mathcal{K})$ is $\mathcal{K}$ itself. For $k > 0$, $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is given by the following procedure: Partition the vertexes of $\mathcal{K}$ into two disjoint sets, A and B, where $\mathcal{A}$ is nonempty. Let $\mathcal{A}$ and $\mathcal{B}$ be the pure subcomplexes of $\mathcal{K}$ induced (respectively) by the vertexes in A and the vertexes in B. The subdivision $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is the complex consisting of all simplexes in $\mathcal{B}$, all simplexes in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, and all simplexes of the form $S \star T$, where S is a simplex in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, T is a simplex in $\mathcal{B}$, and $carrier(S) \star T$ is a simplex in $\mathcal{K}$.

We note that in the above definition, when we say induced subcomplexes $\mathcal{A}$ and $\mathcal{B}$, we mean that a simplex S in $\mathcal{K}$ is in $\mathcal{A}$ if the vertexes spanning S are all in A, and it is in $\mathcal{B}$ if its spanning vertexes are all in $\mathcal{B}$. Since $\mathcal{K}$ is pure, so are $\mathcal{A}$ and $\mathcal{B}$.

Informally speaking, a nonuniform chromatic subdivision of level k is one in which there is some simplex in $\mathcal{K}$ which is subdivided k times but no simplex that is subdivided more than k times. Note that the k-level standard chromatic subdivision is a special case of the k-level nonuniform chromatic subdivision. Hence for all $k \geq 0$, there exists some nonuniform chromatic subdivision of level k.

Our definition of nonuniform chromatic subdivisions is designed to model protocol complexes of the NIIS model. The main difference between IIS and NIIS protocols is that in NIIS, some processes may decide after passing through fewer IS objects than others. This is captured by the recursive definition that splits vertexes into two groups A and B. At any level of the recursion, the vertexes in $\mathcal{A}$ can be thought of as corresponding to processes that continue computing given their current local state, while the vertexes in $\mathcal{B}$ correspond to processes that decide.

To better understand our construction, let us jump slightly ahead of ourselves and consider how the protocol complex of any NIIS protocol with input complex $\mathcal{I}$ in some subset of processes accesses one IS object and is captured by a nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^1(\mathcal{I})$ up to isomorphism.

Consider any vertex $\vec{v}$ in $\mathcal{I}$. It is labeled with $\langle i, v_i \rangle$, where i is a process id and v_i represents an input value to process i. According to the specification of NIIS protocols in section 2.6, process i will (provided it does not fail), upon having received the input v_i, execute either an *inv_writeread(v)*$_{i,1}$ action or a *decide(S)*$_i$ action, depending on whether τ(*local_state*) evaluates to true or not. In this way, the predicate map τ induces a partition of the vertexes of $\mathcal{I}$ into two disjoint sets A and B. We now construct complexes $\mathcal{A}$ and $\mathcal{B}$ as in Definition 3.30; that is, a simplex T in $\mathcal{I}$ is in $\mathcal{A}$ if and only if all its vertexes are in A, and it is in $\mathcal{B}$ if and only if all its vertexes are in B.

The vertexes in A correspond to processes that, based on their input values, execute an *inv_writeread(v)*$_{i,1}$ action with the object IS_1. The protocol complex on $\mathcal{A}$ equals $\mathcal{X}(\mathcal{A})$ up to isomorphism. In any execution α of the protocol, some of the participating, nonfailing processes decide on their input values (corresponding to vertexes in B), while some decide on the snapshots they receive from the object IS_1 (corresponding to vertexes in the protocol complex of $\mathcal{A}$). It follows that the protocol complex on $\mathcal{I}$ contains every simplex in $\mathcal{B}$, every simplex in $\mathcal{X}(\mathcal{A})$, and every simplex of the form $S \star T$, where S is in $\mathcal{X}(\mathcal{A})$, T is in $\mathcal{B}$, and $carrier(S) \star T$ is in $\mathcal{I}$.

Note that the structure of the recursion of $\widetilde{\mathcal{X}}^k$ is such that $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$ is applied and not $\mathcal{X}(\widetilde{\mathcal{X}}^{k-1}(\mathcal{A}))$. This guarantees that we model a situation in which the subset of processes with nodes A in $\mathcal{K}$ go through the first IS object, and only a subset of these can then go through the next IS object, and so on. It is never the case that a node corresponding to a process that has stopped passing through earlier IS objects is later subdivided.

An example of a level 1 nonuniform chromatic subdivision of a 2-complex $\mathcal{K}$ is given in Figure 17, and an example of a level 2 nonuniform chromatic subdivision of a slightly bigger 2-complex $\mathcal{L}$ is given in Figure 18. Note that in Figure 18 the complex $\mathcal{A}$ for the second level of recursion is isomorphic to the complex $\mathcal{A}$ for the first level of recursion in Figure 17.

An example of a chromatic subdivision that does *not* satisfy Definition 3.30 is given in Figure 19. It is not a nonuniform chromatic subdivision because the vertex

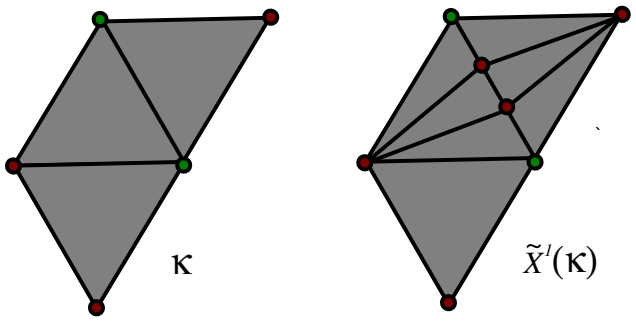


FIG. 17. *Example of level* 1 *nonuniform chromatic subdivision of a* 2*-complex.*

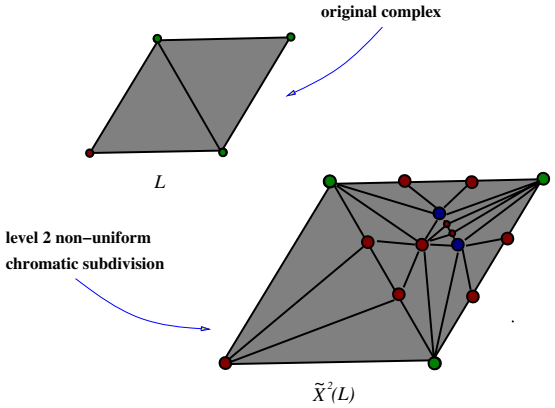


FIG. 18. *Example of level* 2 *nonuniform chromatic subdivision of a* 2*-complex.*

b is part of the $\mathcal{B}$ complex at the first level of recursion (that is, it is not part of the subcomplex that is subdivided further), while in the next level of recursion, the edge between d (which is in the $\mathcal{A}$ complex at the first level of recursion, and hence is to be subdivided further) and b is subdivided, meaning that b is in the $\mathcal{A}$ complex at the second level of recursion, which is clearly impossible, since the carrier of any vertex in the $\mathcal{A}$ complex at the second level must be a simplex in the $\mathcal{A}$ complex at the first level of recursion. Informally, this simply means that, if a vertex is not to be part of the complex to be further subdivided at the first level, it cannot be part of the complex to be further subdivided at the second level.

LEMMA 3.31. *Any nonuniform chromatic subdivision* $\widetilde{\mathcal{X}}^k(\mathcal{K})$ *is a chromatic subdivision of* $\mathcal{K}$*.*

Proof. We first note that $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is well defined, since each recursive step lowers the level of subdivision by 1, and $\widetilde{\mathcal{X}}^0(\mathcal{K})$ is defined for all $\mathcal{K}$. We will prove that $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is a chromatic subdivision by induction on k.

The case where $k = 0$ is trivial, since $\widetilde{\mathcal{X}}^0(\mathcal{K}) = \mathcal{K}$. Now suppose that $k > 0$, and that for $0 \leq l \leq k-1$, and any complex $\mathcal{K}$, $\widetilde{\mathcal{X}}^l(\mathcal{K})$ is a chromatic subdivision of $\mathcal{K}$.

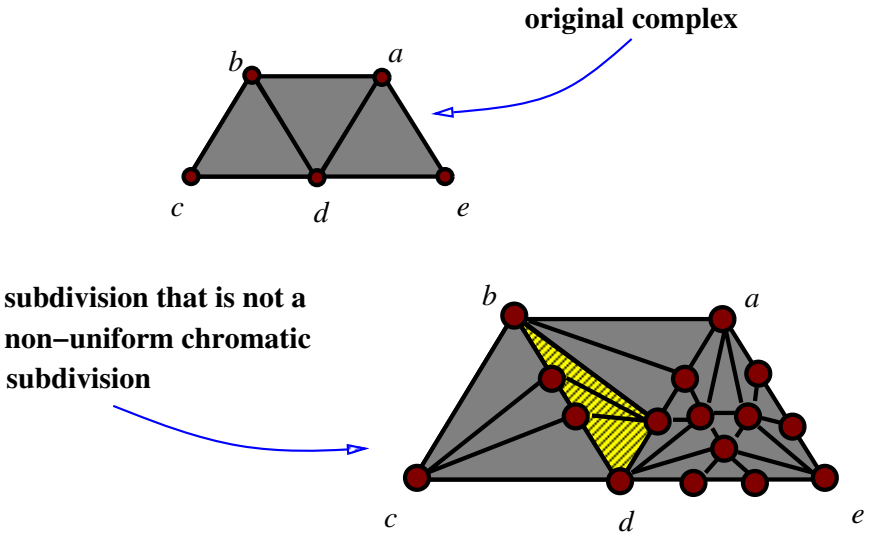


FIG. 19. *Example of a subdivision that is* not *a nonuniform chromatic subdivision.*

If $B = \emptyset$, the result follows by induction and by Lemma 3.26. So suppose that B is nonempty.

We first show that $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is closed under containment. Let U be a simplex in $\widetilde{\mathcal{X}}^k(\mathcal{K})$, and let V be a face of U. If U is in $\mathcal{B}$, then so is V, since $\mathcal{B}$ is a complex. Hence V is in $\widetilde{\mathcal{X}}^k(\mathcal{K})$. Similarly, if U is in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, then so is V, since $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$ is a complex by our induction hypothesis. Now suppose $U = S \star T$ for some S in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, T in $\mathcal{B}$. Then $S \cap V$ is in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, and $T \cap V$ is in $\mathcal{B}$. It follows that $V = (S \cap V) \star (T \cap V)$, where $carrier(S \cap V) \star (T \cap V)$ is a simplex in $\mathcal{K}$. By Definition 3.30, V is in $\widetilde{\mathcal{X}}^k(\mathcal{K})$. It follows that $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is closed under containment.

Let U, V be simplexes in $\widetilde{\mathcal{X}}^k(\mathcal{K})$, and let W be their intersection. If both U, V are in $\mathcal{B}$, then so is W, since $\mathcal{B}$ is closed under intersection. Similarly, if both U and V are in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, then so is W, since $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$ is closed under intersection. If U is in $\mathcal{B}$ and V is in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, or vice versa, then $U \cap V = \emptyset$, and so containment under intersection holds vacuously. We now consider the case where either U or V is not contained in either complex; that is, suppose $U = S \star T$ for some S in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, T in $\mathcal{B}$, and $V = X \star Y$ for some X in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, Y in $\mathcal{B}$. Now, $U \cap V = (S \star T) \cap (X \star Y)$, and $(S \star T) \cap (X \star Y) = (S \cap X) \star (T \cap Y)$ [30]. If $S \cap X = \emptyset$ or $T \cap Y = \emptyset$, then since the remaining nonempty intersecting pair of subsets is completely in $\mathcal{B}$ or completely in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, it follows that W is also. So suppose now that S, T, X, Y, and the intersections $S \cap X$ and $Y \cap T$ are all nonempty. Since $carrier(S \cap X)$ is a face of both $carrier(S)$ and $carrier(X)$, it follows that $carrier(S \cap X) \star T$ and $carrier(S \cap X) \star Y$ are simplexes in $\mathcal{K}$, and so is their intersection $carrier(S \cap X) \star (T \cap Y)$, since $\mathcal{K}$ is a complex. It follows that W is in $\widetilde{\mathcal{X}}^k(\mathcal{K})$. This concludes the proof that $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is a complex. That it is a chromatic complex follows directly from Lemma 3.26.

We now prove that $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is a chromatic subdivision. Given is a simplex U in $\widetilde{\mathcal{X}}^k(\mathcal{K})$. If U is in $\mathcal{B}$, then U is clearly contained in a simplex in $\mathcal{K}$, namely, itself, and the colors of U are contained in the set of colors of its carrier. If U is in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, it follows by induction and Lemma 3.26 that U is contained in some some simplex

carrier(U) in $\mathcal{A}$, and hence in $\mathcal{K}$, and that the colors of U are a subset of the colors of its carrier. Now suppose $U = S \star T$, where *carrier*$(S) \star T$ is in $\mathcal{K}$. Then U is contained in *carrier*$(S) \star T$, and the colors of U are a subset of the colors of *carrier*$(S) \star T$. Now consider any simplex U in $\mathcal{K}$. We can decompose it into two disjoint faces S and T, such that $S \in \mathcal{A}$ and $T \in \mathcal{B}$. The simplex S is subdivided according to $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, which by induction and Lemma 3.26 consists of finitely many simplexes. The simplex T is not subdivided at all. It follows that the subdivision $\widetilde{\mathcal{X}}^{k}(\mathcal{K})$ subdivides U into finitely many simplexes (those in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{S})) \star T$). This completes the proof that $\widetilde{\mathcal{X}}^{k}(\mathcal{K})$ is a chromatic subdivision. □

A nonuniform subdivision $\widetilde{\mathcal{X}}^{k}(\mathcal{K})$ of a complex induces a nonuniform chromatic subdivision of any subcomplex $\mathcal{L}$ of $\mathcal{K}$. The level of the induced subdivision of $\mathcal{L}$ may vary from subcomplex to subcomplex. We slightly abuse our notation to be able to define the effect of $\widetilde{\mathcal{X}}^{k}(\mathcal{K})$ on the restricted subcomplex $\mathcal{L}$.

DEFINITION 3.32. *Let $\mathcal{K}$ be a chromatic complex, let $\mathcal{L}$ be a subcomplex or simplex of $\mathcal{K}$, and let $\widetilde{\mathcal{X}}^{k}(\mathcal{K})$ be a nonuniform iterated chromatic subdivision of $\mathcal{K}$. We denote its restriction to simplexes in $\mathcal{L}$ by $\widetilde{\mathcal{X}}^{k}(\mathcal{L}/\mathcal{K})$. The* level *of subdivision $k_{\mathcal{L}}$ is the maximal level of subdivision of $\widetilde{\mathcal{X}}^{k}(\mathcal{L}/\mathcal{K})$.*

It is clear that for any subcomplex or simplex $\mathcal{L}$ of $\mathcal{K}$, $k_{\mathcal{L}} \leq k$.

4. The asynchronous complexity theorem. The strength and usefulness of the NIIS model of computation comes from the fact that each of its associated protocol complexes has a nice, recursive structure. In fact, it turns out that any protocol complex of NIIS is equal to some nonuniform iterated chromatic subdivision of the input complex, and vice versa. This is the essence of our main theorem, which we state and prove in this section.

The level of subdivision necessary for the existence of a simplicial map from the input to the output complex of a decision task that agrees with the task specification can be interpreted as a topological measure of the task's time complexity. The following definition introduces the concept of *mappability*, which is a useful construct for reasoning about this topological measure.

DEFINITION 4.1. *Given a decision task $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$ and a nonnegative integer k, we say that $\widetilde{\mathcal{X}}^{k}(\mathcal{I})$ is a* mappable *subdivision of the input complex and k is a* mappable *level of subdivision if there exists some chromatic simplicial map μ from $\widetilde{\mathcal{X}}^{k}(\mathcal{I})$ to $\mathcal{O}$ such that for all T in $\widetilde{\mathcal{X}}^{k}(\mathcal{I})$, $\mu(T) \in \Gamma($*carrier*$(T))$.*

This definition extends naturally to individual simplexes as the map induces different levels of subdivision on the individual simplexes in accordance with the idea that, in order to solve a decision task, some processes may have to do more computational work than others, and some inputs may require more computation than others. We can now state our main theorem.

THEOREM 4.2 (time complexity). *A decision task $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$ has a wait-free solution protocol in the NIIS model with worst case time complexity k_S on inputs $S \in \mathcal{I}$ if and only if there is a mappable nonuniform iterated chromatic subdivision $\widetilde{\mathcal{X}}^{k}(\mathcal{I})$ with level k_S on S.*

Keeping in style with Herlihy and Shavit [24, 25, 26], the theorem simply states that solvability of a decision task $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$ in the NIIS model is equivalent to the existence of a chromatic simplicial map μ from some nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^{k}(\mathcal{I})$ to $\mathcal{O}$ that agrees with the task specification Γ; that is, for all T in $\widetilde{\mathcal{X}}^{k}(\mathcal{I})$, $\mu(T^m) \in \Gamma($*carrier*$(T))$. The minimum possible level k_S is a lower bound on the worst case time complexity of solving this task with inputs in S in the NIIS model.

As noted in the introduction, the theorem directly implies Proposition 3.1 of [12]. In [12], Borowsky and Gafni provided a simulation of atomic snapshot memory from IIS memory and showed that, based on this simulation, if one is given a constructive proof of an asynchronous computability theorem for the IIS model (which they called Proposition 3.1), it will imply one for the general read-write model. The proof we are about to present provides a constructive proof of computability for the NIIS model, and since IIS is a subset of NIIS, it provides the first known proof of Proposition 3.1 of [12].

Our theorem immediately provides a solution algorithm for a task given the subdivision and simplicial mapping. Simply run the protocol of Figure 6. Since each process can locally store the subdivision and mapping, the termination predicate map τ just needs to test if the *local_state* variable is equal to some node v in the subdivision and if so return $\mu(v)$.

In the remainder of this section, we will give the proof of our asynchronous time complexity theorem. We begin by proving a lemma about the protocol complex of a protocol in the IIS model with only one available IS object.

LEMMA 4.3. *Let $\mathcal{A}$ be an input complex in the IIS model with a single IS object. The corresponding protocol complex is isomorphic to $\mathcal{X}(\mathcal{A})$.*

Proof. We will construct an isomorphism Ψ from the abstract complex $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{A})$ to the abstract complex (vertex scheme) $\mathcal{X}(A)$, as specified by Lemma 3.29. Let $\vec{v} = \langle i, S_i\rangle$ be any vertex in $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{A})$. Then $\Psi(\vec{v}) = \langle i, T_i\rangle$, where T_i is the simplex in $\mathcal{A}$ such that for all j, $S_i[j] = v_j$ if and only if $\langle j, v_j\rangle \in T_i$. Notice that this isomorphism is chromatic; that is, the id of a vertex equals the id of its image under Ψ.

By Lemma 3.29, we must show that a set of vertexes $\vec{v}_0, \ldots, \vec{v}_m$ in $skel^0(\mathcal{P}_{(n,\tau,\delta)}(\mathcal{A}))$, where $m \leq n$, forms a simplex in $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{A})$ if and only if the set of vertexes $\Psi(\vec{v}_0)$, $\ldots, \Psi(\vec{v}_m)$ in $skel^0(\mathcal{X}(\mathcal{A}))$ forms a simplex in $\mathcal{X}(\mathcal{A})$. Suppose without loss of generality that for all i, where $0 \leq i \leq m$, $\vec{v}_i = \langle i, S_i\rangle$, where $S_i \in \vartheta(D_I)$, and D_I is the input data type (that is, the id of the ith vertex is i).

Suppose that the vertexes $\vec{v}_0, \ldots, \vec{v}_m$ do form a simplex V in $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{A})$. This output simplex corresponds to some execution α in the one-shot IS model, with corresponding input simplex U in $\mathcal{A}$. Each vertex in U is labeled with a process id i and an input value $v_i \in D_I$. Notice that $\dim(V) \leq \dim(U)$, since some participating processes may not decide, that is, they may fail (execute a $fail_i$ action) before executing a *decide* action.

From Lemma 2.14, we have that, for any vertex $\vec{v}_i = \langle i, S_i\rangle$ in V, $S_i[i] = v_i$. This implies that $\langle i, v_i\rangle$ is in T_i. From Lemma 2.15, we have that, for any two vertexes $\vec{v}_i = \langle i, S_i\rangle$ and $\vec{v}_j = \langle j, S_j\rangle$ in V, either S_i is a prefix of S_j or vice versa. Suppose without loss of generality that S_j is a prefix of S_i. Then for all x, where $0 \leq x \leq n$, if $S_i[x] = \perp$, then $S_j[x] = \perp$, and if $S_j[x] \neq \perp$, then $S_i[x] = S_j[x]$. It follows that if $\langle x, v_x\rangle$ is in T_j, it is also in T_i, and if x is not in $ids(T_i)$, then it is also not in $ids(T_j)$. This implies that T_j is a face of T_i. From Lemma 2.16, it follows that, if $S_i[j] = v_j$, then S_j is a prefix of S_i. This means that, if $\langle j, v_j\rangle$ is in T_i, then T_j is a face of T_i.

Now suppose that the vertexes $\Psi(\vec{v}_0), \ldots, \Psi(\vec{v}_m)$ in $skel^0(\mathcal{X}(\mathcal{A}))$ form a simplex V in $\mathcal{X}(\mathcal{A})$. We will construct an execution α with corresponding output simplex U such that $\Psi(U) = V$. Let $W = carrier(V)$. Partition the set $ids(V)$ into a collection of nonempty *concurrency classes* of process ids, $\mathcal{C}_1, \ldots, \mathcal{C}_k$ for some $k \geq 0$, such that any two process indexes i, j are in the same concurrency class if and only if $T_i = T_j$.

We can define a total order $\prec$ on this collection of concurrency classes as follows. Let $\mathcal{C}_x, \mathcal{C}_y$ be distinct concurrency classes. Then $\mathcal{C}_x \cap \mathcal{C}_y = \emptyset$. Since both classes are nonempty, we can pick an element from each, say, $i \in \mathcal{C}_x$ and $j \in \mathcal{C}_y$. By assumption,

$T_i \neq T_j$. Then by Lemma 3.29, either T_i is a face of T_j or T_j is a face of T_i. In the first case, let $\mathcal{C}_x \prec \mathcal{C}_y$, and in the second case, let $\mathcal{C}_y \prec \mathcal{C}_x$. Thus $\prec$ is a total order of the concurrency classes.

Now use this ordered partition of the participating processes in α to define a second partition $\mathcal{C}'_1, \ldots, \mathcal{C}'_k$ of the set $ids(W)$ as follows. For each concurrency class $\mathcal{C}$ of $ids(V)$, define a concurrency class $\mathcal{C}'$ of $ids(W)$ as follows. $\mathcal{C}'$ is the union of $\mathcal{C}$ and all $i \in ids(W) - ids(V)$ such that $\mathcal{C}$ is the least concurrency class (as determined by $\prec$) such that for all $j \in \mathcal{C}$, $i \in T_j$. Note that this is a partition of all $ids(W)$ since $W = carrier(V)$. This partition gives us a new collection of concurrency classes $\mathcal{C}'_1, \ldots, \mathcal{C}'_k$.

We are now ready to construct α. First position $update_{\mathcal{C}'_i}$ actions in increasing order according to the $\prec$ ordering. For each concurrency class $\mathcal{C}'_x$, position the *inv_writeread(v)*$_{i,1}$ actions of all i such that $i \in \mathcal{C}'_x$ immediately before the $update_{\mathcal{C}'_x}$ action (their internal ordering does not matter). Similarly, position the *ret_writeread(v)*$_{i,1}$ and *decide(S)*$_i$ actions of all i such that $i \in \mathcal{C}_x$ and $i \in ids(V)$ immediately after the $update_{\mathcal{C}'_x}$ action, but before the *inv_writeread(v)*$_{i,1}$ actions associated with the next concurrency class $\mathcal{C}'_y$. Processes i whose indexes are not in $ids(W)$ do not participate and hence take no steps in α. Processes i whose indexes are in some concurrency class $\mathcal{C}'_x$ but not in $ids(V)$ do not execute a *ret_writeread(v)*$_{i,1}$ action; instead, they execute a $fail_i$ action after the $update_{\mathcal{C}'_x}$ action, but before the *inv_writeread(v)*$_{i,1}$ actions associated with the next concurrency class $\mathcal{C}'_y$. Recall that earlier concurrency classes could not have included i since by construction $\mathcal{C}'_x$ is the least class including i. By construction, each deciding process i decides S_i in α, as required. The lemma follows. □

We now consider the protocol complex of a protocol in NIIS with time complexity 1 on the input complex $\mathcal{I}$; that is, some processes access a single IS object, while some decide based only on their own inputs. We will show that, if δ is trivial, which we denote by $\delta = 1$, then this protocol complex is indeed a nonuniform chromatic subdivision.

LEMMA 4.4. *For all $k \geq 0$, the protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$ of any protocol in the NIIS model, with time complexity k on inputs in $\mathcal{I}$, is equal to some nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^k(\mathcal{I})$ up to isomorphism.*

Proof. We use induction on the time complexity k. The result holds for $k = 0$ trivially. Now suppose $k > 0$ and that the result holds for $1, \ldots, k-1$. Consider the protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$ of any protocol in the NIIS model with time complexity k on inputs in $\mathcal{I}$.

Any vertex $\vec{v}$ in $\mathcal{I}$ is labeled with $\langle i, v_i \rangle$, where i is a process id and v_i represents an input value to process i. According to the specification of NIIS protocols in section 2.6, any nonfailing process i will, upon having received the input v_i, execute either an *inv_writeread(v)*$_{i,1}$ action or a *decide(S)*$_i$ action, depending on whether $\tau(local\_state)$ evaluates to true or not. In this way, the predicate map τ induces a partition of the vertexes of $\mathcal{I}$ into two disjoint sets A and B. Since the time complexity of $\mathcal{P}_{(n,\tau,1)}$ on inputs in $\mathcal{I}$ is k, the set A must be nonempty. We now construct complexes $\mathcal{A}$ and $\mathcal{B}$ as in Definition 3.30; that is, a simplex T in $\mathcal{I}$ is in $\mathcal{A}$ if and only if all its vertexes are in A, and it is in $\mathcal{B}$ if and only if all its vertexes are in B.

The vertexes in A correspond to processes that, based on their input values, execute an *inv_writeread(v)*$_{i,1}$ action with the object IS_1. By Lemma 4.3, the output protocol complex on inputs in $\mathcal{A}$ after the first IS access equals $\mathcal{X}(\mathcal{A})$ up to isomorphism. The final protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{A})$ is given by applying $\mathcal{X}(\mathcal{A})$ as an input complex to the protocol. Since the complexity of the protocol on inputs in $\mathcal{I}$, and hence

on inputs in $\mathcal{A}$, is k, the complexity of the protocol on inputs in $\mathcal{X}(\mathcal{A})$ must be $k-1$. It follows by induction that the protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{A})$ equals some nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$ of $\mathcal{X}(\mathcal{A})$ up to isomorphism. A simplex U is in $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$ if and only if it corresponds to a valid set of outputs of an execution α of the protocol. In any execution α of the protocol, some of the participating, nonfailing processes decide on their input values (corresponding to vertexes in B), while some decide on the returned snapshots they receive from some IS object. It follows that $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$ contains any simplex in $\mathcal{B}$, any simplex in $\mathcal{P}_{(n,\tau,1)}(\mathcal{A}) = \widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, and any simplex of the form $S \star T$, where S is in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, T is in $\mathcal{B}$, and $carrier(S) \star T$ is in $\mathcal{I}$. It follows from Definition 3.30 that the protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$ equals some nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^{k}(\mathcal{I})$ of $\mathcal{I}$ up to isomorphism. □

We must also prove that, for any mappable nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^{k}(\mathcal{I})$ of an input complex $\mathcal{I}$, there is a matching protocol $\mathcal{P}_{(n,\tau,1)}$ in the NIIS model.

LEMMA 4.5. *For any mappable nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^{k}(\mathcal{I})$ of an input complex $\mathcal{I}$, there is a matching protocol $\mathcal{P}_{(n,\tau,1)}$ in the NIIS model such that the protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{I}) = \widetilde{\mathcal{X}}^{k}(\mathcal{I})$ up to isomorphism.*

Proof. Given is a vertex $\vec{v}$ in $\mathcal{I}$. The definition of the subdivision $\widetilde{\mathcal{X}}^{k}(\mathcal{I})$ induces a sequence of nonuniform chromatic subdivisions $\mathcal{I}, \widetilde{\mathcal{X}}^{1}(\mathcal{I}), \ldots, \widetilde{\mathcal{X}}^{k}(\mathcal{I})$ and corresponding sequences $\mathcal{A}_0, \ldots, \mathcal{A}_{k-1}$ and $\mathcal{B}_0, \ldots, \mathcal{B}_{k-1}$ of complexes, the former sequence specifying the subcomplex to be subdivided further at each level of recursion.

In order to construct a protocol for $n+1$ processes, we must specify the function $\tau : \bigcup_{l=0}^{k} \vartheta^l(D) \to \{\texttt{true}, \texttt{false}\}$ and the decision map $\delta : \bigcup_{l=0}^{k} \vartheta^l(D) \to D_O$. We specify τ to be `true` for all values v such that there is a vertex $\vec{v}$ in one of the complexes $\mathcal{A}_0, \ldots, \mathcal{A}_{k-1}$ with $val(\vec{v}) = v$. For all other values v, τ evaluates to `false`. This definition is well formed, since for all p, where $0 \le p \le k$, it follows from Definitions 3.28 and 3.30 that there are no two vertexes in $\widetilde{\mathcal{X}}^{p}(\mathcal{I})$ with the same process-value label pair, and for all p, q, where $0 \le p, q \le k$ and $p \ne q$, $\mathcal{B}_p$ and $\mathcal{B}_q$ have no vertexes with common labels (process id *and* value label). This concludes the proof. □

We now give the proof of Theorem 4.2.

Proof of Theorem 4.2. Let $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$ be a decision task. Lemma 4.4 states that any protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$, with worst case complexity k_S on input S, corresponds to a nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^{k}(\mathcal{I}^n)$ with level k_S on S. Suppose now the decision map δ is not trivial. Then, if $\mathcal{P}_{(n,\tau,\delta)}$ solves $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$, $\mu = \delta$ is a simplicial map from $\widetilde{\mathcal{X}}^{k}(\mathcal{I}^n)$ to $\mathcal{O}$ that is in correspondence with Γ, so $\widetilde{\mathcal{X}}^{k}(\mathcal{I}^n)$ is mappable.

Lemma 4.5 states that any mappable nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^{k}(\mathcal{I}^n)$ with level k_S on S is equal to the protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$ (where δ is trivial) of a protocol in the NIIS model with worst case complexity k_S on input S. If there is a simplicial map μ from $\widetilde{\mathcal{X}}^{k}(\mathcal{I}^n)$ to $\mathcal{O}$ that is consistent with Γ, then by setting $\delta = \mu$, we have a protocol $\mathcal{P}_{(n,\tau,\delta)}$ solving $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$ with complexity k_S on input S. The theorem follows. □

5. Approximate Agreement. As an application of Theorem 4.2, we analyze the well-known *Approximate Agreement* [15] task, defined as follows: each process $i \in \{0, \ldots, n\}$ has an input x_i taken from some finite subset of the reals and chooses a unique output y_i such that, for some predetermined $\epsilon \ge 0$, (1) $\max_i y_i - \min_i y_i < \epsilon$, and (2) for all i, $y_i \in [\min_i x_i, \max_i x_i]$.

This problem, which at first glance may seem similar to Consensus, is in fact quite different and is solvable in the read-write memory model. (If ϵ were 0 this problem

would be equivalent to Consensus and hence not solvable.) Aspnes and Herlihy [2] proved a lower bound on Approximate Agreement in the read-write memory model that implies a worst case time complexity of $\lceil \log_3 \frac{\max_i x_i - \min_i x_i}{\epsilon} \rceil$ and an upper bound of $\lceil \log_2 \frac{\max_i x_i - \min_i x_i}{\epsilon} \rceil$ in the NIIS model. We will show that this $\log_2$ versus $\log_3$ gap is not simply a technical fluke.

DEFINITION 5.1. *Let V be some finite subsequence of values from $\mathbb{R}$, at most ϵ apart from its successor. The finite $n+1$-process* Approximate Agreement *task is the tuple $\mathcal{D} = \langle I, O, \gamma \rangle$:*

- $I = \{[x_0, \ldots, x_n] \mid x_i \in V \cup \{\perp\}\}$.
- $O = \{[y_0, \ldots, y_n] \mid y_i \in V \cup \{\perp\}, (y_i, y_j \neq \perp) \Rightarrow |y_i - y_j| \leq \epsilon\}$.
- $\gamma = \{(\vec{I}, \vec{O}) \mid \vec{O}[i] \in [\min_i \vec{I}[i], \max_i \vec{I}[i]] \cup \{\perp\}\}$.

Define an input vector $\vec{I}$ to be *nontrivial* if the $\max_i x_i$ and $\min_i x_i$ are at least ϵ apart and each belongs to at least one other disjoint input vector. We can now state the complexity bounds for the Approximate Agreement problem.

THEOREM 5.2. *Given $\epsilon > 0$, there is a protocol $\mathcal{P}_{(n,\tau,\delta)}$ solving* Approximate Agreement *for any nontrivial input vector $\vec{I}$ with complexity $\lceil \log_d \frac{\max_i \vec{I}[i] - \min_i \vec{I}[i]}{\epsilon} \rceil$, where $d = 3$ if the size of the participating set of $\vec{I}$ is 2, and $d = 2$ if the size of the participating set of $\vec{I}$ is 3 or more. Moreover, this protocol is optimal for $\vec{I}$.*

We note that in many cases, for trivial input vectors one can "statically" predefine the outputs for each input value so that no access to an *IS* object is necessary.

Our proof structure will be as follows. The upper bound will follow by showing a subdivision and simplicial map and then applying Theorem 4.2. The lower bound proof will follow from a geometric observation regarding the structure of any NIIS subdivided complex for approximate agreement.

We first restate the description of the Approximate Agreement task using our topological framework.

- $\mathcal{I}$ is the closure under containment of the collection of all simplexes of the form $(\langle 0, x_0 \rangle, \ldots, \langle n, x_n \rangle)$.
- $\mathcal{O}$ is the closure under containment of the collection of all simplexes of the form $(\langle 0, y_0 \rangle, \ldots, \langle n, y_n \rangle)$, where for all i, j, $y_i \in V$ and $|y_i - y_j| \leq \epsilon$.
- $\Gamma = \{(S, T) \mid \mathit{vals}(T) \subseteq [\min \mathit{vals}(S), \max \mathit{vals}(S)]\}$.

Note that the size of the participating set for the input vector corresponding to a simplex S in $\mathcal{I}$ equals $\dim(S) + 1$.

To understand our proof approach, consider Figure 20, which shows the subdivisions induced by a three process protocol. In [2], the lower bound for any $n+1$-process algorithm is derived from a "bad" execution in which only the two processes P and Q with inputs farthest apart participate. Cast in our model, processes P and Q have inputs p and q (the corners of the input simplex in Figure 20). Because there are other simplexes adjacent to the input 1-simplex $(\langle P, p \rangle, \langle Q, q \rangle)$ that share only the input value p and, respectively, q, μ must map $\langle P, p \rangle$ to output value p and $\langle Q, q \rangle$ to output value q. An execution in the NIIS model corresponds to a sequence of chromatic subdivisions of the edge $(\langle P, p \rangle, \langle Q, q \rangle)$ (a path of 1-simplexes) from which there is a simplicial map to a path in the output complex. In the end of the execution, the vertexes of each 1-simplex along this edge must be mapped to an output 1-simplex with values less than ϵ apart. Each subdivision, corresponding to an NIIS execution step, introduces two new vertexes and splits the edge $(\langle P, p \rangle, \langle Q, q \rangle)$ in three. This implies that reaching a distance of ϵ or less along the path of simplexes in the subdivision of $(\langle P, p \rangle, \langle Q, q \rangle)$ requires at least $\log_3(\text{distance}(p, q)/\epsilon)$. This is the bound of [2].

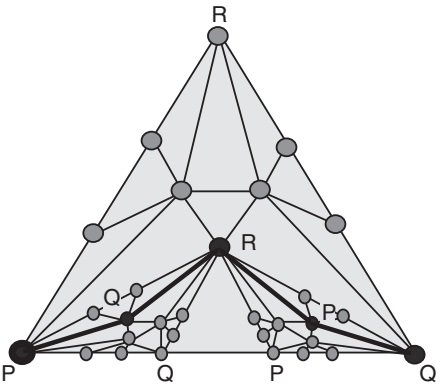


FIG. 20. *Simplex subdivided by an Approximate Agreement protocol.*

However, if one considers executions in which three processes participate, this proof does not work. Consider the first round of subdivision of the simplex for three processors. There is a path (a sequence of adjoining 1-simplexes) between the two endpoints P and Q, along which only a single vertex is introduced by the subdivision. This is the path of length two through the central vertex marked R in Figure 20 (it is not marked since its simplexes are further subdivided in the figure). So the maximum distance is cut by at most a half in the first subdivision. In the next step of subdivision, even though each of the original 1-simplexes (P, R) and (R, Q) of $\widetilde{\mathcal{X}}(S)$ can be subdivided into three simplexes, there is still a path (highlighted in the figure) from P to Q along which only a single node was added connecting P and R (and, respectively, R and Q). In general, after k subdivisions, there is always a path that was divided only 2^k times—hence the lower bound of $\log_2(\text{distance}(p, q)/\epsilon)$. Our upper bounds follow directly from Theorem 4.2 by specifying the proper subdivision and map. The thing to note about the proof we will present for Theorem 5.2 is that it will not involve any mention of the actual executions; all we need to do is argue about the topology of the inputs and outputs and then apply Theorem 4.2.

Proof. Theorem 5.2 states that, given $\epsilon > 0$, there is a protocol $\mathcal{P}_{(n,\tau,\delta)}$ solving Approximate Agreement with complexity $\left\lceil \log_d \frac{\max vals(S) - \min vals(S)}{\epsilon} \right\rceil$ on any input simplex S, where $d = 3$ if $\dim(S) = 1$ and $d = 2$ if $\dim(S) \geq 1$. Moreover, this protocol is optimal on each input simplex S.

We first establish the lower bound. Let $\mathcal{P}_{(n,\tau,\delta)}$ be a protocol that solves Approximate Agreement with worst case complexity k_S on S, where S is any simplex of dimension $n \geq \dim(S) > 0$. Let $D(S) = \max_{\vec{v},\vec{u} \in S}(val(\vec{v}) - val(\vec{u}))$ and let $D(\widetilde{\mathcal{X}}^k(\mathcal{I}))$ equal $\max_{S \in \widetilde{\mathcal{X}}^k(\mathcal{I})} D(S)$. Then Theorem 4.2 states that there is some mappable nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^k(\mathcal{I})$, with level k_S on S. We will show that $k_S \geq \left\lceil \log_d \frac{D(S)}{\epsilon} \right\rceil$. The proof uses the following lemma.

LEMMA 5.3. *Let $l \leq k$. Label the vertexes of $\widetilde{\mathcal{X}}^l(S/\mathcal{I})$ with real numbers in a way that agrees with the initial value labeling of S, and let l_S be the level of $\widetilde{\mathcal{X}}^l(S/\mathcal{I})$. Then*

$$D(\widetilde{\mathcal{X}}^l(S/\mathcal{I})) \geq \frac{D(S)}{d^{l_S}}.$$

Proof. Suppose without loss of generality that $l = l_S$. We first give the proof for the case of two participating processes and $d = 3$. By definition of $D(S)$, there

is a 1-simplex $U = (\vec{u_0}, \vec{u_1})$ in $\mathcal{S}$ such that $D(U) = D(S)$. The complex $\widetilde{\mathcal{X}}^l(U)$ contains at most 3^l 1-simplexes, denoted $U_1, \ldots, U_M$, where $M \leq 3^l$. These form a continuous path from $\vec{u}_0$ to $\vec{u}_1$, the endpoints of which are labeled with $val(\vec{u}_0)$ and $val(\vec{u}_1)$, respectively. So the best we can do is cut $D(U)$ into 3^l pieces. The triangle inequality tells us that $D(U) \leq \sum_{i=1}^{M} D(U_i) \leq M \max_i D(U_i) \leq 3^l \max_i D(U_i)$. Hence $\max_i D(U_i) \geq D(U)/3^l = D(S)/3^l$. The lemma follows, since $\max_i D(U_i) \leq D(\widetilde{\mathcal{X}}^l(S/\mathcal{I}))$.

We now prove the case where the size of the participating set is greater than 2 (and hence $\dim(S)$ is greater than 1) and $d = 2$. We argue by induction on l. The case $l = 0$ is trivial. Now suppose the claim is true for $l - 1$. By definition of $D(\widetilde{\mathcal{X}}^{l-1}(S/\mathcal{I}))$, there is a 1-simplex $U = (\vec{u_0}, \vec{u_1})$ in $\widetilde{\mathcal{X}}^{l-1}(S/\mathcal{I})$ such that $D(U) = D(\widetilde{\mathcal{X}}^{l-1}(S/\mathcal{I}))$. U is a face of some 2-simplex $U' = (\vec{u_0}, \vec{u_1}, \vec{u}_2)$. Suppose first that the next level of nonuniform chromatic subdivision does not subdivide U completely. Then there is some 1-simplex T in the level l nonuniform subdivision of U' with $D(T) \geq D(U)/2$. Since $D(U) = D(\widetilde{\mathcal{X}}^{l-1}(S/\mathcal{I}))$ and $D(T) \leq D(\widetilde{\mathcal{X}}^{l}(S/\mathcal{I}))$, the lemma follows by induction. Suppose instead that the next level of subdivision does subdivide U' completely. Then the level l subdivision has an internal vertex $\vec{m}_2$, colored with $id(\vec{u}_2)$, and two neighboring 1-simplexes $T_0 = (\vec{u}_0, \vec{m}_2)$ and $T_1 = (\vec{m}_2, \vec{u}_1)$. The triangle inequality then tells us that $D(U) \leq D(T_0) + D(T_1) \leq 2 \max_i D(T_i)$, where $i \in \{0, 1\}$. It follows that $D(\widetilde{\mathcal{X}}^{l}(S/\mathcal{I})) \geq D(\widetilde{\mathcal{X}}^{l-1}(S/\mathcal{I}))/2$. The lemma follows by induction. □

Suppose now that there exists a chromatic simplicial map $\mu : \widetilde{\mathcal{X}}^k(\mathcal{I}) \to \mathcal{O}$ such that, for all simplexes T in $\widetilde{\mathcal{X}}^k(\mathcal{I})$, $\mu(T) \in \Gamma(carrier(T))$. We can associate this map with a labeling of the vertexes in $\widetilde{\mathcal{X}}^k(\mathcal{I})$ as follows. Label each vertex $\vec{v}$ in $\widetilde{\mathcal{X}}^k(\mathcal{I})$ with $val(\mu(\vec{v}))$. This labeling agrees with the input value labeling of $\mathcal{I}$, since for any vertex $\vec{v}$, the task specification requires that for any simplex S_0 that contains $\vec{v}$, it must be the case that $\mu(\vec{v})$ is in the range of S_0. Based on the nontriviality assumption, choose two neighboring simplexes S_0 and S_1 containing $\vec{v}$ such that the intersection of the ranges of S_0 and S_1 is $val(\vec{v})$. It follows that $\mu(\vec{v}) = val(\vec{v})$. Now let T be any simplex in $\widetilde{\mathcal{X}}^k(\mathcal{I})$. By definition of μ, $\mu(T)$ is a simplex in $\mathcal{O}$, and hence $D(\mu(T)) < \epsilon$. It follows that $D(T) = D(\mu(T)) < \epsilon$ and hence that $D(\widetilde{\mathcal{X}}^k(\mathcal{I})) < \epsilon$. Clearly, for any input simplex S, it follows that the labels on $\widetilde{\mathcal{X}}^k(S/\mathcal{I})$ have range less than ϵ. The previous lemma then states that $\epsilon > D(\widetilde{\mathcal{X}}^k(S/\mathcal{I})) \geq \frac{D(S)}{d^{k_S}}$. We conclude that

$$k_S \geq \left\lceil \log_d \frac{D(S)}{\epsilon} \right\rceil .$$

To prove the upper bound, we construct a mappable nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^k(\mathcal{I})$ of the input complex with level $k_S = \left\lceil \log_d \frac{D(S)}{\epsilon} \right\rceil$ on each input simplex S, according to Definition 3.30. As argued above, the requirement that the subdivision be mappable is equivalent to saying that there is a vertex labeling of $\widetilde{\mathcal{X}}^k(\mathcal{I})$ that agrees with the initial value labeling of $\mathcal{I}$ with the additional property that $D(\widetilde{\mathcal{X}}^k(\mathcal{I})) < \epsilon$.

Apply $\widetilde{\mathcal{X}}^k$ to $\mathcal{I}$ where for every input n-simplex $S \in \mathcal{I}$, and construct $\widetilde{\mathcal{X}}^k(S/\mathcal{I})$ by repeatedly applying the procedure $\widetilde{\mathcal{X}}^k$ (as specified by Definition 3.30) for each level of subdivision $l \leq k$. For each level, split the vertexes into two sets so that a vertex $\vec{v}$ is in A if there is another adjacent vertex $\vec{u}$ such that $val(\vec{v}) - val(\vec{u}) > \epsilon$; otherwise it is in B. Before applying the next level of subdivision to $\mathcal{X}(\mathcal{A})$, we relabel all new vertexes in $\mathcal{X}(\mathcal{A})$ (those not in $skel^0(\mathcal{A})$) as follows: If the dimension of $\mathcal{A}$ is 1,

label the new vertexes in $\mathcal{X}(\mathcal{A})$ with $(2\min val(\mathcal{A})+\max val(\mathcal{A}))/3$ and $(\min val(\mathcal{A})+2\max val(\mathcal{A}))/3$, respectively. This cuts the distance between the vertexes with values apart in 3. Otherwise, label the new vertexes with $(\min val(\mathcal{A})+\max val(\mathcal{A}))/2$. This cuts the distance between the values furthest apart in 2.

It is clear from this construction that, at each level of recursion, for all simplexes S in $\mathcal{I}$ we have that, if $D(\widetilde{\mathcal{X}}^l(S)) > \epsilon$, then either $D(\widetilde{\mathcal{X}}^{l+1}(S)) = D(\widetilde{\mathcal{X}}^l(S))/d$ or $D(\widetilde{\mathcal{X}}^{l+1}(S)) < \epsilon$. It follows that the level k_S of $\widetilde{\mathcal{X}}^k(\mathcal{I})$ on S is $\lceil \log_d \frac{D(S)}{\epsilon} \rceil$, where $d = 3$ if $\dim(S) = 1$, and $d = 2$ if $\dim(S) > 1$. We conclude from Theorem 4.2 that there is a wait-free protocol that solves Approximate Agreement with worst case time complexity $\lceil \log_d \frac{D(S)}{\epsilon} \rceil$ on input S, where $d = 3$ for two participating processes and $d = 2$ for three or more. □

6. Conclusion and directions for further research. This paper extended the topological framework of Herlihy and Shavit [24, 25, 26] to obtain a complete characterization of the complexity of solving decision tasks in the NIIS model, a generalization of Borowsky and Gafni's IIS model [12]. The main difference between the proof of Theorem 4.2 and Herlihy and Shavit's proof of their asynchronous computability theorem is that our proof rests on the ability to explicitly construct a protocol complex for the NIIS model and to show that this complex is indeed equal to a nonuniform chromatic subdivision. Since nonuniform chromatic subdivisions have a recursive structure, they are well suited for arguing about complexity; the level of recursion of a mappable nonuniform chromatic subdivision of a task's input complex *is* the complexity of the corresponding wait-free NIIS solution protocol.

We have applied Theorem 4.2 to tighten the upper and lower bounds on solving the *Approximate Agreement* task implied by the work of Aspnes and Herlihy [2]. The intuition behind this result as well as its formal proof are based on simple, geometric, and topological arguments about the level of nonuniform chromatic subdivision that is necessary and sufficient for mappability. We believe this is an excellent example of how Theorem 4.2 exposes subtle properties of protocols in asynchronous shared memory systems and how it allows us to reason formally about them without having to argue directly about concurrent executions.

We believe it is possible to extend our existing topological framework to develop a characterization of *work complexity*, the total number of steps taken by all processors in a computation. As was the case for time complexity, a mappable nonuniform chromatic subdivision of an input complex does contain the information necessary to describe work complexity, and the question is really how to quantify and measure it using a simple topological invariant.

Another possible direction is to try to extend the framework to other models of computation, such as the atomic snapshot model, the single-writer multireader model, or even the multiwriter multireader model. Our choice of the NIIS model was motivated by the fact that its protocol complex is highly structured and corresponds to a nonuniform chromatic subdivision, as the proof of Theorem 4.2 shows. Other, less restricted, models, such as atomic snapshots, do not have this property, and so in order to prove a result similar to Theorem 4.2 in any of these models, one would need to identify some invariant, recursive substructure that one can model topologically with reasonable ease.

An alternative approach would be to use simulation techniques to relate the NIIS model to other models of computation, thereby obtaining an indirect characterization of the complexity of solving decision tasks in these models. Currently, however, the best known wait-free simulation of a single IS object using atomic snapshots re-

quires $O(N)$ accesses to shared memory by each process, where N is the number of processes. There is thus an important open problem in finding an *optimal*, wait-free implementation of NIIS using atomic snapshot, and vice versa.

Acknowledgments. We wish to thank Nancy Lynch for her comments on initial drafts of this paper. We also thank the anonymous referees for their careful reading of the manuscript and for providing an abundance of constructive comments.

REFERENCES

[1] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit, *Atomic snapshots of shared memory*, J. ACM, 40 (1993), pp. 873–890.

[2] J. Aspnes and M. P. Herlihy, *Wait-Free Data Structures in the Asynchronous Pram Model*, manuscript, Brown University, Providence, RI, 1996.

[3] H. Attiya and S. Rajsbaum, *A Combinatorial Topology Framework for Wait-Free Computability*, preprint, The Technion, Haifa, Israel, 1995.

[4] H. Attiya, A. Bar-Noy, and D. Dolev, *Sharing memory robustly in message-passing systems*, J. ACM, 42 (1995), pp. 124–142.

[5] H. Attiya, A. Bar-Noy, D. Dolev, D. Peleg, and R. Reischuk, *Renaming in an asynchronous environment*, J. ACM, 37 (1990), pp. 524–548.

[6] H. Attiya, N. Lynch, and N. Shavit, *Are wait-free algorithms fast?*, J. ACM, 41 (1994), pp. 725–763.

[7] O. Biran, S. Moran, and S. Zaks, *A combinatorial characterization of the distributed* 1*-solvable tasks*, J. Algorithms, 11 (1990), pp. 420–440.

[8] E. Borowsky, *Capturing the Power of Resiliency and Set Consensus in Distributed Systems*, Technical report, University of California Los Angeles, Los Angeles, CA, 1995.

[9] E. Borowsky, E. Gafni, N. Lynch, and S. Rajsbaum, *The BG distributed simulation algorithm*, Distrib. Comput., 14 (2001), pp. 127–146.

[10] E. Borowsky and E. Gafni, *Generalized FLP impossibility result for t-resilient asynchronous computation*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing, ACM, New York, 1993, pp. 91–100.

[11] E. Borowsky and E. Gafni, *Immediate atomic snapshots and fast renaming*, in Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing, ACM, New York, 1993, pp. 41–51.

[12] E. Borowsky and E. Gafni, *A simple algorithmically reasoned characterization of wait-free computation (extended abstract)*, in Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing, ACM, New York, 1997, pp. 189–198.

[13] S. Chaudhuri, *Agreement is harder than consensus: Set consensus problems in totally asynchronous systems*, in Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing, ACM, New York, 1990, pp. 311–234.

[14] S. Chaudhuri, M. Herlihy, N. A. Lynch, and M. R. Tuttle, *Tight bounds for k-set agreement*, J. ACM, 47 (2000), pp. 912–943.

[15] D. Dolev, N. A. Lynch, S. S. Pinter, E. W. Stark, and W. E. Weihl, *Reaching approximate agreement in the presence of faults*, J. ACM, 33 (1986), pp. 499–516.

[16] A. D. Fekete, *Asymptotically optimal algorithms for approximate agreement*, in Proceedings of the 5th Annual ACM Symposium on Principles of Distributed Computing, ACM, New York, 1986, pp. 73–87.

[17] M. Fischer, N. A. Lynch, and M. S. Paterson, *Impossibility of distributed consensus with one faulty process*, J. ACM, 32 (1985), pp. 374–382.

[18] E. Gafni and E. Koutsoupias, *Three-processor tasks are undecidable*, SIAM J. Comput., 28 (1999), pp. 970–983.

[19] J. Havlicek, *Computable obstructions to wait-free computability*, in Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97), IEEE Computer Society, Los Alamitos, CA, 1997, pp. 80–89.

[20] M. Herlihy and S. Rajsbaum, *Set consensus using arbitrary objects (preliminary version)*, in Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing, ACM, New York, 1994, pp. 324–333.

[21] M. Herlihy and S. Rajsbaum, *The decidability of distributed decision tasks (extended abstract)*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, ACM, New York, 1997, pp. 589–598.

[22] M. Herlihy and S. Rajsbaum, *Algebraic spans*, Math. Structures Comput. Sci., 10 (2000), pp. 549–573.
[23] M. Herlihy, S. Rajsbaum, and M. R. Tuttle, *Unifying synchronous and asynchronous message-passing models*, in Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing, ACM, New York, 1998, pp. 133–142.
[24] M. Herlihy and N. Shavit, *The asynchronous computability theorem for t-resilient tasks*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing, ACM, New York, 1993, pp. 111–120.
[25] M. Herlihy and N. Shavit, *A simple constructive computability theorem for wait-free computation*, in Proceedings of the 26th Annual ACM Symposium on Theory of Computing, ACM, New York, 1994, pp. 243–252.
[26] M. Herlihy and N. Shavit, *The topological structure of asynchronous computability*, J. ACM, 46 (1999), pp. 858–923.
[27] M. P. Herlihy, *Wait-free synchronization*, ACM Trans. Programming Languages and Systems, 13 (1991), pp. 123–149.
[28] N. A. Lynch and M. R. Tuttle, *An introduction to input/output automata*, CWI Quarterly, 2 (1989), pp. 219–246.
[29] N. A. Lynch, *Distributed Algorithms*, Morgan–Kaufman, San Francisco, CA, 1996.
[30] J. R. Munkres, *Elements of Algebraic Topology*, Addison–Wesley, Reading, MA, 1984.
[31] G. Neiger, *Set-linearizability*, in Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing, ACM, New York, 1994, p. 396.
[32] M. Saks and F. Zaharoglou, *Wait-free k-set agreement is impossible: The topology of public knowledge*, SIAM J. Comput., 29 (2000), pp. 1449–1483.
[33] E. Schenk, *Computability and Complexity Results for Agreement Problems in Shared Memory Distributed Systems*, Technical report, University of Toronto, Toronto, Canada, 1996.
[34] E. H. Spanier, *Algebraic Topology*, Springer-Verlag, New York, 1966.

SIAM J. COMPUT.
Vol. 36, No. 2, pp. 498–515

COVERING PROBLEMS WITH HARD CAPACITIES*

JULIA CHUZHOY† AND JOSEPH (SEFFI) NAOR‡

Abstract. We consider the classical vertex cover and set cover problems with *hard* capacity constraints. This means that a set (vertex) can cover only a limited number of its elements (adjacent edges), and the number of available copies of each set (vertex) is bounded. This is a natural generalization of the classical problems which also captures resource limitations in practical scenarios.

We obtain the following results. For the unweighted vertex cover problem with hard capacities we give a 3-approximation algorithm that is based on randomized rounding with alterations. We prove that the weighted version is at least as hard as the set cover problem, yielding an interesting separation between the approximability of weighted and unweighted versions of a "natural" graph problem. A logarithmic approximation factor for both the set cover and the weighted vertex cover problem with hard capacities follows from the work of Wolsey [*Combinatorica*, 2 (1982), pp. 385–393] on submodular set cover. We provide here a simple and intuitive proof for this bound.

Key words. vertex cover, set cover, hard capacities, submodular set cover

AMS subject classifications. 68Q25, 68W25, 90C27, 90C59

DOI. 10.1137/S0097539703422479

1. Introduction.

The *set cover* problem is the following. Let $E = \{1, \ldots, n\}$ be a ground set of elements, and let $\mathcal{S}$ be a collection of sets defined over E. Each $S \in \mathcal{S}$ has a nonnegative *cost* $w(S)$ associated with it. A *cover* is a collection of sets such that their union is E. The goal is to find a cover of minimum cost. The set cover problem is a classic NP-hard problem that has been studied extensively in the literature, and the best approximation factor achievable for it is $\Theta(\log n)$ [9, 11, 20, 22].

We consider in this paper the set cover problem with *capacity* constraints, or the *capacitated set cover* problem. We assume that each set $S \in \mathcal{S}$ has a *capacity* $k(S)$ associated with it, meaning that it can cover at most $k(S)$ elements.

Generally, capacitated covering problems come in two flavors. In the case of *soft* capacities, an unbounded number of copies of each covering object is available. In the case of *hard* capacities, which is considered in this paper, each covering object (set S) has a bound (denoted by $m(S)$) on the number of available copies. Thus, a cover $\mathcal{C}$ is a multiset of input sets that can cover all the elements, while $\mathcal{C}$ contains at most $m(S)$ copies of each $S \in \mathcal{S}$, and each copy covers at most $k(S)$ elements.

The capacitated (multi-)set cover problem is a natural generalization of a basic and well-studied problem that captures practical scenarios where resource limitations are present.

A special case of the capacitated set cover problem that we consider is the *capacitated vertex cover* problem, defined as follows. An undirected graph $G = (V, E)$ is

*Received by the editors February 9, 2003; accepted for publication (in revised form) February 2, 2006; published electronically July 31, 2006. A preliminary version of this work appeared in the *Proceedings of the* 43*rd Annual IEEE Symposium on Foundations of Computer Science*, Vancouver, 2002, pp. 481–489.

http://www.siam.org/journals/sicomp/36-2/42247.html

†Laboratory for Computer Science, MIT, Cambridge, MA 02139, and Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104 (cjulia@csail.mit.edu). This work was done while this author was a graduate student at the Computer Science Department at the Technion.

‡Computer Science Department, Technion, Haifa 32000, Israel (naor@cs.technion.ac.il). This author's research was supported in part by US-Israel BSF grant 2002276 and by EU contract IST-1999-14084 (APPOL II).

given, and each vertex $v \in V$ is associated with a *cost* $w(v)$, a *capacity* $k(v)$, and a *multiplicity* $m(v)$ (we assume that no parallel edges are present). The goal is to find a minimum cost multiset U of vertices that cover all the edges such that, for each vertex $v \in V$, at most $m(v)$ copies appear in U, and each copy covers at most $k(v)$ edges adjacent to v. The capacitated vertex cover problem generalizes the well-known vertex cover problem, probably one of the most studied problems (see [19] for an overview) in the area of approximation algorithms. The best currently known approximation factor for vertex cover is $2 - \frac{\log\log|V|}{2\log|V|}$ [3, 18].

The capacitated vertex cover problem was first introduced by Guha et al. [17]. They considered the version of the problem with *soft* capacities, a special case where the number of available copies of each vertex is unbounded. A straightforward rounding of a linear programming relaxation of the problem gives a 4-approximate solution. Guha et al. [17] show a 2-approximation primal-dual algorithm, and they also give a 3-approximation for the case where each edge $e \in E$ has an (unsplittable) demand $d(e)$. (Gandhi et al. [13] provide further results on the capacitated vertex cover problem with soft capacities.) Guha et al. [17] motivate the study of the vertex cover problem with soft capacities by an application in glycobiology. The problem emerged in the redesign of known drugs involving glycoproteins and can be represented as an instance of the capacitated vertex cover problem.

Two other closely related capacitated covering problems are *capacitated facility location* and *capacitated k-median*. In both problems, the input consists of a set of facilities and a set of clients. For each facility and each client, there is a distance that defines the cost of assigning the client to the facility. Each facility f has a capacity k_f and a number of available copies m_f. Each client i has a demand d_i. The goal is to open facilities and to assign all the clients to them. In the facility location problem, each facility f has a cost w_f. Any number of facilities can be opened, as long as the number of copies of any facility f does not exceed m_f. The cost of a solution is the total cost of the open facilities plus the assignment costs of the clients. In the k-median problem, we are given a bound k on the number of facilities. A solution must contain at most k facilities, and the cost of the solution is the sum of the assignments costs of the clients to the facilities. The capacitated set cover problem is a special case of facility location with hard capacities, where all the distances are either 0 or ∞ (note that this distance function is not a metric). Bar-Ilan, Kortsarz, and Peleg [2] gave an $O(\log n + \log M)$-approximation for the facility location with hard capacities, where M is the value of the maximum input parameter.

Prior work. There is extensive research on the set cover problem, and the reader is referred to the surveys in [15, 7, 1, 26, 19]. The set cover problem is known to be $\Omega(\log n)$ hard to approximate [11, 23]. A greedy heuristic gives an $O(\log n)$-approximation [9, 22] for the set cover problem.

Wolsey [30] considered the *submodular set cover* problem. Let f be an integer valued function defined over all subsets of a finite set of elements E. Function f is called *nondecreasing* if $f(S) \leq f(T)$ for all $S \subseteq T \subseteq E$, and *submodular* if $f(S) + f(T) \geq f(S \cap T) + f(S \cup T)$ for all $S, T \subseteq E$. The input to the submodular set cover problem is a family $\mathcal{S}$ of subsets of E together with a nonnegative cost function. There is a nonnegative nondecreasing submodular function f defined over all collections of the input sets. The goal is to find a minimum cost collection $\mathcal{P}$ of sets such that $f(\mathcal{P}) = f(\mathcal{S})$. The special case where $f(\mathcal{S}) = |\cup_{S \in \mathcal{S}} S|$ for each set $S \in \mathcal{S}$ is the classical set cover problem.

Consider the capacitated set cover problem. We can assume without loss of generality that the multiplicities of all the sets are unit, as is the case in the submodular

set cover problem, by viewing each one of the $m(S)$ copies of each set $S \in \mathcal{S}$ as a distinct set. For any family $\mathcal{A}$ of input sets, define $f(\mathcal{A})$ to be the maximum number of elements that $\mathcal{A}$ can cover (given the capacity constraints). It is not hard to see that f is a nondecreasing nonnegative submodular function. Wolsey [30] showed using dual fitting that the approximation factor of a greedy heuristic for the submodular set cover problem is $1 + O(\log f_{\max})$, where $f_{\max} = \max_{S \in \mathcal{S}} f(\{S\})$.

Metric facility location is a well-studied special case of the facility location problem, where the distance function defined on the clients is a metric. Many heuristics, as well as approximation algorithms with bounded performance guarantees, were developed for this version [6, 25, 27, 29]. For the metric facility location problem with hard capacities, Pál, Tardos, and Wexler [28] gave a $(9 + \epsilon)$-approximation using local search. This result has been improved to $(8 + \epsilon)$-approximation by Mahdian and Pál [24], and the best currently known approximation is $(6 + \epsilon)$, due to Zhang, Chen, and Ye [31] and Garg, Khandekar, and Pandit [16]. Unlike the facility location problem, it is not known whether the capacitated k-median problem has a constant approximation, even if the capacities are soft. Bartal, Charikar, and Raz [4] show a constant factor bicriteria approximation for the soft capacities version, where the number of open facilities k is exceeded by at most a constant. Chuzhoy and Rabani [8] provide a different constant factor bicriteria approximation for the same problem. In their algorithm, the capacities are violated by at most a constant factor, while the number of open facilities is guaranteed to be at most k.

1.1. Our contribution. Our first result is a randomized 3-approximation algorithm for the unweighted capacitated vertex cover problem in simple graphs (i.e., with no parallel edges). Our algorithm uses randomized rounding with *alterations*. The first rounding step in our algorithm applies randomized rounding where the probabilities are derived from a solution to a linear programming relaxation of the problem. However, the rounding may not yield a feasible cover, and therefore we need to add more vertices to the cover. This is done in the alteration step. Our analysis uses a charging scheme to bound the number of vertices that are added to the cover in this step. We also prove that the more general version where edges have unsplittable demands is not approximable in the presence of hard capacities. Contrast this with the 3-approximation algorithm of Guha et al. [17] for this case (with soft capacities).

We next consider the weighted capacitated vertex cover problem and prove that it is set cover hard. This means that the best approximation factor that can be achieved for this problem is $\Omega(\log n)$. Our hardness proof holds even for the case of $\{0, 1\}$ weights and unit multiplicity. Interestingly, we are not aware of any other "natural" graph problem where there is a logarithmic separation between the approximability of weighted and unweighted versions. (However, there are several examples of problems where the unweighted version is polynomially solvable, while the weighted version is NP-hard.)

We leave open the version of the capacitated vertex cover problem where the graph is unweighted, yet there are parallel edges. Our constant-factor approximation algorithm is not applicable to this version of the problem. Following our work, Gandhi et al. [12] obtained a 2-approximation for the unweighted vertex cover with hard capacities, using a modification of our algorithm.

We proceed to consider the capacitated set cover problem. As already noted, it follows from Wolsey's work [30] that a natural greedy heuristic achieves an approximation factor of $O(\log n)$ for this problem. We note that the integrality gap of the natural linear programming relaxation of the problem is unbounded, similar to the

case of facility location with hard capacities [28]. Indeed, Wolsey uses a different linear programming formulation (see section 6 for a formulation of the linear program). We consider the same greedy heuristic as Wolsey and provide a direct combinatorial proof of the approximation factor of this heuristic. We believe that our proof is simple and intuitive. We note that the main obstacle in applying the "standard" (set cover) charging scheme in the presence of hard capacities is that it is not clear how to "charge" the sets in the optimal solution for the sets in the solution computed by the greedy algorithm. Since there are hard capacities, the assignment of elements to sets in the cover is dynamic, and, moreover, elements may be covered and uncovered several times during the iterations of the algorithm.

The paper is organized as follows. In section 3, we show a 3-approximation algorithm for unweighted capacitated vertex cover. In section 4.1 we show that the weighted capacitated vertex cover problem is at least as hard as the set cover problem, even in the case where $m(v) = 1$ for all $v \in V$. In section 4.2 we consider the version where edges have unsplittable demands, and show that this version is not approximable in the presence of hard capacities. In section 5 we provide a description of the greedy algorithm for the set cover problem with hard capacities, and give a simple proof that the algorithm achieves an $O(\log n)$-approximation, implying an $O(\log |V|)$-approximation for the weighted capacitated vertex cover problem. In section 6 we discuss extensions of the algorithm to more general covering problems, such as submodular set cover and multiset multicover.

2. Preliminaries. A set cover instance with hard capacities contains a ground set of elements $E = \{1, \ldots, n\}$ and a collection of sets $\mathcal{S}$ defined over E. Each set $S \in \mathcal{S}$ is associated with a nonnegative cost $w(S)$, a capacity $k(S)$ that bounds the number of elements it can cover, and a bound $m(S)$ on the number of available copies of S. Let $\mathcal{P}$ be a multiset of sets from $\mathcal{S}$. Then $C \subseteq \mathcal{P} \times E$ is called a *partial cover* iff for each $(S, e) \in C$, $e \in S$. We say that element $e \in E$ is covered by S in C if $(S, e) \in C$. Without loss of generality we can assume that each element $e \in E$ is covered in C at most once. Cover C is *feasible* if $\mathcal{P}$ contains at most $m(S)$ copies of each $S \in \mathcal{S}$, and each copy covers at most $k(S)$ elements. The *value* of C, i.e., the number of elements it covers, is denoted by $|C|$. Given a multiset $\mathcal{P}$, denote by $f(\mathcal{P})$ the maximal value of a feasible (partial) cover $C \subseteq \mathcal{P} \times E$. The cost of C is defined to be the sum of the costs of the sets belonging to $\mathcal{P}$. We start by showing that when $\mathcal{P}$ is fixed, a feasible cover $C \subseteq \mathcal{P} \times E$ of value $f(\mathcal{P})$ can be computed in polynomial time.

LEMMA 2.1. *Given an instance of set cover with hard capacities and a multiset $\mathcal{P}$ of sets from $\mathcal{S}$, a cover C of value $f(\mathcal{P})$ can be computed in polynomial time. In particular, whether $\mathcal{P}$ defines a feasible solution to the set cover problem can be established.*

Proof. For each $S \in \mathcal{S}$, let $m^{\mathcal{P}}(S)$ denote the number of copies of S that appear in $\mathcal{P}$. We build the following directed network. Let $G = (L, R, E')$ be the directed incidence graph of $\mathcal{P}$ and E; i.e., L contains a vertex for each copy of each set in $\mathcal{P}$: $L = \{v_i(S) |\ S \in \mathcal{S}, 1 \le i \le m^{\mathcal{P}}(S)\}$, $R = E$. For each $v_i(S) \in L$, $e \in R$, there is an edge $(v_i(S), e) \in E'$ of capacity 1 iff $e \in S$. Add a source vertex s and an edge $(s, v_i(S))$ of capacity $k(S)$ for each $S \in \mathcal{S}, 1 \le i \le m^{\mathcal{P}}(S)$. Add a sink vertex t and an edge (e, t) of capacity 1 for each $e \in E$.

Consider the maximum flow in this network. The value of the flow is at least $f(\mathcal{P})$, since the optimal cover defines a feasible flow in the network. Also, the maximum flow in the network is integral, and thus it induces a feasible partial cover of the same value.

Clearly, $\mathcal{P}$ is a feasible solution to the set cover problem iff $f(\mathcal{P}) = |E|$ and for each $S \in \mathcal{S}$, $m^{\mathcal{P}}(S) \leq m(S)$. □

Since vertex cover with hard capacities is a special case of set cover with hard capacities (where each vertex $v \in V$ can be viewed as a set whose elements are the edges adjacent to v), all the above definitions, as well as Lemma 2.1, can also be applied to the vertex cover problem.

3. Vertex cover with hard capacities. In this section, we show a randomized 3-approximation algorithm for the unweighted capacitated vertex cover problem. Our starting point is the following linear programming relaxation of the problem. For $v \in V$, let $x(v)$ be a variable indicating the number of copies of v that belong to the cover. For $e = (u, v) \in E$, let $y(e, v)$ be a variable indicating whether vertex v covers edge e. Denote by (x, y) a solution to (UVC). For each $v \in V$, $N(v)$ denotes the set of edges adjacent to v.

$$
\begin{array}{lll}
\text{(UVC)} & \min \quad \displaystyle\sum_{v \in V} x(v) & \\
& \text{s.t.} & \\
(1) & y(e,u) + y(e,v) = 1 & \text{for all } e = (u,v) \in E, \\
(2) & y(e,v) \leq x(v) & \text{for all } e \in E,\ v \in e, \\
(3) & \displaystyle\sum_{e \in N(v)} y(e,v) \leq k(v) \cdot x(v) & \text{for all } v \in V, \\
& x(v), y(e,v) \ \geq \ 0 & \text{for all } v \in V,\ e \in E, \\
& x(v) \ \leq \ m(v) & \text{for all } v \in V.
\end{array}
$$

LEMMA 3.1. *Let (x, y) be a feasible solution to* (UVC)*, where x is integral. Then there exists a feasible solution (x, y') to* (UVC)*, where y' is integral. Moreover, this solution can be computed in polynomial time (given x).*

Proof. Let U be the multiset of vertices defined by x; i.e., for each vertex $v \in V$, there are exactly $x(v)$ copies of v in U. We use Lemma 2.1 to compute an (integral) cover y' of the edges by vertices in U. Note that y induces a fractional flow of value $|E|$ in the network constructed in the proof of Lemma 2.1. Thus, $f(U) = |E|$, and therefore, in y', all the edges are covered. □

3.1. A simple 8-approximation algorithm. In this section we show a simple 8-approximation algorithm for the special case of unit multiplicities (i.e., for each vertex exactly one copy is available). The description and the analysis of the algorithm are presented in an informal way. The goal is to give an intuitive explanation of the ideas behind the algorithm. The next section contains a formal description and analysis of a modified (and more complicated) version of the algorithm, which achieves a 3-approximation for the general version (with arbitrary multiplicities).

Let (x, y) be a fractional optimal solution to (UVC). We will find a feasible solution (x', y'), where x' is integral and the expected cost is at most 8 times the cost of the original solution (x, y). By Lemma 3.1, (x', y') can be converted into an integral solution of the same cost. The algorithm consists of three steps.

Step 1 (*setting it up*). We define $U = \left\{v \mid x_v \geq \frac{1}{2}\right\}$. Note that each edge $e \in E$ has at least one endpoint in U. Let $\overline{U} = V \setminus U$, and let E' denote all the edges with one endpoint in $\overline{U}$. Thus, we have two types of edges: edges with both endpoints in

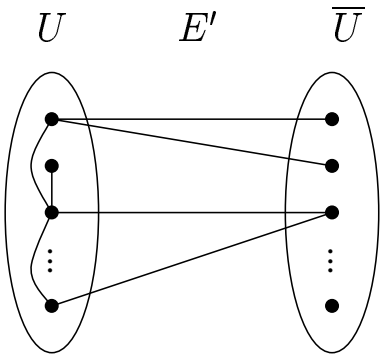


FIG. 3.1. *The sets U and $\overline{U}$.*

U, and edges with one endpoint in U and one endpoint in $\overline{U}$. Figure 3.1 shows the partition of the input graph vertices into sets U, $\overline{U}$, and the types of edges present in the graph.

For each vertex $u \in U$, let $N'(u)$ denote the set of edges in E' incident to u. Set $\ell(u) = \sum_{e \in N'(u)} y(e,u)$ and $r(u) = \sum_{e=(u,v) \in N'(u)} y(e,v) = |N'(u)| - \ell(u)$. Note that the value of $\ell(u)$ denotes the total contribution of u to the coverage of edges in $N'(u)$ (or the "budget" of u), and $r(u)$ denotes the total contribution of vertices in $\overline{U}$ to the coverage of these edges (or the "deficit" of u). The idea is to choose a small subset of vertices $I \subseteq \overline{U}$ such that each $u \in U$ can receive a contribution of at least $r(u)$ from the vertices in I for covering the edges in $N'(u)$. The coverage of the edges with both endpoints in U remains the same as in the linear programming solution. The construction of set I is performed in two steps: randomized rounding and then alterations.

Step 2 (*randomized rounding*). Each vertex $v \in \overline{U}$ is chosen randomly and independently into I with probability $2x_v$. Consider some edge $e = (u,v)$ with $u \in U$, $v \in I$. We set the contribution of v to the coverage of e to be $z(e,v) = \frac{y(e,v)}{x(v)}$. Note that since $y(e,v) \leq x(v)$ is required in (UVC), $z(e,v) \leq 1$, and it follows from constraint (3) that the capacity of v is not exceeded. If vertex $u \in U$ receives a contribution of at least $r(u)$ from the vertices in I, then the budget of u is sufficient to complete the fractional cover of all the edges adjacent to u. However, it is still possible that some vertices $u \in U$ receive a contribution smaller than $r(u)$. This is corrected in the next step.

Step 3 (*alterations*). We denote by P the vertices in U that are in "deficit," i.e.,

$$P = \left\{ u \in U \mid \sum_{e=(v,u) \in E', v \in I} z(e,v) < r(u) \right\}.$$

We proceed iteratively. In each iteration, a new vertex v that fractionally covers some edges adjacent to vertices in P is added to I. Then set P is updated. We continue until P becomes an empty set. The cost of the newly added vertices is charged to the vertices in P.

An iteration is performed as follows. (See Figure 3.2.) Let u be some vertex in P. Then there is at least one edge $e = (v,u) \in E'$, where $v \notin I$. Add v to I. For each edge $e' = (v,w)$, where $w \in P$, $w \neq u$, set the contribution of v to the coverage of e' to be $z(e',v) = \frac{y(e',v)}{x(v)}$. The contribution of v to the coverage of e is defined to

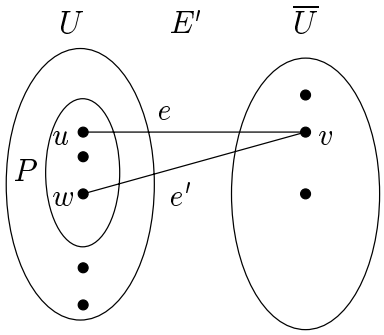


FIG. 3.2. *A Step* 3 *iteration.*

be the minimum between 1 and the remaining capacity of v (which must be at least $y(e,v)$). The cost of v is charged to the vertices in P as follows. Each $w \in P$, $w \neq u$ such that $e' = (w,v) \in E$, pays $z(e',v)$, which is exactly the contribution of v to the coverage of edge e'. The remaining cost is charged to u.

Observe that at the end of this procedure, the cost charged to each vertex $u \in P$ is at most $r(u)+1$. Let i be the last iteration when u belongs to P. In each iteration, u is charged with the amount which is bounded by the contribution it receives in this iteration. Once the total contribution exceeds $r(u)$, u is removed from P. So at the beginning of iteration i, the total amount charged to u does not exceed $r(u)$, and u pays at most 1 in iteration i.

Analysis. The cost of the algorithm is divided into three parts.

1. The cost incurred in Step 1 is at most $2\sum_{u\in U} x(u)$.
2. The expected cost of randomized rounding is at most $2\sum_{u\in \overline{U}} x_u$.
3. The expected cost of Step 3 is bounded as follows. Consider some $u \in U$.
   - If $r(u) \leq 2$, then u pays at most $r(u)+1 = 3$ for Step 3.
   - Assume that $r(u) > 2$. In this case the probability that u belongs to P after Step 2 is at most $\frac{2}{r(u)}$ (this follows from a simple application of the Chebyshev inequality; since the next section contains a similar proof, we omit the proof here). Therefore, the expected cost incurred by u is at most $\frac{2}{r(u)} \cdot (r(u)+1) \leq 3$.

   In total, the expected cost of Step 3 is at most $3|U| \leq 6\sum_{u\in U} x(u)$.

Summing up over the three steps, the expected cost of the solution is bounded by

$$8 \cdot \sum_{u\in U} x(u) + 2\sum_{v\in\overline{U}} x(v) \leq 8 \cdot \sum_{v\in V} x(v).$$

3.2. A 3-approximation algorithm. In this section we show a randomized 3-approximation algorithm for vertex cover with hard capacities and arbitrary multiplicities. The algorithm is based on the ideas presented in the previous section. Consider a fractional optimal solution (x,y) to (UVC). We show how to round this solution, obtaining a feasible solution (x',y'), where x' is integral. By Lemma 3.1, x' induces an integral capacitated vertex cover. As before, the rounding algorithm consists of three major steps.

Step 1 (*setting it up*). We need the following definitions.

- Define $U = \{u \mid x(u) \geq \frac{1}{3}\}$ and $\overline{U} = V \setminus U$. Let U' be the multiset of vertices, where for each $u \in U$ there are $\lceil x(u) \rceil$ copies of u in U'.

- Define E' to be the set of edges with one endpoint in U and the other endpoint in $\overline{U}$.
- For each $u \in V$, $N'(u) = E' \cap N(u)$.
- For each $u \in U$, define: $\ell(u) = \sum_{e \in N'(u)} y(e,u)$ and $r(u) = \sum_{e=(u,v) \in N'(u)} y(e,v) = |N'(u)| - \ell(u)$.
  Note that the value of $\ell(u)$ denotes the total contribution of u to the coverage of edges in $N'(u)$, and $r(u)$ denotes the total contribution of vertices in $\overline{U}$ to the coverage of these edges.
- For each $u \in U$, define $\epsilon(u) = \frac{\lceil x(u) \rceil}{x(u)} - 1$ and $h(u) = (1 - 2\epsilon(u))r(u)$. The meaning of these variables is explained below.

The constraints of (UVC) guarantee that each edge $e = (u,v) \in E$ has at least one endpoint in U: Since $y(e,u) + y(e,v) = 1$, $y(e,u) \le x(u)$, and $y(e,v) \le x(v)$, it follows that either $x(u) \ge \frac{1}{3}$ or $x(v) \ge \frac{1}{3}$ must hold.

Consider a vertex $u \in U$. Let (u,v) be some edge such that $v \in \overline{U}$. Since $y(e,v) \le x(v) < \frac{1}{3}$, it follows that $x(u) \ge y(e,u) > \frac{2}{3}$. It also follows that for each $u \in U$, $\ell(u) \ge 2r(u)$, since $y(e,u) \ge 2y(e,v)$ for each $(u,v) \in N'(u)$.

Our cover is going to contain the vertices of U' together with a subset $I \subseteq \overline{U}$, such that $U' \cup I$ can fractionally cover all the edges. (By Lemma 3.1, $U' \cup I$ is also an integral feasible vertex cover.)

First, we round up $x(u)$ to be equal to $\lceil x(u) \rceil$ for each vertex $u \in U$. As a result, u can increase its contribution to the coverage of the edges belonging to $N'(u)$ by a factor of $\lceil x(u) \rceil / x(u)$; i.e., now it can contribute $\frac{\lceil x(u) \rceil}{x(u)} l(u)$ to the coverage of $N'(u)$. By the definition of $\epsilon(u)$, the new contribution of u is at least $\ell(u)(1+\epsilon(u)) \ge \ell(u) + 2r(u)\epsilon(u)$. If $\epsilon(u) \ge \frac{1}{2}$, then this is enough to complete the coverage of $N'(u)$. Therefore, assume that $\epsilon(u) < \frac{1}{2}$. To complete the fractional cover, we need an additional coverage of value $(1 - 2\epsilon(u))r(u) = h(u)$ from vertices belonging to $\overline{U}$, since $\ell(u) + r(u)$ suffices to cover $N'(u)$. Our goal in the next two steps is to find $I \subseteq \overline{U}$ such that for each $u \in U$, the vertices from I can contribute at least $h(u)$ to the coverage of $N'(u)$.

Step 2 (*randomized rounding*). Each vertex $v \in \overline{U}$ is independently chosen to be in I with probability equal to $3x(v)$. Note that for each $v \in \overline{U}$, we add at most one copy of v to I. For each vertex $v \in I$, for each $e \in N'(v)$, define a new cover of edge e by vertex v: $z(e,v) = \frac{y(e,v)}{x(v)}$.

Step 3 (*alterations*). In this step we start with a feasible fractional solution (x', y') and iteratively alter it until x' becomes integral, while maintaining feasibility of (x', y'). We denote by P the vertices in U that are in "deficit," i.e.,

$$P = \left\{ u \in U \mid \sum_{e=(v,u) \in E', v \in I} z(e,v) < h(u) \right\}.$$

Our initial feasible solution (x', y') for (UVC) is defined as follows: If $v \in U$, then $x'(v) = \lceil x(v) \rceil$. If $v \in I$, then $x'(v) = 1$. Otherwise $x'(v) = x(v)$. For $e = (u,v)$, $y'(e,v)$ and $y'(e,u)$ are defined as follows:

- If $u, v \in U$, then $y'(e,u) = y(e,u)$ and $y'(e,v) = y(e,v)$.
- If $u \in U \setminus P$: if $v \in I$, then $y'(e,v) = z(e,v)$; else $y'(e,v) = 0$. Set $y'(e,u) = 1 - y'(e,v)$. Note that since $u \notin P$, it has enough capacity to complete the cover of $N'(u)$.
- If $u \in P$: if $v \in I$, then $y'(e,v) = z(e,v)$ and $y'(e,u) = 1 - z(e,v)$. (Note that $y'(e,u) \le y(e,u)$, since $z(e,v) \ge y(e,v)$.) Else $(v \notin I)$; set $y'(e,u) = y(e,u)$

and $y'(e,v) = y(e,v)$.

It is easy to see that (x', y') is a feasible solution for (UVC). We now show how to get rid of P by adding new vertices to I. We charge the cost of the new vertices added to I to the vertices of P.

PROCEDURE ELIMINATE. While $P \neq \emptyset$:

1. Let $u \in P$, $e = (u,v) \in E'$, such that $v \in \overline{U} \setminus I$ (there must be at least one such v). Let $P' = \{w \in P \mid w \neq u,\ e' = (w,v) \in E'\}$.
2. Add v to I (set $x'(v) = 1$).

   Update the cover: For each $w \in P'$, where $e' = (v,w) \in E'$, set $y'(e',v) := z(e',v) = \frac{y(e',v)}{x(v)}$. Set $y'(e',w) := 1 - y'(e',v)$. Note that the value of $y'(e',w)$ can only decrease. Set $y'(e,v)$ to be the minimum between 1 and the remaining capacity of v (which must be at least $y(e,v)$). Set $y'(e,u) = 1 - y'(e,v)$.

   Update the set P: For each $w \in P$ for which $\sum_{e=(w,a):a\in I} y'(e,a) \geq h(w)$, remove w from P. Update the cover of $N'(w)$ as follows. For each $e = (b,w) \in E'$ such that $b \notin I$, set $y'(e,b) = 0$ and $y'(e,w) = 1$. Note that w has enough capacity to cover all such edges.

It is easy to see that feasibility is maintained after each iteration. The number of iterations of Procedure Eliminate is bounded by $|\overline{U}|$, since $|I|$ is increased by one in each iteration. At the end, when P becomes empty, for each v with $x'(v) < 1$, we set $x'(v) = 0$. In the final solution, for each $v \in U \cup I$, $x(v) = 1$, and for all other vertices v, $x(v) = 0$. The next theorem follows from the discussion.

THEOREM 3.2. *The algorithm computes a feasible solution (x', y') to* (UVC)*, where x' is integral.*

To obtain an integral capacitated vertex cover, we apply Lemma 3.1 to the solution (x', y').

3.3. Analysis. The analysis of the rounding is divided into two parts.

Charging scheme for Step 3. We show that we can charge the cost of adding vertices to I in Procedure Eliminate to the vertices in P, such that each $u \in P$ pays at most $h(u)+1$. Consider an iteration of Procedure Eliminate. We charge the vertices of $P' \cup \{u\}$ for adding v to I. Each $w \in P'$, where $e' = (w,v) \in E'$, pays $z(e',v)$ (which is exactly the contribution of v to the cover of e'). Vertex u pays the remaining cost (if any remains), which is also at most the contribution of v to the cover of the edge (u,v). We now bound the total amount charged to $a \in P$. While a is still in P, at each iteration it pays at most the amount of coverage that edges in $N'(a)$ get from the newly added vertex v. Once the coverage of $N'(a)$ coming from vertices in I exceeds $h(a)$, a is removed from P. Therefore, in total a pays at most $h(a) + 1$.

Bounding the cost. We now bound the total cost of the solution produced.

CLAIM 3.3. *Let $u \in U$ such that $r(u) \geq \frac{3}{4(1+\epsilon(u))^2}$. Then, the probability that $u \in P$ after Step* 2 *is at most $\frac{3}{4(1+\epsilon(u))^2 r(u)}$.*

Proof. Consider $e = (u,v) \in N'(u)$. We define the random variable

$$t_e = \begin{cases} z(e,v), & v \in I, \\ 0 & \text{otherwise.} \end{cases}$$

Variables t_e are independent since there are no parallel edges in the graph. Note that

- $u \in P$ iff $\sum_{e\in N'(u)} t_e < h(u)$;

- the expectation of $\sum_{e\in N'(u)} t(e)$ is

$$\begin{aligned}\mu &= \mathbf{Exp}\left[\sum_{e\in N'(u)} t_e\right]\\ &= \sum_{e=(v,u)\in N'(u)} 3z(e,v)\cdot x(v)\\ &= 3r(u);\end{aligned}$$

- the variance of $\sum_{e\in N'(u)} t(e)$ is

$$\begin{aligned}\sigma^2 &= \mathbf{Var}\left[\sum_{e\in N'(u)} t_e\right]\\ &= \sum_{e=(u,v)\in N'(u)} z^2(e,v)\cdot 3x(v)\cdot(1-3x(v))\\ &\le \mu.\end{aligned}$$

When applying Chebyshev's inequality to the random variable $\sum_{e\in N'(u)} t_e$, it follows that

$$\begin{aligned}&\mathbf{Prob}\left[\sum_{e\in N'(u)} t_e < h(u)\right]\\ &\le \mathbf{Prob}\left[\left|\sum_{e\in N'(u)} t_e - \mu\right| \ \ge\ \mu - r(u)(1-2\epsilon(u))\right]\\ &= \mathbf{Prob}\left[\left|\sum_{e\in N'(u)} t_e - \mu\right| \ \ge\ 2r(u)(1+\epsilon(u))\right]\\ &\le \frac{\sigma^2}{4r^2(u)(1+\epsilon(u))^2}\\ &\le \frac{3}{4r(u)(1+\epsilon(u))^2}. \qquad \square\end{aligned}$$

We are now ready to compute the expected cost of the solution.

- For $v\in\overline{U}$, the expected cost we pay in Step 2 is $3x(v)$.
- For $u\in U$, where $N'(u)=\emptyset$ or $\epsilon(u)\ge\frac{1}{2}$, we pay at most $3x(u)$ in Step 1, and we do not pay in Step 3.
- For $u\in U$, where $N'(u)\neq\emptyset$ and $\epsilon(u)<\frac{1}{2}$, for the sake of convenience, denote $k=\lceil x(u)\rceil$, $\epsilon=\epsilon(u)$, $x=x(u)$. Consider the following two cases:
  - If $r(u)\ge\frac{3}{4(1+\epsilon)^2}$, then in Step 1 we pay k for the copies of u we use. In Step 3, we pay at most $h(u)+1$ with probability at most $\frac{3}{4r(u)(1+\epsilon)^2}$.

Thus, the expected cost is bounded by

$$\begin{aligned} k + (h(u)+1) \cdot \frac{3}{4r(u)(1+\epsilon)^2} &= k + ((1-2\epsilon)r(u)+1)\,\frac{3}{4r(u)(1+\epsilon)^2} \\ &= k + \frac{3(1-2\epsilon)}{4(1+\epsilon)^2} + \frac{3}{4r(u)(1+\epsilon)^2} \\ &\le k + 1 + \frac{3(1-2\epsilon)}{4(1+\epsilon)^2}. \end{aligned}$$

– If $r(u) < \frac{3}{4(1+\epsilon)^2}$, then in Step 1 we pay k for the copies of u, and at most $h(u)+1$ in Step 3. In total we pay

$$\begin{aligned} k + 1 + h(u) &= k + 1 + (1-2\epsilon)r(u) \\ &\le k + 1 + \frac{3(1-2\epsilon)}{4(1+\epsilon)^2}. \end{aligned}$$

In both cases it suffices to prove that

$$k + 1 + \frac{3(1-2\epsilon)}{4(1+\epsilon)^2} \le 3x.$$

Since $x = \frac{k}{1+\epsilon}$, this is equivalent to showing that

$$k + 1 + \frac{3(1-2\epsilon)}{4(1+\epsilon)^2} \le \frac{3k}{1+\epsilon}.$$

Claim 3.4. *For any $k \ge 1$, $0 \le \epsilon < \frac{1}{2}$, the following inequality holds:*

$$k + 1 + \frac{3(1-2\epsilon)}{4(1+\epsilon)^2} \le \frac{3k}{1+\epsilon}.$$

Proof. The claim is equivalent to

$$k\left(1 - \frac{3}{1+\epsilon}\right) + 1 + \frac{3(1-2\epsilon)}{4(1+\epsilon)^2} \le 0.$$

Note that $1 - \frac{3}{1+\epsilon} < 0$ for $\epsilon < \frac{1}{2}$. Therefore, the left-hand side is maximized when $k = 1$, and it is enough to prove that

$$2 - \frac{3}{1+\epsilon} + \frac{3(1-2\epsilon)}{4(1+\epsilon)^2} \le 0.$$

This is equivalent to

$$8\epsilon^2 - 2\epsilon - 1 \le 0,$$

which holds for all ϵ, $0 \le \epsilon \le \frac{1}{2}$. ∎

4. Hardness results.

4.1. Weighted vertex cover. We show that the capacitated vertex cover with arbitrary weights is at least as hard to approximate as the set cover problem. Given an instance of the set cover problem, let $G = (L, R, E')$ be its bipartite incidence graph, where $L = \mathcal{S}$, $R = E$, $(S, e) \in E'$ iff $e \in S$. For each vertex v in the graph, let $\delta(v)$ denote its degree. For each $v \in L$, define $w(v)$ to be the weight of the corresponding set, and $k(v) = \delta(v)$. For each $v \in R$, define $w(v) = 0$ and $k(v) = \delta(v) - 1$. For each vertex v in the graph, define the multiplicity to be $m(v) = 1$. Given a solution to the set cover instance, the solution to the capacitated vertex cover consists of all the vertices of R and the vertices from L corresponding to the sets in the set cover. The set vertices can cover all their adjacent edges. Since each element is covered in the set cover solution, for each $v \in R$, at least one of its adjacent edges is covered by a set vertex, and thus v has enough capacity to cover the remaining edges. The converse is also true. Given a feasible solution to the vertex cover problem, we can find a feasible solution to the set cover problem of the same cost. The solution to the set cover problem consists of the sets corresponding to the vertices of L that participate in the solution of the vertex cover instance.

4.2. Vertex cover with unsplittable demands. We assume that each edge e has a demand $d(e)$ that must be supplied by one of its endpoints. For each $v \in V$, the sum of the demands of the adjacent edges that v supplies must not exceed the capacity $k(v)$. It is impossible to approximate this problem, since, given a problem instance, it is NP-hard to answer the question of whether V (the set of all the vertices in the problem instance) is a feasible vertex cover, even if the demands are given in unary. The reduction is from the 3-partition problem, which is defined as follows. We are given a bound $B \in Z^+$ and a collection of $3m$ numbers, $a_1, \ldots, a_{3m}$, such that $\sum_{i=1}^{3m} a_i = mB$, and for each $i : 1 \le i \le 3m$, $B/4 < a_i < B/2$. The question is whether the numbers can be partitioned into m sets, such that the sum of numbers in each set is B. Note that if such a partition exists, each set will contain exactly three numbers. This problem is NP-hard in the strong sense (i.e., it is NP-hard even if the numbers are given in unary) [14].

The reduction proceeds as follows. We have $3m$ vertices $v_1, \ldots, v_{3m}$ representing the $3m$ numbers. The capacity of vertex v_i, $1 \le i \le 3m$, is $(m-1)a_i$. We also have m vertices $u_1, \ldots, u_m$ representing the sets, and the capacity of each such vertex is B. Thus, the set of vertices is $V = \{v_i \mid 1 \le i \le 3m\} \cup \{u_j \mid 1 \le j \le m\}$. For each v_i, u_j: $1 \le i \le 3m$, $1 \le j \le m$, there is an edge between the two vertices with demand a_i. Suppose there is a valid partition of the $3m$ numbers into m sets $S_1, \ldots, S_m$. Then for each vertex u_j, $1 \le j \le m$, and for each vertex v_i, $1 \le i \le 3m$, such that $a_i \in S_j$, vertex u_j covers the edge connecting u_j and a_i. As the sum of the numbers in each S_j is exactly B, the capacity of u_j is enough to cover all these edges. Now for each vertex v_i, $1 \le i \le 3m$, one of the edges adjacent to this vertex is covered by one of the vertices $u_1, \ldots, u_m$, and therefore the capacity of v_i is sufficient to cover the remaining $(m-1)$ edges.

The converse direction is also true. Suppose that the set V of vertices can cover all the edges. We show a valid partition of the input elements into m sets $S_1, \ldots, S_m$. Note that each vertex v_i, $1 \le i \le 3m$, has enough capacity to cover only $(m-1)$ of its adjacent edges. Therefore, at least one edge adjacent to v_i is covered by some u_j, $1 \le j \le m$. Set S_j contains all such elements a_i for which vertex u_j covers the edge that connects it with v_i. As the capacities of the vertices u_j, $1 \le j \le m$, are B, the elements in each set sum up also to at most B. Thus, a solution to the capacitated vertex cover problem defines a solution to the 3-partition problem.

5. Set cover with hard capacities. In this section we consider the set cover problem with hard capacities. For the sake of simplicity, we assume that for each set $S \in \mathcal{S}$, only one copy is available, i.e., $m(S) = 1$. If this is not the case, we can view each available copy of each set as a distinct set. Note that the input size remains polynomial, as at most n copies of each set are needed. Thus, given two families $\mathcal{A}, \mathcal{B} \subset \mathcal{S}$ of sets, their union is now defined as usual: $\mathcal{A} \cup \mathcal{B} = \{S \mid S \in \mathcal{A} \text{ or } S \in \mathcal{B}\}$.

We need the following notation. Let $\mathcal{T} \subseteq \mathcal{S}$ be a collection of sets, and let $f(\mathcal{T})$ denote the maximum number of elements that the sets belonging to $\mathcal{T}$ can cover without violating the capacity constraints. Note that $f(\mathcal{T})$ can be computed using Lemma 2.1. For $S \in \mathcal{S}$, define $f_{\mathcal{T}}(S) = f(\mathcal{T} \cup \{S\}) - f(\mathcal{T})$ (i.e., $f_{\mathcal{T}}(S)$ is the increase in the number of elements that can be covered when S is added to set family $\mathcal{T}$). Consider the following greedy algorithm for the set cover problem with hard capacities.

Algorithm Greedy Cover.

1. Initially, $\mathcal{P} = \emptyset$.
2. While $\mathcal{P}$ is not a feasible capacitated set cover:
   (a) Let $S = \arg\min_{S:\ f_{\mathcal{P}}(S)>0} \frac{w(S)}{f_{\mathcal{P}}(S)}$.
   (b) Add S to $\mathcal{P}$.

Wolsey [30] showed using the *dual fitting* technique that Algorithm Greedy Cover achieves an approximation factor of $O(\log(\max_S |S|))$. We show a simpler and a more intuitive charging scheme that proves the same result.

Let $\mathcal{T} \subseteq \mathcal{S}$ be a collection of sets, and let $C \subseteq \mathcal{T} \times E$ be a feasible partial cover. Denote by $|C|$ the number of elements covered by C. We can assume without loss of generality that no element is covered by more than one set in $\mathcal{T}$. For each $\mathcal{T}' \subseteq \mathcal{T}$, we denote by $C_{\mathcal{T}'}$ the projection of C on $\mathcal{T}'$, and by $f_C(\mathcal{T}')$ the number of elements covered by sets belonging to $\mathcal{T}'$ in C. We need the following lemma.

Lemma 5.1. *Consider an instance of the set cover problem with hard capacities. Let $\mathcal{T}$ be a feasible cover, and let $\mathcal{T}_1, \mathcal{T}_2$ be a partition of $\mathcal{T}$ into two disjoint subsets. Then, there is a feasible cover $C \subseteq \mathcal{T} \times E$ such that all the elements are covered in C and $f_C(\mathcal{T}_1) = f(\mathcal{T}_1)$.*

Proof. Let $C \subseteq \mathcal{T} \times E$ be a feasible cover, where each element $e \in E$ is covered by some $S \in \mathcal{T}$, and assume that $f_C(\mathcal{T}_1) < f(\mathcal{T}_1)$. Let $C' \subseteq \mathcal{T}_1 \times E$ be a feasible partial cover, where $f_{C'}(\mathcal{T}_1) = f(\mathcal{T}_1)$. Since C' is a partial cover, some elements may not be covered in C'. We gradually change the cover C, while maintaining its feasibility, until the lemma is satisfied. Perform the following procedure:

While $f_C(\mathcal{T}_1) < f(\mathcal{T}_1)$:

1. Let $S \in \mathcal{T}_1$ be a set such that S covers fewer elements in C than it does in C'. There is at least one such set, since $f_C(\mathcal{T}_1) < f(\mathcal{T}_1)$.
2. Let $e \in E$ be an element covered by S in C' but covered by some $T \neq S$ in C. (Note that T can belong to either $\mathcal{T}_1$ or $\mathcal{T}_2$.)
3. Change C so that e is covered by S; i.e., remove (T, e) and add (S, e) to C.

It is clear that we can perform the procedure and maintain a feasible cover C, while $f_C(\mathcal{T}_1) < f(\mathcal{T}_1)$. Once a pair $(S, e) \in C'$ is added to C, it remains there until the end of the procedure. Thus, the number of iterations is bounded by $|C'|$ and is therefore finite. Upon termination of the procedure, we have a cover C that satisfies the conditions of the lemma. ∎

We now proceed with analyzing Algorithm Greedy Cover. Denote the solution computed by Algorithm Greedy Cover by $\mathcal{P} = \{S_1, S_2, \ldots, S_k\}$, and assume that the sets are added to the solution by the algorithm in this order. For each i, $0 \le i \le k$, let $\mathcal{P}_i = \{S_1, S_2, \ldots, S_i\}$ be the solution at the end of iteration i. Let OPT be an optimal solution. We "replay" the algorithm, while charging the costs of the sets added to $\mathcal{P}$ by Algorithm Greedy Cover to the sets in OPT.

Start with $\mathcal{P}_0 = \emptyset$. For each $S \in OPT$, let $a_0(S)$ be the number of elements covered by S in OPT (assuming every element is covered by exactly one set in OPT). For each iteration i of Algorithm Greedy Cover, new values $a_i(S)$ of sets in $OPT \setminus \mathcal{P}_i$ are defined. The following invariant holds throughout the analysis: we can cover all the elements by the sets in $OPT \cup \mathcal{P}_i$, even if the capacities of sets $S \in OPT \setminus \mathcal{P}_i$ are restricted to be $a_i(S)$.

The invariant is clearly true for $\mathcal{P}_0$ and a_0. Consider iteration i of Algorithm Greedy Cover. We add set S_i to the solution. Since the invariant holds for $\mathcal{P}_{i-1}$, a_{i-1}, the collection of sets $\mathcal{P}_i \cup OPT$ is a feasible cover, even if we restrict the capacities of sets $S \in OPT \setminus \mathcal{P}_i$ to be $a_{i-1}(S)$. By Lemma 5.1, there is a feasible cover $C \subseteq (\mathcal{P}_i \cup OPT) \times E$, where the sets in $\mathcal{P}_i$ cover exactly $f(\mathcal{P}_i)$ elements and each set $S \in OPT \setminus \mathcal{P}_i$ covers at most $a_{i-1}(S)$ elements. For each $S \in OPT \setminus \mathcal{P}_i$, define $a_i(S)$ to be the number of elements covered by S in C. Note that $a_i(S) \le a_{i-1}(S)$.

If $S_i \in OPT$, we do not charge any sets for its cost, since OPT also pays for it. Otherwise, suppose $f_{\mathcal{P}_{i-1}}(S_i) = n_i$. The number of elements covered by sets in $OPT \setminus \mathcal{P}_i$ in C is $\sum_{S \in OPT \setminus \mathcal{P}_i} a_{i-1}(S) - n_i$. Therefore, $\sum_{S \in OPT \setminus \mathcal{P}_i} (a_{i-1}(S) - a_i(S)) = n_i$. We charge each $S \in OPT \setminus \mathcal{P}_i$ with $\frac{w(S_i)}{n_i} \cdot (a_{i-1}(S) - a_i(S))$. Note that the total cost charged to the sets in OPT in this iteration is exactly $w(S_i)$.

We now bound the cost charged to each $S \in OPT$. If $S \in \mathcal{P}$, let j denote the last iteration before S is added to $\mathcal{P}$ (i.e., S is added to $\mathcal{P}$ at iteration $j+1$). Otherwise, let j be the first iteration after which $a_j = 0$ (note that the equality holds for the last iteration). For each $i < j$, at the beginning of iteration i, $f_{\mathcal{P}_{i-1}}(S) \ge a_{i-1}(S)$. This follows from the way the value of $a_{i-1}(S)$ is determined. Since Algorithm Greedy Cover chooses a set other than S in this iteration, $\frac{w(S_i)}{n_i} \le \frac{w(S)}{a_{i-1}(S)}$. Therefore, the total value charged to S is

$$\sum_{i=1}^{j} (a_{i-1}(S) - a_i(S)) \frac{w(S_i)}{n_i} \le w(S) \sum_{i=1}^{j} \frac{(a_{i-1}(S) - a_i(S))}{a_{i-1}(S)}.$$

Observe that for each $i : 1 \le i \le j$, the term $\frac{a_{i-1}(S) - a_i(S)}{a_{i-1}(S)}$ can be written as

$$\sum_{\ell = a_i(S)+1}^{a_{i-1}(S)} \frac{1}{a_{i-1}(S)} \le \sum_{\ell = a_i(S)+1}^{a_{i-1}(S)} \frac{1}{\ell}.$$

Thus, the value charged to S is bounded by $w(S)H(|S|)$, and the total cost of the solution is at most $OPT(1 + \ln(\max_S |S|))$.

6. Extensions.

6.1. Submodular set cover. Recall the *submodular set cover* problem. Let f be an integer valued function defined over all subsets of a finite set E of elements.

Function f is called *nondecreasing* if $f(S) \leq f(T)$ for all $S \subseteq T \subseteq E$, and *submodular* if $f(S) + f(T) \geq f(S \cap T) + f(S \cup T)$ for all $S, T \subseteq E$. The input to the submodular set cover problem is a family $\mathcal{S}$ of subsets of E, together with a nonnegative cost function. There is a nondecreasing nonnegative integer valued submodular function f defined over all collections of the input sets. The goal is to find a minimum cost collection $\mathcal{P} \subset \mathcal{S}$ of sets such that $f(\mathcal{P}) = f(\mathcal{S})$.

It is not hard to show that the natural linear programs for the set cover problem with hard capacities, as well as the more general submodular set cover problem, have an unbounded integrality gap. We note that Algorithm Greedy Cover can be applied to the general submodular set cover problem as well. Indeed, Wolsey [30] showed using the dual fitting technique that Algorithm Greedy Cover achieves a $1 + O(\log(f_{\max}))$-approximation for this problem, where $f_{\max} = \max_{S \in \mathcal{S}} f(\{S\})$. For the sake of completeness, we describe Wolsey's linear program (SSC) below. As before, for $\mathcal{T} \subseteq \mathcal{S}$ and $S \in \mathcal{S}$, $f_{\mathcal{T}}(S) = f(\mathcal{T} \cup \{S\}) - f(\mathcal{T})$. For each set $S \in \mathcal{S}$, there is an indicator variable $x(S)$ showing whether S is in the solution. The goal is to minimize the solution cost, i.e., $\sum_{S \in \mathcal{S}} w(S)x(S)$.

Consider now some collection of input sets $\mathcal{T} \subset \mathcal{S}$, and suppose $f(\mathcal{S}) - f(\mathcal{T}) > 0$. Let $\mathcal{P}$ denote any feasible solution and $\mathcal{P}' = \mathcal{P} \setminus \mathcal{T}$. As $f(\mathcal{S}) - f(\mathcal{T}) > 0$, some sets in $\mathcal{S} \setminus \mathcal{T}$ must be in the solution; i.e., $\mathcal{P}'$ is nonempty. Moreover, $f(\mathcal{P}' \cup \mathcal{T}) - f(\mathcal{T}) \geq f(\mathcal{S}) - f(\mathcal{T})$. Note that, due to the submodularity of f, $\sum_{S \in \mathcal{P}'} (f(\{S\} \cup \mathcal{T}) - f(\mathcal{T})) \geq f(\mathcal{P}' \cup \mathcal{T}) - f(\mathcal{T})$, and therefore $\sum_{S \in \mathcal{P}'} f_{\mathcal{T}}(S) \geq f(\mathcal{S}) - f(\mathcal{T})$ must hold. This condition is expressed by the set of constraints (1).

$$
\begin{array}{lll}
\text{(SSC)} & \min & \displaystyle\sum_{S \in \mathcal{S}} w(S)x(S) \\
& \text{s.t.} & \\
\text{(1)} & \displaystyle\sum_{S \notin \mathcal{T}} f_{\mathcal{T}}(S)x(S) \geq f(\mathcal{S}) - f(\mathcal{T}) & \text{for all } \mathcal{T} \subseteq \mathcal{S}, \\
& x(S) \geq 0 & \text{for all } S \in \mathcal{S}.
\end{array}
$$

We show that our analysis of Algorithm Greedy Cover can be extended to prove a similar approximation guarantee for the submodular set cover problem.

Denote by $\mathcal{P} = \{S_1, S_2, \ldots, S_k\}$ the solution computed by Algorithm Greedy Cover, and assume that the sets are added to the solution by the algorithm in this order. For each i, $0 \leq i \leq k$, let $\mathcal{P}_i = \{S_1, S_2, \ldots, S_i\}$ be the solution at the end of iteration i. Let OPT be an optimal solution. Choose an arbitrary ordering of sets in OPT such that the sets in $\mathcal{P} \cap OPT$ appear at the beginning, in the same order in which they are added to the solution by Algorithm Greedy Cover. For each j, let $\mathcal{T}_j$ denote the first j sets in OPT.

Consider some iteration i of the algorithm. Let $S \in OPT$, and assume that S is the jth set in OPT. Define

$$a_i(S) = f(\mathcal{T}_j \cup \mathcal{P}_i) - f(\mathcal{T}_{j-1} \cup \mathcal{P}_i).$$

If $S_i \in OPT$, then we do not have to charge its cost to sets in OPT. Otherwise, each set $S \in OPT \setminus \mathcal{P}_i$ is charged with $\frac{w(S_i)}{f(\mathcal{P}_i) - f(\mathcal{P}_{i-1})}(a_{i-1}(S) - a_i(S))$.

Observe that the amount charged in iteration i to sets in $OPT \setminus S_i$ is at least $w(S_i)$. This is true, since

$$\begin{aligned}\sum_{S\in OPT\setminus\mathcal{P}_i}(a_{i-1}(S)-a_i(S)) &= \sum_j (f(\mathcal{T}_j\cup\mathcal{P}_{i-1})-f(\mathcal{T}_{j-1}\cup\mathcal{P}_{i-1}))\\ &\quad -\sum_j (f(\mathcal{T}_j\cup\mathcal{P}_i)-f(\mathcal{T}_{j-1}\cup\mathcal{P}_i))\\ &= f(OPT\cup\mathcal{P}_{i-1})-f(\mathcal{P}_{i-1})-f(OPT\cup\mathcal{P}_i)+f(\mathcal{P}_i)\\ &= f(OPT)-f(OPT)+f(\mathcal{P}_i)-f(\mathcal{P}_{i-1})\\ &= f(\mathcal{P}_i)-f(\mathcal{P}_{i-1}).\end{aligned}$$

Observe also that at the beginning of iteration $i+1$, for each $S\in OPT\setminus\mathcal{P}_i$, $f(\{S\})\geq a_i(S)$. Since $f(\{S\})\geq f(\{S\}\cup\mathcal{P}_i)-f(\mathcal{P}_i)$ (due to the submodularity of f), it is enough to show that

$$f(\{S\}\cup\mathcal{P}_i)-f(\mathcal{P}_i)\geq f(\mathcal{T}_j\cup\mathcal{P}_i)-f(\mathcal{T}_{j-1}\cup\mathcal{P}_i).$$

Rearranging the sides, this is equivalent to

$$f(\{S\}\cup\mathcal{P}_i)+f(\mathcal{T}_{j-1}\cup\mathcal{P}_i)\geq f(\mathcal{T}_j\cup\mathcal{P}_i)+f(\mathcal{P}_i),$$

which holds by the submodularity of f.

Using the same reasoning as in the case of the proof of the set cover with hard capacities, the total amount charged to any $S\in OPT$ is at most $w(S)H(|S|)$, and the total cost of the solution is bounded by $OPT(1+\ln(\max_S f(\{S\})))$.

6.2. Multiset multicover. An interesting special case of the submodular set cover problem is the multiset multicover problem. In this problem, the input sets are actually multisets; i.e., an element $e\in E$ can appear in $S_j\in\mathcal{S}$ more than once, and the elements have splittable demands. An integer programming formulation of the multiset multicover problem with unbounded set capacities is the following: $\min\{w^Tx \mid Ax\geq d, 0\leq x\leq b, x\in Z\}$. The constraints $x\leq b$ are called multiplicity constraints, and they generally make covering problems much harder, as the natural linear programming relaxation has an unbounded integrality gap. Dobson [10] gives a combinatorial greedy $H(\max_{1\leq j\leq m}\sum_{1\leq i\leq n}A_{ij})$-approximation algorithm, where $H(t)$ is the tth harmonic number. This is a logarithmic approximation factor for the case where A is a $\{0,1\}$ matrix (set multicover), but can be as bad as a polynomial approximation bound in the general case (multiset multicover). Recently, Carr et al. [5] gave a p-approximation algorithm, where p denotes the maximum number of variables in any constraint. Their algorithm is based on a linear relaxation in the spirit of (SSC). Using similar ideas for strengthening the linear program, Kolliopoulos and Young [21] obtained an $O(\log n)$-approximation.

We can assume again that the multiplicities of the sets are unit, by viewing each copy of each set as a distinct set. We can then define, for each collection $\mathcal{T}$ of input sets, $f(\mathcal{T})$ to be the maximum number of elements that can be covered by $\mathcal{T}$ (with the capacity constraints). It is not hard to see that f is a nonnegative nondecreasing submodular function, and thus the algorithm of Wolsey and our analysis hold for this case.

Notice that the number of sets now is not necessarily polynomial, as the initial set multiplicities $m(S)$ for $S\in\mathcal{S}$ are not necessarily polynomial in the input size. However, the function $f(\mathcal{T})$ can still be computed in polynomial time. Thus, Algorithm Greedy Cover can be implemented to run in polynomial time, achieving an approximation ratio of $O(\log(\max_{S\in\mathcal{S}}|S|))$.

Acknowledgments. We would like to thank the anonymous referee for pointing out the reduction in section 4.2.

REFERENCES

[1] E. Balas and M. W. Padberg, *Set partitioning: A survey*, SIAM Rev., 18 (1976), pp. 710–760.
[2] J. Bar-Ilan, G. Kortsarz, and D. Peleg, *Generalized submodular cover problems and applications*, in Proceedings of the 4th Israel Symposium on Theory of Computing and Systems, Jerusalem, Israel, 1996, IEEE Computer Society Press, Piscataway, NJ, 1996, pp. 110–118.
[3] R. Bar-Yehuda and S. Even, *A local-ratio theorem for approximating the weighted vertex cover problem*, Ann. Discrete Math., 25 (1985), pp. 27–45.
[4] Y. Bartal, M. Charikar, and D. Raz, *Approximating min-sum k-clustering in metric spaces*, in Proceedings of 33rd ACM Symposium on Theory of Computing, Heraklion, Crete, Greece, 2001, ACM, New York, 2001, pp. 11–20.
[5] R. D. Carr, L. K. Fleischer, V. J. Leung, and C. A. Phillips, *Strengthening integrality gaps for capacitated network design and covering problems*, in Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, 2000, SIAM, Philadelphia, 2000, pp. 106–115.
[6] M. Charikar, S. Guha, E. Tardos, and D. Shmoys, *A constant-factor approximation algorithm for the k-median problem*, in Proceedings of the 31st Annual ACM Symposium on the Theory of Computing, Atlanta, GA, 1999, ACM, New York, 1999, pp. 1–10.
[7] N. Christofides and S. Korman, *A computational survey of methods for the set covering problem*, Management Sci., 21 (1975), pp. 591–599.
[8] J. Chuzhoy and Y. Rabani, *Approximating k-median with non-uniform capacities*, in Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms, Vancouver, BC, Canada, 2005, SIAM, Philadelphia, 2005, pp. 952–958.
[9] V. Chvátal, *A greedy heuristic for the set-covering problem*, Math. Oper. Res., 4 (1979), pp. 233–235.
[10] G. Dobson, *Worst-case analysis of greedy heuristics for integer programming with non-negative data*, Math. Oper. Res., 7 (1972), pp. 515–531.
[11] U. Feige, *A threshold of* $\ln n$ *for approximating set cover*, J. ACM, 45 (1998), pp. 634–652.
[12] R. Gandhi, E. Halperin, S. Khuller, G. Kortsarz, and A. Srinivasan, *An improved approximation algorithm for vertex cover with hard capacities*, in Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP), Eindhoven, The Netherlands, 2003, Lecture Notes in Comput. Sci. 2719, Springer, New York, 2003, pp. 164–175.
[13] R. Gandhi, S. Khuller, S. Parthasarathy, and A. Srinivasan, *Dependent rounding in bipartite graphs*, in Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, Vancouver, BC, Canada, 2002, IEEE Press, Piscataway, NJ, pp. 323–332.
[14] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco, 1979.
[15] R. S. Garfinkel and G. L. Nemhauser, *Optimal set covering: A survey*, in Perspectives on Optimization: A Collection of Expository Articles, A. M. Geofrion, ed., Addison-Wesley, Reading, MA, 1972, pp. 164–193.
[16] N. Garg, R. Khandekar, and V. Pandit, *Improved approximation for universal facility location*, in Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms, Vancouver, BC, Canada, 2005, SIAM, Philadelphia, 2005, pp. 959–960.
[17] S. Guha, R. Hassin, S. Khuller, and E. Or, *Capacitated vertex covering with applications*, in Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, 2002, SIAM, Philadelphia, 2002, pp. 858–865.
[18] D. S. Hochbaum, *Approximation algorithms for the set covering and vertex cover problems*, SIAM J. Comput., 11 (1982), pp. 555–556.
[19] D. S. Hochbaum, ed., *Approximation Algorithms for NP-Hard Problems*, PWS Publishing, Toronto, 1996.
[20] D. S. Johnson, *Approximation algorithms for combinatorial problems*, J. Comput. System Sci., 9 (1974), pp. 256–278.
[21] S. G. Kolliopoulos and N. E. Young, *Tight approximation results for general covering integer programs*, in Proceedings of the 42nd Annual Symposium on Foundations of Computer Science, Las Vegas, NV, 2001, IEEE Press, Piscataway, NJ, pp. 522–528.
[22] L. Lovász, *On the ratio of optimal and fractional covers*, Discrete Math., 13 (1975), pp. 383–390.
[23] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, J. ACM, 41 (1994), pp. 960–981.

[24] M. Mahdian and M. Pál, *Universal facility location*, in Proceedings of the 11th Annual European Symposium on Algorithms, Budapest, Hungary, 2003, Lecture Notes in Comput. Sci. 2832, Springer, New York, 2003, pp. 409–422.
[25] P. B. Mirchandani and R. L. Francis, eds., *Discrete Location Theory*, Wiley-Interscience, New York, 1990.
[26] M. W. Padberg, *Covering, packing and knapsack problems*, Ann. Discrete Math., 4 (1979), pp. 265–287.
[27] D. B. Shmoys, É. Tardos, and K. Aardal, *Approximation algorithms for the facility location problem*, in Proceedings of the 29th Annual ACM Symposium on the Theory of Computing, El Paso, TX, 1997, ACM, New York, 1997, pp. 265–274.
[28] M. Pál, É. Tardos, and T. Wexler, *Facility location with nonuniform hard capacities*, in Proceedings of the 42nd Annual Symposium on Foundations of Computer Science, Las Vegas, NV, 2001, IEEE Press, Piscataway, NJ, pp. 329–338.
[29] V. V. Vazirani, *Approximation Algorithms*, Springer-Verlag, New York, 2001.
[30] L. A. Wolsey, *An analysis of the greedy algorithm for the submodular set covering problem*, Combinatorica, 2 (1982), pp. 385–393.
[31] J. Zhang, B. Chen, and Y. Ye, *A multi-exchange local search algorithm for the capacitated facility location problem*, Math. Oper. Res., 30 (2005), pp. 389–403.

SIAM J. COMPUT.
Vol. 36, No. 2, pp. 516–542

PROPERTIES OF NP-COMPLETE SETS*

CHRISTIAN GLAßER†, A. PAVAN‡, ALAN L. SELMAN§, AND SAMIK SENGUPTA§

Abstract. We study several properties of sets that are complete for NP. We prove that if L is an NP-complete set and $S \not\supseteq L$ is a p-selective sparse set, then $L - S$ is $\leq_m^p$-hard for NP. We demonstrate the existence of a sparse set $S \in \mathrm{DTIME}(2^{2^n})$ such that for every $L \in \mathrm{NP} - \mathrm{P}$, $L - S$ is not $\leq_m^p$-hard for NP. Moreover, we prove for every $L \in \mathrm{NP} - \mathrm{P}$ that there exists a sparse $S \in \mathrm{EXP}$ such that $L - S$ is not $\leq_m^p$-hard for NP. Hence, removing sparse information in P from a complete set leaves the set complete, while removing sparse information in EXP from a complete set may destroy its completeness. Previously, these properties were known only for exponential time complexity classes. We use hypotheses about pseudorandom generators and secure one-way permutations to derive consequences for longstanding open questions about whether NP-complete sets are immune. For example, assuming that pseudorandom generators and secure one-way permutations exist, it follows easily that NP-complete sets are not p-immune. Assuming only that secure one-way permutations exist, we prove that no NP-complete set is $\mathrm{DTIME}(2^{n^\epsilon})$-immune. Also, using these hypotheses we show that no NP-complete set is quasi-polynomial-close to P. We introduce a strong but reasonable hypothesis and infer from it that disjoint Turing-complete sets for NP are not closed under union. Our hypothesis asserts the existence of a UP-machine M that accepts 0^* such that for some $0 < \epsilon < 1$, no 2^{n^ϵ} time-bounded machine can correctly compute infinitely many accepting computations of M. We show that if $\mathrm{UP} \cap \mathrm{coUP}$ contains $\mathrm{DTIME}(2^{n^\epsilon})$-bi-immune sets, then this hypothesis is true.

Key words. NP-completeness, robustness, immunity, one-way permutations, disjoint unions

AMS subject classification. 68Q15

DOI. 10.1137/S009753970444421X

1. Introduction.

This paper continues the long tradition of investigating the structure of complete sets under various kinds of reductions. Concerning the most interesting complexity class, NP, almost every question has remained open. While researchers have always been interested primarily in the structure of complete sets for NP, for the most part, success, where there has been any, has come from studying the exponential time classes. In this paper we focus entirely on the complexity class NP.

The first topic we study concerns the question, How robust are complete sets? Schöning [Sch86] raised the following question: If a small amount of information is removed from a complete set, does the set remain hard? Tang, Fu, and Liu [TFL93] proved the existence of a sparse set S such that for every $\leq_m^p$-complete set L for EXP, $L - S$ is not hard. Their proof depends on the fact that for any exponential time computable set B and any exponential time complete set A, there exists a length-increasing, one-one reduction from B to A [Ber76]. We do not know that about NP. Buhrman, Hoene, and Torenvliet [BHT98] proved that $L - S$ still remains hard for EXP if S is any p-selective sparse set.

*Received by the editors June 1, 2004; accepted for publication (in revised form) April 11, 2006; published electronically July 31, 2006. A preliminary version of this paper appeared in Proceedings of the 19th IEEE Conference on Computational Complexity, 2004.

http://www.siam.org/journals/sicomp/36-2/44421.html

†Lehrstuhl für Informatik IV, Universität Würzburg, 97074 Würzburg, Germany (glasser@informatik.uni-wuerzburg.de).

‡Department of Computer Science, Iowa State University, Ames, IA 50011 (pavan@cs.iastate.edu). The research of this author was supported in part by NSF grants CCR-0344187 and CCF-0430807.

§Department of Computer Science and Engineering, 201 Bell Hall, University at Buffalo, Buffalo, NY 14260 (selman@cse.buffalo.edu, samik@cse.buffalo.edu). The research of the third author was partially supported by NSF grant CCR-0307077.

Here we prove these results unconditionally for sets that are NP-complete. We prove that if L is an NP-complete set and $S \not\supseteq L$ is a p-selective sparse set, then $L - S$ is $\leq_m^p$-hard for NP. We use the left-set technique of Ogiwara and Watanabe [OW91] to prove this result, and we use this technique elsewhere in the paper also. We demonstrate the existence of a sparse set $S \in \mathrm{DTIME}(2^{2^n})$ such that for every $L \in \mathrm{NP} - \mathrm{P}$, $L - S$ is not $\leq_m^p$-hard for NP. Moreover, we prove for every $L \in \mathrm{NP} - \mathrm{P}$ that there exists a sparse $S \in \mathrm{EXP}$ such that $L - S$ is not $\leq_m^p$-hard for NP. Hence, removing sparse information in P from a complete set leaves the set complete, while removing sparse information in EXP from a complete set may destroy its completeness.

In the fourth section of this paper we build on results of Agrawal [Agr02], who demonstrated that pseudorandom generators can be used to prove structural theorems on complete degrees. We use hypotheses about pseudorandom generators to answer the longstanding open question of whether NP-complete sets can be immune. Assuming the existence of pseudorandom generators and secure one-way permutations, we prove easily that no NP-complete set is p-immune. (This too is a well-known property of the EXP-complete sets.) Assuming only that secure one-way permutations exist, we prove that no NP-complete set is $\mathrm{DTIME}(2^{n^\epsilon})$-immune. Also, we use this hypothesis to show that no NP-complete set is quasi-polynomial-close to P. It is already known [Ogi91, Fu93] that no NP-complete set is p-close to a set in P unless P = NP.

The fifth section studies the question of whether the union of disjoint Turing-complete sets for NP is Turing-complete. Here is the background. If A and B are two disjoint computably enumerable (c.e.) sets, then $A \leq_T A \cup B$, $B \leq_T A \cup B$, and it follows that if either A or B is Turing-complete for the c.e. sets, then so is $A \cup B$ [Sho76]. The proofs are straightforward: To demonstrate that $A \leq_T A \cup B$, on input x, ask whether $x \in A \cup B$. If not, then $x \notin A$. Otherwise, simultaneously enumerate A and B until x is output. The proof suggests that these properties may not hold for $\leq_T^p$-complete sets for NP. In particular Selman [Sel88] raised the question of whether the union of two disjoint $\leq_T^p$-complete sets for NP is $\leq_T^p$-complete. It is unlikely that $A \leq_T^p A \cup B$ for every two disjoint sets A and B in NP; if this holds, then $\mathrm{NP} \cap \mathrm{coNP} = \mathrm{P}$: Take $A \in \mathrm{NP} \cap \mathrm{coNP}$; then $\overline{A} \in \mathrm{NP} \cap \mathrm{coNP}$ as well, and $A \leq_T^p (A \cup \overline{A}) \implies A \in \mathrm{P}$.

First, we will prove that if UEE $\neq$ EE, then there exist two disjoint languages A and B in NP such that $A \not\leq_T^p A \cup B$. Second, we introduce the following reasonable but strong hypothesis: There is a UP-machine M that accepts 0^* such that for some $0 < \epsilon < 1$, no 2^{n^ϵ} time-bounded machine can correctly compute infinitely many accepting computations of M. This hypothesis is similar to hypotheses used in several earlier papers [FFNR96, HRW97, FPS01, PS01]. We prove, assuming this hypothesis, that there exist disjoint Turing-complete sets for NP whose union is not Turing-complete. Also, we show that if UP $\cap$ coUP contains $\mathrm{DTIME}(2^{n^\epsilon})$-bi-immune sets, then this hypothesis is true. Finally, we make several observations about the question of whether the union of two disjoint NP-complete sets is NP-complete. It would be difficult to obtain results about these questions without introducing hypotheses about complexity classes, because there are oracles relative to which the answers to these questions are both positive and negative. Proofs that would settle these questions would not relativize to all oracles.

2. Preliminaries. We use standard notation and assume familiarity with standard resource-bounded reducibilities. Given a complexity class $\mathcal{C}$ and a reducibility $\leq_r$, a set A is $\leq_r$-*hard* for $\mathcal{C}$ if for every set $L \in \mathcal{C}$, $L \leq_r A$. The set A is $\leq_r$-*complete* if, in addition, $A \in \mathcal{C}$. We use the phrase "NP-complete" to mean $\leq_m^p$-complete for NP.

A set S is *sparse* if there exists a polynomial p such that for all positive integers n, $\|S \cap \Sigma^n\| \leq p(n)$. We use polynomial-time invertible pairing functions $\langle \cdot, \cdot \rangle : \Sigma^* \times \Sigma^* \to \Sigma^*$.

A set S is *p-selective* [Sel79] if there is a polynomial-time-computable function $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that for all words x and y, (i) $f(x, y) = x$ or $f(x, y) = y$ and (ii) $x \in A$ or $y \in A$ implies $f(x, y) \in A$.

A set L is *immune* to a complexity class $\mathcal{C}$, or *$\mathcal{C}$-immune*, if L is infinite and no infinite subset of L belongs to C. A set L is *bi-immune* to a complexity class $\mathcal{C}$, or *$\mathcal{C}$-bi-immune*, if both L and $\overline{L}$ are $\mathcal{C}$-immune.

3. Robustness. In this section we consider the following question: If L is NP-complete and S is a sparse set, then does $L - S$ remain complete? This question was studied for exponential time complexity classes by Tang, Fu, and Liu [TFL93] and by Buhrman, Hoene, and Torenvliet [BHT98]. The basic result [TFL93] is that there exists a subexponential-time computable sparse set S such that for every $\leq_m^p$-complete set L for EXP, $L - S$ is not EXP-complete. On the other hand, for any p-selective sparse set S, $L - S$ still remains hard [BHT98]. The theorems of Tang, Fu, and Liu depend on the fact that for any exponential-time computable set B and any exponential-time complete set A, there exists a length-increasing, one-one reduction from B to A [BH77]. We do not know that about NP. Nevertheless, here we prove the analogues of these results for NP. Observe that our first result, Theorem 3.1, holds unconditionally.

THEOREM 3.1. *Let L be an* NP*-complete set, and let S be a p-selective sparse set such that $L \not\subseteq S$. Then $L - S$ is $\leq_m^p$-hard for* NP.

Proof. Note that $L - S \neq \emptyset$. If $L - S$ is finite, then L is sparse as well. Since L is $\leq_m^p$-complete for NP, NP = P [Mah82]. Therefore, $L - S$ is also NP-complete. So we assume that $L - S$ is infinite in the rest of the proof.

We use the left set technique of Ogiwara and Watanabe [OW91]. Assume that M is a nondeterministic machine that accepts L. Let T_x be the computation tree of M on any string x. Without loss of generality, assume that T_x is a complete binary tree, and let d be the depth of T_x. Given two nodes u and v in T_x, we say that $u < v$ if the path from the root to u lies to the left of the path from the root to v, and $u \leq v$ if either $u < v$ or u lies on the path from the root to v. Let

$$\begin{aligned}\text{Left}(L) = \{\langle x, u\rangle \mid \exists v, u \leq v, u, v \in T_x, \text{ an accepting} \\ \text{computation of } M \text{ on } x \text{ passes through } v\}.\end{aligned}$$

Since L is NP-complete and Left(L) is in NP, Left$(L) \leq_m^p L$ via some $f \in$ PF. When it is understood that $v \in T_x$, we will write v as an abbreviation for $\langle x, v\rangle$ and $f(v)$ as an abbreviation for $f(\langle x, v\rangle)$. Given x of length n, the length of every node of T_x is bounded by a polynomial in n. Since f is polynomial-time computable, the length of $f(v)$, where $v \in T_x$, is bounded by $p(n)$ for some polynomial $p(\cdot)$. We call $f(v)$ the *label* of v. Since S is sparse, there is a polynomial bound $q(n)$ on the number of strings in S of length at most $p(n)$. Let $g(\cdot, \cdot)$ be the selector function for S. Consider the following total preorder [Tod91] on some $Q \subseteq \Sigma^{\leq p(n)}$:

$$\begin{aligned}x \leq_g y \iff & \exists z_1, z_2, \ldots, z_m \in Q, \\ & g(x, z_1) = x, g(z_1, z_2) = z_1, \ldots, \\ & g(z_{m-1}, z_m) = z_{m-1}, g(z_m, y) = z_m.\end{aligned}$$

Observe that if $x \leq_g y$ and $y \in S$, then $x \in S$ also. Given the selector g, the strings in Q can be ordered by $\leq_g$ in time polynomial in the sum of the lengths of

the strings in Q. Therefore, if $\|Q\|$ is polynomial in n, then the strings in Q can be ordered by $\leq_g$ in time polynomial in n as well.

We first make a few simple observations.

Observation 1. If $u < v$, and w is a descendant of u, then $w < v$.

Observation 2. Let v be the left most node of T_x at some level. Then

$$x \in L \Leftrightarrow v \in \text{Left}(L) \Leftrightarrow f(v) \in L.$$

Observation 3. Let $X = \{x_1, x_2, \dots\} \subseteq \Sigma^{\leq p(n)}$ be a set of more than $q(n)$ distinct strings. Then there exists a procedure that runs in time polynomial in n and outputs $x_i \notin S, i \leq q(n) + 1$.

Proof. Order the first $q(n) + 1$ strings in X by $\leq_g$ and output a highest string as x_i. Since there can be at most $q(n)$ strings of length $\leq p(n)$ in S, x_i cannot be in S. □

We now define a reduction from L to $L - S$. Before we give the formal algorithm, we present the idea behind the reduction. At any time during the reduction our reduction maintains a list of nodes in T_x and a node called *special*. Given x, we perform a breadth first search on T_x. If at any level of the search we find two nodes u and v such that $f(u) = f(v)$, then we do an Ogiwara–Watanabe pruning. If at any stage more than $q(n)$ nodes remain after the pruning, then we can find the left most node *special* such that $f(special)$ is not S. The list contains all the nodes that are to the left of *special*. At this point, we would like stop the search and output $f(special)$. However, it is possible that the rightmost accepting computation passes through a node in the list and so *special* is not in $Left(L)$; thus the reduction is wrong. Thus before we output $f(special)$, we have to verify that the rightmost accepting computation does not pass through a node in the list. However, it is not clear that this can be done in polynomial time.

To get around this, we further expand the nodes in the list and do an Ogiwara–Watanabe pruning again. This process may redefine the node *special*. We stop the search when every node in the list is a leaf node. If any of the leaf nodes is an accepting node, then we know that x is in L, and we output a fixed string not in $L - S$. Otherwise, we output $f(special)$ and argue that the rightmost accepting computation (if it exists) must either pass through *special* or it lies to the right of it.

Now we give a formal description of the reduction.

On input x, $|x| = n$, the reduction traverses T_x in stages. During stage k, the reduction maintains a list $\texttt{list}_k$ of nodes in T_x at level k. The reduction procedure has a variable called "`special`" which holds some node of T_x. At stage 1, $\texttt{list}_1$ contains the root of T_x, and the value of `special` is undefined. Now we define stage $k > 1$.

Step 1 Let $\texttt{list}_{k-1} = \langle v_1, v_2, \dots, v_t \rangle$.

Step 2 Let $u'_1 < u'_2 < \cdots < u'_{2t}$ be the children of nodes in $\texttt{list}_{k-1}$. This ordering is possible since all nodes in $\texttt{list}_{k-1}$ are at depth $k - 1$ of the tree T_x, and therefore $u'_1, \dots, u'_t$ are at level k. Put all these nodes in $\texttt{list}_k$.

Step 3: Pruning If there exist two nodes u'_i and u'_l, $i < l$, in $\texttt{list}_k$ such that $f(u'_i) = f(u'_l)$, then remove $u'_i, \dots, u'_{l-1}$ from $\texttt{list}_k$. Now let $u_1 < \cdots < u_m$ be the nodes in $\texttt{list}_k$, where every u_i has distinct labels. If $m \leq q(n)$, go to the next stage.

Step 4 It must be the case that $m > q(n)$. Therefore, by Observation 3, there must be some $j \leq q(n) + 1$ such that $f(u_j) \notin S$. Set $\texttt{special} = u_j$.

Step 5 If `special` is the leftmost node of T_x at level k, then output `special` and halt.

Step 6 Otherwise, place $u_1, \ldots, u_{j-1}$ in $\texttt{list}_k$ and go to the next stage.

The following algorithm h defines the reduction from L to $L - S$:

```
for k = 1 to d
    run stage k
if any stage halts and outputs v, then
    output f(v)
else /* list_d contains some leaf nodes of T_x */
    if any of the leaf nodes is an accepting computation
        of M on x, then output a predetermined fixed
        string w ∈ L − S
    else
        output f(special)
    endif
endif
```

We prove that the above reduction is correct by the following series of claims.

CLAIM 3.2. *For any $k < d$, if stage k outputs a string v, then*

$$x \in L \Leftrightarrow f(v) \in L - S.$$

Proof. If stage k outputs v, then v is the leftmost node of T_x at level k and $f(v) \notin S$. By Observation 2, the claim follows. □

From now assume that for no k, stage k halts in Step 5. First we make some observations.

Observation 4. During stage $k \geq 1$, $\|\texttt{list}_k\| \leq q(n)$.

Proof. For any stage k, assume that $\texttt{list}_{k-1}$ has $t \leq q(n)$ nodes. The number of nodes in $\texttt{list}_k$ before pruning is at most $2t$. After the pruning step, every $v \in \texttt{list}_k$ has a different label. If there are $\leq q(n)$ nodes in $\texttt{list}_k$, then the procedure goes to the next stage. Otherwise, the node u_j, where $j \leq q(n) + 1$, has a label outside S. Since we assume that stage k does not halt in Step 5, the procedure goes to stage k with $\|\texttt{list}_k\| = j - 1 \leq q(n)$. □

Observation 5. Suppose $\texttt{special} = v$ at the end of stage k. Then for $l \geq k$, for all $u \in \texttt{list}_l$, $u < v$.

Proof. At the end of stage k, let $v = \texttt{special} = u_j$. After Step 6, $\texttt{list}_k$ is a subset of $\{u_1, \ldots, u_{j-1}\}$. Thus for all $u \in \texttt{list}_k$, $u < v$. Note that in any subsequent stage $l > k$, the nodes that belong to $\texttt{list}_l$ are the descendants of nodes in $\texttt{list}_k$. By Observation 1, we obtain the proof. □

Observation 6. No node that is pruned in Step 3 can be on the path containing the rightmost accepting computation.

Proof. If $x \notin L$, no node in T_x is on the path containing any accepting computation. Therefore, let us assume that $x \in L$. If two nodes u'_i and u'_l at the same depth have the identical label w, then $f(u'_i) \in \text{Left}(L) \iff f(u'_l) \in \text{Left}(L)$. Therefore, if any u'_k at the same depth is on the path of the rightmost accepting computation, then either $k < i$ or $k \geq l$. Since only the nodes $u'_i, \ldots, u'_{l-1}$ are pruned, u'_k cannot be pruned. □

CLAIM 3.3. *Assume that $x \in L$ and stage $k \geq 1$ does not halt in Step 5. If $\exists v \in \texttt{list}_k$ that is on the path containing the rightmost accepting computation, then either $\exists u \in \texttt{list}_{k+1}$ that is on the path containing the rightmost accepting computation or* $\texttt{special} \in \text{Left}(L)$.

Proof. Since there is a node v in $\texttt{list}_k$ that is on the path containing the rightmost accepting computation, let u'_r be the node that is generated at Step 2 of stage $k+1$ that

is on the path containing the rightmost accepting computation. By Observation 6, u'_r cannot get pruned in Step 3, and therefore it is in $\texttt{list}_k$ at Step 4. Let us denote this node by u_r. If a node u_j is assigned `special` in Step 4, then either $j \leq r$, in which case $\texttt{special} \in \text{Left}(L)$, or $r < j$, and therefore u_r is in $\texttt{list}_{k+1}$ after Step 6. □

CLAIM 3.4. *If, for every k, stage k does not halt in Step 5, then $x \in L$ if and only if $\texttt{list}_d$ contains a leaf node that is an accepting computation or* $\texttt{special} \in \text{Left}(L)$.

Proof. Note that if x is not in L, then no leaf node can be accepting, and no node of T_x can be in $\text{Left}(L)$. Therefore, the *if* direction is trivial. We show the *only if* direction. We prove the following by induction on the number of stages: If $x \in L$, then after stage k, either the rightmost accepting computation passes through a node in $\texttt{list}_k$ or $\texttt{special} \in \text{Left}(L)$.

After stage 1, $\texttt{list}_1$ contains the root of the tree. Thus the claim is true after stage 1. Assume that the claim is true after stage $k-1$. Thus either the rightmost accepting computation passes through a node in $\texttt{list}_{k-1}$ or $\texttt{special} \in \text{Left}(L)$. We consider two cases.

Case 1. The rightmost accepting computation passes through a node in $\texttt{list}_{k-1}$. By Claim 3.3, either there is a node in $\texttt{list}_k$ that is on the path of the rightmost accepting computation, or the node that is assigned `special` during stage k is in $\text{Left}(L)$.

Case 2. $\texttt{special} \in \text{Left}(L)$. Let s be the node that is currently assigned to `special`. It suffices to show that if a node u is assigned to `special` at stage k, then u will also be in $\text{Left}(L)$. By Observation 5, for every node $v \in \texttt{list}_{k-1}$, $v < s$. Since u is a descendant of some node v in $\texttt{list}_{k-1}$, $u < s$ as well. Therefore, $s \in \text{Left}(L) \implies u \in \text{Left}(L)$.

Therefore, after stage k, $k \geq 1$, the rightmost accepting computation of M either passes through a node in $\texttt{list}_k$ or $\texttt{special} \in \text{Left}(L)$. When $k = d$, this implies that either the rightmost accepting computation is a node in $\texttt{list}_d$ or $\texttt{special} \in \text{Left}(L)$. This completes the proof. □

The correctness of the reduction now follows.

CLAIM 3.5. *The reduction $h(\cdot)$ is correct, and it runs in polynomial time.*

Proof. If the reduction halts at Step 5 during any stage, then by Claim 3.2 $x \in L \Leftrightarrow h(x) \in L - S$. Assume that no stage halts in Step 5. Assume $x \in L$. By Claim 3.4, either $\texttt{list}_d$ contains an accepting leaf or $\texttt{special} \in \text{Left}(L)$. If $\texttt{list}_d$ contains an accepting computation, then $h(x) = w \in L - S$. Otherwise, if $\texttt{special} \in \text{Left}(L)$, then $f(\texttt{special}) \in L$. However, by the definition of `special`, $f(\texttt{special}) \notin S$. Therefore, $f(\texttt{special}) \in L - S$. On the other hand, if $x \notin L$, then no node of T_x can be in $\text{Left}(L)$, and so, in particular, $\texttt{special} \notin \text{Left}(L)$. Therefore, $h(x) = f(\texttt{special}) \notin L$.

By Observation 4, the number of nodes in $\texttt{list}_k$ for any $k \geq 1$ is bounded by $q(n)$. Therefore, the number of nodes visited by the reduction is at most $d \times 2q(n)$. Since d is bounded above by the running time of M on x, the total time required by the reduction is at most polynomial in n. □

Therefore, $L \leq^p_m L - S$. So $L - S$ is $\leq^p_m$-hard for NP. □

COROLLARY 3.6. *Let L be a $\leq^p_m$-complete set for* NP*, and let $S \in$ P be sparse. Then $L - S$ is $\leq^p_m$-complete for* NP.

Observe that the reduction h that is constructed in the proof of Theorem 3.1 actually satisfies the following property:

$$x \in L \implies h(x) \in L - S,$$
$$x \notin L \implies h(x) \in \overline{L}.$$

So for any $S' \subseteq S$ it holds that $L \leq_m^p L - S'$ via h. This shows the following generalization of Theorem 3.1.

THEOREM 3.7. *Let L be a $\leq_m^p$-complete set of* NP*, and let S be a subset of a sparse p-selective set. Then $L - S$ is $\leq_m^p$-hard for* NP.

In contrast to the theorem we just proved, in Theorem 3.9, we construct a sparse set $S \in \mathrm{DTIME}(2^{2^n})$ such that for any set $L \in \mathrm{NP} - \mathrm{P}$, $L - S$ is not $\leq_m^p$-hard for NP. Again, as in Theorem 3.1, we cannot assert that $L - S \in \mathrm{NP}$. In Corollary 3.10, we obtain that for every $L \in \mathrm{NP} - \mathrm{P}$, there is a sparse $S \in \mathrm{EXP}$ such that $L - S$ is not $\leq_m^p$-hard for NP.

The following lemma shows a collapse to P for a restricted form of truth-table reduction from $\overline{\mathrm{SAT}}$ to a sublogarithmically dense set. In other words, we show that if $\overline{\mathrm{SAT}}$ disjunctively reduces to some sublogarithmically dense set where the reduction machine makes logarithmically many nonadaptive queries, then NP = P. We exploit this strong consequence in Theorem 3.9.

LEMMA 3.8. *If there exist $f \in \mathrm{FP}$, $S \subseteq \Sigma^*$, and a real number $\alpha < 1$ such that*

1. *for all $n \geq 0$, $\|S^{\leq n}\| \leq O(\log^\alpha n)$ and*
2. *for all x, $f(x)$ is a set of words such that $\|f(x)\| \leq O(\log |x|)$ and*

$$x \in \mathrm{SAT} \Leftrightarrow f(x) \cap S = \emptyset,$$

then P = NP.

Proof. Assume f, S, and α exist. Let

$$\mathrm{LeftSAT} \stackrel{df}{=} \{\langle x, z\rangle \mid \text{formula } x \text{ has a satisfying assignment } y \geq z\}.$$

Note that for a formula x with n variables, $\langle x, 0^n\rangle \in \mathrm{LeftSAT} \iff x \in \mathrm{SAT}$. Also, LeftSAT is in NP. Let us assume that $\mathrm{LeftSAT} \leq_m^p \mathrm{SAT}$ via reduction $g \in \mathrm{PF}$. Let $h(w) \stackrel{df}{=} f(g(w))$, and let $p(\cdot)$ be the computation time of h. Therefore, by assumption, for all w, $h(w)$ is a set of words such that $\|h(w)\| \leq O(\log |w|)$ and

$$w \in \mathrm{LeftSAT} \Leftrightarrow g(w) \in \mathrm{SAT} \Leftrightarrow h(w) \cap S = \emptyset.$$

Therefore, for every $S' \subseteq S$, and for all x, y,

$$\langle x, y\rangle \in \mathrm{LeftSAT} \implies h(\langle x, y\rangle) \cap S' = \emptyset.$$

Choose constants c and d such that $\|S^{\leq n}\| \leq c \log^\alpha n$ and $\|h(w)\| \leq d \log |w|$. Below we describe a *nondeterministic* polynomial-time-bounded algorithm that accepts SAT. We will see that this algorithm can be simulated in *deterministic* polynomial time. The input is a formula x.

```
1   S' := ∅
2   n := number of variables in x
3   if 1^n satisfies x, then accept x
    /* Otherwise, ⟨x, 1^n⟩ ∉ LeftSAT, and so h(⟨x, 1^n⟩) ∩ S ≠ ∅. */
4   choose some s ∈ h(⟨x, 1^n⟩) nondeterministically
5   S' := S' ∪ {s}
6   for i = 1 to c log^α p(|x| + n)
7       if h(⟨x, 0^n⟩) ∩ S' ≠ ∅, then reject
        /* At this point, h(⟨x, 0^n⟩) ∩ S' = ∅ and h(⟨x, 1^n⟩) ∩ S' ≠ ∅. */
8       Use binary search to determine a word y ∈ Σ^n − {1^n}
        such that h(⟨x, y⟩) ∩ S' = ∅ and h(⟨x, y + 1⟩) ∩ S' ≠ ∅.
```

9 if y satisfies x, then accept
10 choose some $s \in h(\langle x, y\rangle)$ nondeterministically
11 $S' := S' \cup \{s\}$
12 increment i
13 reject

We argue that the algorithm runs in nondeterministic polynomial time. The loop in steps 6–12 runs at most $c \log^\alpha p(|x| + n)$ times, and the binary search takes at most $O(n)$ steps for a formula of n variables. Therefore, the running time is bounded by a polynomial in $(n + |x|)$.

We argue that the algorithm accepts SAT. The algorithm accepts only if we find a satisfying assignment (step 3 or step 9). So all unsatisfiable formulas are rejected. We now show that all satisfiable formulas are accepted by at least one computation path.

Let x be a satisfiable formula; we describe an accepting computation path. On this path, S' will always be a subset of S. If x is accepted in step 3, then we are done. Otherwise, $\langle x, 1^n\rangle \notin$ LeftSAT, and therefore $h(\langle x, 1^n\rangle) \cap S \neq \emptyset$. So in step 4 at least one computation path chooses some $s \in S$.

Since $x \in$ SAT, $\langle x, 0^n\rangle \in$ LeftSAT. Hence $h(\langle x, 0^n\rangle) \cap S = \emptyset$. Since $S' \subseteq S$, it follows that $h(\langle x, 0^n\rangle) \cap S' = \emptyset$. Therefore, if $x \in$ SAT, the nondeterministic path that makes the correct choice for s in step 4 cannot reject x in step 7. Now we have

$$h(\langle x, 0^n\rangle) \cap S' = \emptyset \quad \text{and} \quad h(\langle x, 1^n\rangle) \cap S' \neq \emptyset.$$

Therefore, there must be some y as required by the algorithm, which can be obtained by binary search as follows. Initially, the algorithm considers the interval $[0^n, 1^n]$ and chooses the middle element 10^{n-1}. If $h(\langle x, 10^{n-1}\rangle) \cap S' \neq \emptyset$, then we proceed with the interval $[0^n, 10^{n-1}]$. Otherwise, we proceed with the interval $[10^{n-1}, 1^n]$. By continuing this procedure, we obtain intervals $[a, b]$ of decreasing size such that $a < b$ and

$$h(\langle x, a\rangle) \cap S' = \emptyset \quad \text{and} \quad h(\langle x, b\rangle) \cap S' \neq \emptyset.$$

If we accept in step 9, then we are done. Otherwise, we can argue as follows: By step 8, we have $h(\langle x, y + 1\rangle) \cap S' \neq \emptyset$ and therefore $h(\langle x, y + 1\rangle) \cap S \neq \emptyset$. Hence $\langle x, y + 1\rangle \notin$ LeftSAT. Together with the fact that y does not satisfy x in step 9, we obtain $\langle x, y\rangle \notin$ LeftSAT. Therefore, $h(\langle x, y\rangle) \cap S \neq \emptyset$. On the other hand, $h(\langle x, y\rangle) \cap S' = \emptyset$. Therefore, the correct nondeterministic path can choose an $s \in S - S'$ and continues with the next iteration of the loop. Along this path, S' is always a subset of $S \cap \Sigma^{\leq p(|x|+n)}$. By assumption,

$$\|S^{\leq p(|x|+n)}\| \leq c \cdot \log^\alpha p(|x| + n).$$

We enter the loop with $\|S'\| = 1$, and in each iteration we add a new element to S'. Hence at the beginning of the $(c \cdot \log^\alpha p(|x| + n))$th iteration it holds that $S' = S \cap \Sigma^{\leq p(|x|+n)}$. Now consider this iteration at step 8. Elements of $h(\langle x, y\rangle)$ and elements of $h(\langle x, y + 1\rangle)$ are of length $\leq p(|x| + n)$. So in this iteration we obtain a word y such that

$$h(\langle x, y\rangle) \cap S = \emptyset$$

and

$$h(\langle x, y + 1\rangle) \cap S \neq \emptyset.$$

It follows that $\langle x, y\rangle \in \text{LeftSAT}$ and $\langle x, y+1\rangle \notin \text{LeftSAT}$. So y is the lexicographically largest satisfying assignment of x. Therefore, we accept in step 9. It follows that our algorithm accepts SAT.

We argue that the algorithm can be simulated in deterministic polynomial time. Clearly, each path of the nondeterministic computation is polynomially bounded. We estimate the total number of paths as follows. Each path has at most

$$c \cdot \log^{\alpha} p(|x| + n) + 1$$

nondeterministic choices, where $\alpha < 1$. Each such nondeterministic choice guesses an $s \in h(\langle x, y\rangle)$ for some $y \in \Sigma^n$. By assumption, $\|h(\langle x, y\rangle)\| \leq d \cdot \log(|x| + n)$. Hence the total number of paths is

$$\begin{aligned}
&(d \cdot \log(|x| + n))^{c \cdot \log^{\alpha} p(|x|+n)+1} \\
&\leq 2^{O(\log\log(|x|+n)) \cdot O(\log^{\alpha}(|x|+n))} \\
&\leq 2^{O(\log^{1-\alpha}(|x|+n)) \cdot O(\log^{\alpha}(|x|+n))} \\
&\leq 2^{O(\log(|x|+n))} \\
&\leq (|x| + n)^{O(1)}.
\end{aligned}$$

Hence there is only a polynomial number of nondeterministic paths. Therefore, the algorithm can be simulated in deterministic polynomial time. □

THEOREM 3.9. *There exists a sparse* $S \in \text{DTIME}(2^{2^n})$ *such that for every* $L \in \text{NP} - \text{P}$, $L - S$ *is not* $\leq_m^p$*-hard for* NP.

Proof. Let $\{N_i\}_{i\geq 0}$ be an enumeration of all nondeterministic polynomial-time-bounded Turing machines such that for all i, the running time of N_i is bounded by the polynomial $p_i(n) = n^i + i$. Similarly, let $\{f_j\}_{j\geq 0}$ be an enumeration of all polynomial-time computable functions such that for all j, the running time of f_j is bounded by the polynomial $p_j(n) = n^j + j$. We use a polynomial-time computable and polynomial-time invertible pairing function $\langle\cdot,\cdot\rangle$ such that $r = \langle i, j\rangle$ implies $i \leq r$ and $j \leq r$.

A *requirement* is a natural number r. If $r = \langle i, j\rangle$, then we interpret this as the requirement that $L(N_i)$ does not many-one reduce to $L(N_i) - S$ via reduction function f_j.

Let $t(m) = 2^{2^m}$. We describe a decision algorithm for S. Let w be the input and let $n = |w|$.

```
1   if |w| < 4, then reject
2   n := |w|, m := greatest number such that t(m) ≤ n
3   for k = 1 to m
4       S_k := ∅, L_k := ∅
5       for r = 1 to k
6           if r ∉ L_1 ∪ L_2 ∪ ··· ∪ L_{k−1}, then
7               determine i and j such that r = ⟨i, j⟩,
8               for all z ∈ Σ^{<t(k+2)} in increasing lexicographic order
9                   y := f_j(z)
10                  if t(k) ≤ |y| < t(k + 1) and N_i(z) accepts, then
11                      S_k := {y}
12                      L_k := {r}
13                      exit the loops for z and r, and consider next k
14                  endif
15              increment z
```

16 endif
17 increment r
18 increment k
19 accept if and only if $S_m = \{w\}$

The algorithm works in stages $1, \ldots, m$, where m is the greatest natural number such that $t(m) \leq n$. In stage k, we construct a set S_k such that $\|S_k\| \leq 1$ and

$$S_k = \{w \in S \mid t(k) \leq |w| < t(k+1)\}.$$

Hence S can be written as $S_1 \cup S_2 \cup \cdots$. The input w is accepted if and only if it belongs to S_m. The aim of stage k is to satisfy a requirement $r \in \{1, \ldots, k\}$ that has not been satisfied so far (i.e., $r \notin L_1 \cup \cdots \cup L_{k-1}$). If more than one requirement is satisfiable, then we choose the smallest one. Requirement $r = \langle i, j \rangle$ is satisfied by placing a string y into S_k and therefore into S such that $f_j(z) = y$ for a suitable $z \in L(N_i)$. This makes sure that $L(N_i)$ does not many-one reduce to $L(N_i) - S$ via reduction f_j.

Whenever we refer to (the value of) a program variable without mentioning the time when we consider this variable, then we mean the value of the variable when the algorithm stops. Variables L_k represent sets of requirements. If requirement i is satisfied in stage k, then i is added to the set L_k. The algorithm ensures that $\|L_k\| \leq 1$ for every k.

We observe that S is sparse. For all inputs of length $\geq t(k)$ the algorithm constructs the same sets $S_1, S_2, \ldots, S_m$ and $L_1, L_2, \ldots, L_m$. Therefore, it is unambiguous to refer to S_k and L_k. Moreover, it is immediately clear that any S_k contains at most one word, and this word, if it exists, has a length that belongs to the interval $[t(k), t(k+1))$. By the definition of $t(k)$, for every $n \geq 4$,

$$t(\lfloor \log \log n \rfloor) \leq n < t(\lfloor \log \log n \rfloor + 1).$$

Hence on input of some word of length $n \geq 4$, we have

$$m = \lfloor \log \log n \rfloor \tag{1}$$

in step 2. So the algorithm computes singletons $S_1, S_2, \ldots, S_m$ such that $S^{\leq n} \subseteq S_1 \cup S_2 \cup \cdots \cup S_m$. It follows that

$$\|S^{\leq n}\| \leq \lfloor \log \log n \rfloor. \tag{2}$$

In particular, S is sparse.

We observe that $S \in \mathbf{DTIME}(2^{2^n})$. Note that in step 10, $|z| < t(m+2) = t(m)^4 \leq n^4$ and $i \leq r \leq m = \lfloor \log \log n \rfloor$. So a single path of the nondeterministic computation $N_i(z)$ has length

$$\leq n^{4i} + i \leq 2^{O(\log^2 n)}.$$

Hence the simulation of the complete computation takes

$$2^{2^{O(\log^2 n)}} \cdot 2^{O(\log^2 n)} = 2^{2^{O(\log^2 n)}}$$

steps. Similarly, step 9 takes $2^{O(\log^2 n)}$ steps. So the loop at steps 8–15 takes at most

$$2^{n^4} \cdot 2^{2^{O(\log^2 n)}} = 2^{2^{O(\log^2 n)}}$$

steps. The loops 5–17 and 3–18 multiply this number of steps at most by factor

$$m^2 \leq \lfloor \log \log n \rfloor^2.$$

Therefore, the overall running time is $2^{2^{O(\log^2 n)}}$. This shows

$$S \in \mathrm{DTIME}(2^{2^{O(\log^2 n)}}) \subseteq \mathrm{DTIME}(2^{2^n}).$$

We observe that no $L - S$ is many-one hard. Let $L \in \mathrm{NP} - \mathrm{P}$ and choose a machine N_i such that $L = L(N_i)$. Assume that $L - S$ is $\leq_m^p$-hard for NP. Therefore, there exists j such that $L \leq_m^p L - S$ via reduction function f_j. We consider two cases and show that both cases lead to contradiction. This will complete the proof.

Case 1. Assume there exists an $e \geq 1$ such that for all $x \in L^{\geq e}$, $|f_j(x)| < \sqrt{|x|}$. Using Lemma 3.8, we show that this implies $L \in \mathrm{P}$, thereby obtaining a contradiction.

Consider an arbitrary formula x. Let $s_0 \stackrel{df}{=} x$, and let $s_{l+1} \stackrel{df}{=} f_j(s_l)$ for $l \geq 0$. By assumption, for all $y \in L^{\geq e}$ it holds that

$$y \in L \iff f_j(y) \notin S \wedge f_j(y) \in L. \tag{3}$$

Hence

$$x \in L \iff s_1 \notin S \wedge s_1 \in L. \tag{4}$$

If $|s_1| \geq e$, we use equivalence (3) for $y = s_1$. We obtain

$$s_1 \in L \iff s_2 \notin S \wedge s_2 \in L. \tag{5}$$

By equivalences (5) and (4), we have

$$x \in L \iff s_1 \notin S \wedge s_2 \notin S \wedge s_2 \in L. \tag{6}$$

Now we use equivalence (3) again, this time for $y = s_2$. We proceed in this way until we reach an s_k such that either $|s_k| < e$ or $|s_k| \geq \sqrt{|s_{k-1}|}$. The following equivalence holds:

$$x \in L \iff \bigwedge_{l=1}^{k} s_l \notin S \wedge s_k \in L. \tag{7}$$

Note that if $|s_k| < e$, then it is easy to verify whether s_k belongs to L. So in polynomial time we can determine a string s which is defined as follows. If $s_k \in L^{<e}$, then let s be a fixed element from $\overline{S}$. Otherwise, let s be a fixed element from S. We show

$$x \in L \iff \{s_1, \ldots, s_k, s\} \cap S = \emptyset. \tag{8}$$

"⇒" Assume $x \in L$. Therefore, $s_1, \ldots, s_k \in L$. If $|x| < e$, then $k = 0$, $s \in \overline{S}$, and we are done. Otherwise, $|x| \geq e$ and $k \geq 1$. If $|s_k| < e$, then, by equivalence (7), $s \in \overline{S}$. So from equivalence (7) it follows that $\{s_1, \ldots, s_k, s\} \cap S = \emptyset$, and we are done. We show that the remaining case, i.e., $|s_k| \geq e$, is impossible. Since the algorithm terminated, it must be the case that $|s_k| \geq \sqrt{|s_{k-1}|}$. However, since $x \in L$, $s_k \in L$ by equivalence (7). By our assumption, it cannot happen that $s_k \in L^{\geq e}$ and $|s_k| \geq \sqrt{|s_{k-1}|}$. Therefore, $|s_k| < e$.

"⇐" Assume $\{s_1, \ldots, s_k, s\} \cap S = \emptyset$. Hence $s \in \overline{S}$ and therefore $s_k \in L^{<e}$. From equivalence (7) we obtain $x \in L$. This shows equivalence (8).

For $1 \leq l \leq k-1$ it holds that $|s_l| < \sqrt{|s_{l-1}|}$. Therefore, $k \leq |x|$, and so the strings s and s_i can be constructed in polynomial time in $|x|$. Let $g \in \mathrm{FP}$ be the function that on input x computes the set $\{s_1, \ldots, s_k, s\}$. So for all x,

$$x \in L \Leftrightarrow g(x) \cap S = \emptyset. \tag{9}$$

Observe that for all x,

$$\|g(x)\| \leq \lfloor \log \log |x| \rfloor + 1.$$

By assumption, $L - S$ is many-one hard for NP. So there exists a reduction function $h \in \mathrm{FP}$ such that $\mathrm{SAT} \leq_m^p L - S$ via h. By (9), for all x,

$$\begin{aligned} x \in \mathrm{SAT} &\Leftrightarrow h(x) \in L \wedge h(x) \notin S \\ &\Leftrightarrow (g(h(x)) \cup \{h(x)\}) \cap S = \emptyset. \end{aligned}$$

With $h'(x) \stackrel{df}{=} g(h(x)) \cup \{h(x)\}$ it holds that for all x,

$$x \in \mathrm{SAT} \Leftrightarrow h'(x) \cap S = \emptyset. \tag{10}$$

Clearly, h' belongs to FP and for all x,

$$\|h'(x)\| \leq \lfloor \log \log |h(x)| \rfloor + 2 \leq \lfloor \log \log |x| \rfloor + c \tag{11}$$

for a suitable constant c. By (2), (10), and (11), we satisfy the assumptions of Lemma 3.8 (take h', S, and $\alpha = 1/2$). It follows that $L \in \mathrm{P}$. This contradicts our assumption.

Case 2. Assume there exist infinitely many $x \in L$ such that $|f_j(x)| \geq \sqrt{|x|}$. We show that in this case L does not many-one reduce to $L - S$ via f_j. This will give us the necessary contradiction.

Recall that $L = L(N_i)$. Let $\bar{r} \stackrel{df}{=} \langle i, j \rangle$. Since every nonempty L_k contains a unique requirement, we can choose a number $m' \geq \bar{r}$ such that for all $k \geq m'$,

$$L_k \cap \{0, \ldots, \bar{r} - 1\} = \emptyset. \tag{12}$$

Note that for infinitely many strings $x \in L$, $|f_j(x)| \geq \sqrt{|x|}$. Therefore, we can choose some string $\bar{z} \in L$ such that

$$\sqrt{|\bar{z}|} \geq t(m') \tag{13}$$

and

$$|f_j(\bar{z})| \geq \sqrt{|\bar{z}|}. \tag{14}$$

Let $w \stackrel{df}{=} f_j(\bar{z})$ and $n \stackrel{df}{=} |w|$. Let $\bar{m}$ be such that $t(\bar{m}) \leq n < t(\bar{m}+1)$. By the choice of $\bar{z}$, $|\bar{z}| < t(\bar{m}+2)$. We will show that if $\bar{r}$ is not in $L_1 \cup \cdots \cup L_{\bar{m}-1}$, then $L_{\bar{m}} = \{\bar{r}\}$.

Consider the algorithm on input w. By the choice of $\bar{m}$, $t(\bar{m}) \leq |w| = n < t(\bar{m}+1)$. Therefore, by the choice of m in step 2 of the algorithm, $m = \bar{m}$. Consider step 6 when $k = \bar{m}$ and $r = \bar{r}$. We note that as a consequence of (12), for any $k \geq \bar{m} \geq m'$, $L_k \cap \{0, \ldots, \bar{r} - 1\} = \emptyset$. Therefore, the loop (steps 5–17) cannot exit with some $r < \bar{r}$. On the other hand, for $\bar{r}$, the condition in step 6 must be true, since we assumed that

$\bar{r} \notin L_1 \cup \cdots \cup L_{\bar{m}-1}$. Therefore, we reach step 7. By the choice of $\bar{m}$, $|\bar{z}| \leq t(\bar{m}+2)$. Therefore, either we reach step 9 such that $z = \bar{z}$ or $\bar{r}$ is put in $L_{\bar{m}}$ with some other z and $S_{\bar{m}} = \{f_j(z)\}$. If $z = \bar{z}$, then after step 9, $y = f_j(\bar{z}) = w$ and therefore $|y| = n$. Therefore, it must hold in step 10 that $t(\bar{m}) \leq |y| < t(\bar{m}+1)$. Moreover, $N_i(\bar{z})$ accepts since $\bar{z} \in L$. Therefore, we reach steps 11 and 12, where we obtain that $L_{\bar{m}} = \{\bar{r}\}$. As a consequence,

$$\bar{r} \notin L_1 \cup \cdots \cup L_{\bar{m}-1} \implies \bar{r} \in L_{\bar{m}}.$$

It follows that $\bar{r} \in L_1 \cup L_2 \cup \cdots \cup L_{\bar{m}}$. Let k, $1 \leq k \leq \bar{m}$, be such that $L_k = \{\bar{r}\}$. Let $S_k = \{y\}$. By steps 9 and 10, there exists a $z \in L$ such that $y = f_j(z)$. From $y \in S$ it follows that $y \notin L - S$. Therefore, L does not many-one reduce to $L - S$ via reduction function f_j. This contradicts our assumption. □

COROLLARY 3.10. *For every* $L \in \mathrm{NP} - \mathrm{P}$ *there exists a sparse* $S \in \mathrm{EXP}$ *such that* $L - S$ *is not* $\leq_m^p$*-hard for* NP.

Proof. Choose i such that $L = L(N_i)$. We recycle the proof of Theorem 3.9. Here we have to do only the diagonalization against the machine N_i. So we interpret r as the requirement that L does not many-one reduce to $L - S$ via reduction function f_r. We modify the algorithm in the proof of Theorem 3.9 by replacing step 7 with "`j := r`."

Analogously to the proof of Theorem 3.9 we observe that S is sparse. Because we modified the algorithm, now S belongs to EXP. This is seen as follows: Again in step 10,

$$|z| < t(m+2) = t(m)^4 \leq n^4.$$

But now i is a constant. So a single path of the nondeterministic computation $N_i(z)$ now has length $\leq n^{4i} + i$. Hence the simulation of the complete computation takes the following number of steps:

$$2^{n^{4i}+i} \cdot (n^{4i} + i) \leq 2^{n^{O(1)}}.$$

Note that

$$j = r \leq m \leq \lfloor \log \log n \rfloor.$$

So step 9 takes

$$n^{4j} + j \leq 2^{O(\log^2 n)} \leq 2^{O(n)}$$

steps. Therefore, the loop at steps 8–15 takes at most

$$2^{n^4} \cdot 2^{n^{O(1)}} = 2^{n^{O(1)}}$$

steps. The loops 5–17 and 3–18 multiply this number of steps at most by factor

$$m^2 \leq \lfloor \log \log n \rfloor^2.$$

Therefore, the overall running-time remains $2^{n^{O(1)}}$. This shows $S \in \mathrm{EXP}$.

Analogously to the proof of Theorem 3.9 we argue that $L - S$ is not $\leq_m^p$-hard for NP. Here we have to define $\bar{r} \stackrel{df}{=} j$ in Case 2. □

In view of Corollary 3.10 we would like to minimize the complexity of S. For this it suffices to consider $L = \mathrm{SAT}$: Given a sparse set S such that $\mathrm{SAT} - S$ is not $\leq_m^p$-hard for NP, for every $L \in \mathrm{NP}$ it is easy to describe a sparse set S' such that $S' \leq_m^p S$ and $L - S'$ is not $\leq_m^p$-hard for NP. Let $f \in \mathrm{PF}$ be the one-one function that reduces L to SAT. Then $S' = \{x \mid f(x) \in S\}$ is sparse and $L - S'$ reduces to $\mathrm{SAT} - S$ via f. So $L - S'$ cannot be $\leq_m^p$-hard for NP.

4. Immunity and closeness. Agrawal [Agr02] demonstrated that pseudorandom generators can be used to prove structural theorems on complete degrees of NP. Here we build on his results and show that hypotheses about pseudorandom generators and secure one-way permutations answer the longstanding open question of whether NP-complete sets can be immune. Also, using these hypotheses we show that no NP-complete set is quasi-polynomial-close to P.

It is well known that no EXP-complete set is p-immune. To see this, consider $L \in \text{EXP}$ that is $\leq_m^p$-complete. Then $0^* \leq_m^p L$ via some length-increasing reduction f. Since f is length-increasing, $\{f(0^n) \,|\, n \geq 0\}$ is an infinite subset of L. However, while for any EXP-complete set L and any $A \in \text{EXP}$, there is a length-increasing reduction from A to L [BH77], this is not known to hold for NP.

We begin with the following definitions. In particular it is important to distinguish pseudorandom generators, as defined by Nisan and Wigderson [NW94], for derandomization purposes, from cryptographic pseudorandom generators [Yao82, BM84].

DEFINITION 4.1. *A function $G = \{G_n\}_n$, $G_n : \Sigma^{=n} \mapsto \Sigma^{=m(n)}$, is an* $s(n)$-secure cryptographic pseudorandom generator *(*crypto-prg *for short) if G is computable in polynomial time in the input length, $m(n) > n$ and for every $\delta(\cdot)$ such that $\delta(n) < 1$, for every $t(\cdot)$ such that $t(n) \leq \delta(n) \cdot s(n)$, and for every circuit C of size $t(n)$, for all sufficiently large n,*

$$\left| \Pr_{x \in \Sigma^{=m(n)}}[C(x) = 1] - \Pr_{y \in \Sigma^{=n}}[C(G_n(y)) = 1] \right| \leq \delta(n).$$

DEFINITION 4.2. *A function $G = \{G_n\}_n$, $G_n : \Sigma^{=l} \mapsto \Sigma^{=n}$, is a* pseudorandom generator *(*prg *for short) if $l = O(\log n)$, G is computable in time polynomial in n, and for any linear-size circuit C,*

$$\left| \Pr_{x \in \Sigma^{=n}}[C(x) = 1] - \Pr_{y \in \Sigma^{=l}}[C(G_n(y)) = 1] \right| \leq \frac{1}{n}.$$

DEFINITION 4.3. *A function $f = \{f_n\}_n$, $f_n : \Sigma^{=n} \mapsto \Sigma^{=m(n)}$, is* $s(n)$-secure *if for every $\delta(\cdot)$ such that $\delta(n) < 1$, for every $t(\cdot)$ such that $t(n) \leq \delta(n) \cdot s(n)$, and for every nonuniform circuit family $\{C_n\}_n$ of size $t(n)$, for all sufficiently large n,*

$$\Pr_{x \in \Sigma^{=n}}[C_n(x) = f_n(x)] \leq \frac{1}{2^{m(n)}} + \delta(n).$$

Hypothesis A. Pseudorandom generators exist.

Hypothesis B. There is a secure one-way permutation. Technically, there is a permutation $\pi \in \text{PF}$ and $0 < \epsilon < 1$ such that π^{-1} is 2^{n^ϵ}-secure.

Hypothesis B implies the existence of cryptographic pseudorandom generators [Yao82]. Agrawal [Agr02] showed that if Hypothesis B holds, then every $\leq_m^p$-complete set for NP is hard also for one-one, length-increasing, nonuniform reductions. The following theorem is implicit in the proof of his result.

THEOREM 4.4. *If Hypotheses A and B hold, then every set A that is $\leq_m^p$-hard for* NP *is hard for* NP *under length-increasing reductions.*

By Theorem 4.4, Hypotheses A and B imply that for every NP-complete set A, there is a length-increasing reduction f from 0^* to A. This immediately implies that the set

$$\{f(0^n) \,|\, n \geq 0\}$$

is an infinite subset of A that belongs to P; i.e., A cannot be p-immune.

THEOREM 4.5. *If Hypotheses A and B hold, then no* $\leq_m^p$*-complete set for* NP *can be p-immune.*

We consider immunity with respect to classes that are larger than P. Similar questions have been studied for EXP. For example, Homer and Wang [HW94] showed that EXP-complete sets have dense UP subsets.

THEOREM 4.6. *Let* $\mathcal{C} \subseteq$ NP *be a complexity class closed under* $\leq_m^p$*-reductions such that for some* $\epsilon > 0$*, there is a tally set* $T \in \mathcal{C}$ *that is not in* DTIME(2^{n^ϵ})*. Then no* $\leq_m^p$*-complete set for* NP *is* $\mathcal{C}$*-immune.*

COROLLARY 4.7. *If there is a tally set in* UP *that is not in* DTIME(2^{n^ϵ})*, then no* $\leq_m^p$*-complete set for* NP *is* UP*-immune.*

Proof of Theorem 4.6. Let T be a tally set in $\mathcal{C}$ that does not belong to DTIME(2^{n^ϵ}). We will show that no NP-complete set is $\mathcal{C}$-immune.

Let L be an NP-complete set, and let $k > 0$ such that $L \in$ DTIME(2^{n^k}). Let f be a $\leq_m^p$-reduction from T to L. We claim that the set

$$X = \{f(0^n) \,|\, 0^n \in T \text{ and } |f(0^n)| > n^{\epsilon/k}\}$$

is infinite. Assume otherwise: Then, for all but finitely many n, $0^n \in T \implies |f(0^n)| \leq n^{\epsilon/k}$. Consider the following algorithm that accepts a finite variation of T: On input 0^n, if $|f(0^n)| \leq n^{\epsilon/k}$, then accept 0^n if and only if $f(0^n) \in L$. Otherwise, reject 0^n. This algorithm takes time at most $2^{|f(0^n)|^k} \leq 2^{(n^{\epsilon/k})^k} = 2^{n^\epsilon}$. This contradicts the assumption that $T \notin$ DTIME(2^{n^ϵ}). Therefore, X is infinite. Also, $X \subseteq f(T) \subseteq L$. Now we will show that $X \leq_m^p T$. Since T belongs to $\mathcal{C}$ and $\mathcal{C}$ is closed under $\leq_m^p$-reductions, that will demonstrate that L is not $\mathcal{C}$-immune.

To see that $X \leq_m^p T$, we apply the following reduction: On input $y, |y| = m$, determine whether $f(0^i) = y$ for some $i < m^{k/\epsilon}$. If there is such an i, then output the first such 0^i. Otherwise, $y \notin X$. In this case, output some fixed string not in T. We need to show that $y \in X$ if and only if the output of this reduction belongs to T. If $y \in X$, then there exists i such that $i < m^{k/\epsilon}$, $0^i \in T$, and $f(0^i) = y$. Let 0^{i_0} be the output of the reduction. In this case, $y = f(0^i) = f(0^{i_0})$. Now recall that f is a reduction from T to L. For this reason, $0^i \in T$ if and only if $0^{i_0} \in T$. The converse case, that $y \notin X$, is straightforward. □

Agrawal [Agr02] defined a function $g \in$ PF to be γ*-sparsely many-one* on $S \subseteq \{0,1\}^n$ if

$$\forall x \in S,\ \|g^{-1}(g(x)) \cap \{0,1\}^n\| \leq \frac{2^n}{2^{n^\gamma}}.$$

Here $g^{-1}(z) = \{x \,|\, g(x) = z\}$. The function g is *sparsely many-one* on $S \subseteq \{0,1\}^n$ if it is γ-sparsely many-one on $S \subseteq \{0,1\}^n$ for some $\gamma > 0$.

Given a 2^{n^ϵ}-secure one-way permutation, Goldreich and Levin [GL89] construct a 2^{n^α}-secure crypto-prg, $0 < \alpha < \epsilon$. This crypto-prg G is defined only on strings of even length; i.e., G is a partial function. However, Agrawal [Agr02] notes that G can be extended to be total, and the security remains the same. This crypto-prg has a nice property; namely, it is a one-one function.

Let S be any set in NP and L be any NP-complete language. Let $S' = G(S)$. Since S' is in NP, there is a many-one reduction f from S' to L. Let $h \stackrel{def}{=} f \circ G$. Since G is one-one, h is a many-one reduction from S to L.

LEMMA 4.8 (see [Agr02]). *For every* n*,* $h \stackrel{def}{=} f \circ G$ *is a* $\alpha/2$*-sparsely many-one on* $S \cap \Sigma^{=n}$*, where* α *is the security parameter of* G*.*

LEMMA 4.9. *Let f be a γ-sparsely many-one function on $S = 0^* \times \Sigma^* \cap \{0,1\}^n$ for every n, and let $l = n^{2/\gamma}$. Then, for sufficiently large n,*

$$\|\{w \in 0^n \times \Sigma^{=l} \mid |f(w)| > n\}\| \geq \frac{3}{4}2^l.$$

Proof. Let $S_n = 0^n \times \Sigma^{=l}$. Every string in S_n has length $m = n + l$. For every $w \in S_n$, there are at most $\frac{2^m}{2^{m^\gamma}}$ strings of length m that can map to $f(w)$. Therefore, $\|f(S_n)\| \geq 2^l/(\frac{2^m}{2^{m^\gamma}})$. Taking $l = n^{\frac{2}{\gamma}}$, we obtain that at least $\frac{3}{4}$ of the strings in S_n have image of length $> n$. □

THEOREM 4.10. *If Hypothesis B holds, then for every $\epsilon > 0$, no $\leq_m^p$-complete set for* NP *can be* DTIME(2^{n^ϵ})*-immune.*

Proof. The hypothesis implies the existence of a 2^{n^ϵ}-secure one-way permutation. Let G be the 2^{n^α}-secure crypto-prg, $0 < \alpha < \epsilon$, constructed from this secure one-way function. Let $S = 0^* \times \Sigma^*$, and let $S' = G(S)$. Since L is NP-complete $S' \leq_m^p L$ via f. Thus $S \leq_m^p L$ via $h = f \circ G$. By Lemma 4.8, h is $\alpha/2$-sparsely many-one on $S \cap \Sigma^{=n}$ for every n. For any n, take $l = n^{4/\alpha}$. Then, by Lemma 4.9, we know that for large enough n, at least $\frac{3}{4}$ of the strings in $0^n \times \Sigma^{=l}$ map via h to a string of length $> n$.

Let $k = \frac{4}{\epsilon\alpha}$. Assume G maps strings of length n to strings of length n^r, $r > 0$. It is well known that from G we can construct a crypto-prg G' that expands n bits to n^k bits [Gol01, p. 115]. Thus for any string w of length n^ϵ, $G'(w)$ is of length $l = n^{4/\alpha}$. Consider the following circuit that on input $(0^n, y), |y| = l$ accepts if and only if $|h(0^n, y)| > n$. This circuit accepts at least $\frac{3}{4}$ of the inputs $(0^n, y), |y| = l$, if the input is chosen according to uniform distribution. Therefore, there must be some $w, |w| = n^\epsilon$, such that this circuit accepts $G'(w)$. Therefore, for this w, $|h(0^n, G'(w))| > n$. Now, the following DTIME(2^{n^ϵ})-algorithm outputs infinitely many strings of L:

Input 0^n
Let $m = n^\epsilon$
for $w \in \Sigma^{=m}$
  If $|h(0^n, G'(w))| > n$, then output $h(G'(w))$ □

4.1. Closeness. In general, Yesha [Yes83] considered two sets A and B to be close if the census of their symmetric difference, $A\Delta B$, is a slowly increasing function. For example, A and B are *p-close* if there is a polynomial p such that for every n, $\|(A\Delta B)^{=n}\| \leq p(n)$. Ogiwara [Ogi91] and Fu [Fu93] observed that if A is NP-complete, then A is not p-close to any set $B \in$ P, unless P = NP. Define A and B to be *quasi-polynomial-close* if there exists a constant k such that for every n, $\|(A\Delta B)^{=n}\| \leq 2^{\log^k n}$. We show that if Hypothesis B holds, then no NP-complete set is quasi-polynomial-close to a set in P. We show unconditionally that if L is paddable and quasi-polynomial-close to a set in P, then L belongs to BPP. As a corollary, if Hypothesis A holds, then no paddable NP-complete set is quasi-polynomial-close to a set in P.

We recall the following definitions and recall that all known, natural NP-complete sets are paddable [BH77].

DEFINITION 4.11. *A set A is* paddable *if there exists $p(\cdot,\cdot)$, a polynomial-time computable, polynomial-time invertible (i.e., there is a $g \in$ PF such that for all x and y, $g(p(x,y)) = \langle x, y\rangle$) function, such that for all a and x,*

$$a \in A \iff p(a,x) \in A.$$

Recall that a set A is *p-isomorphic* to B if there exists f, a polynomial-time computable, polynomial-time invertible permutation on Σ^*, such that $A \leq_m^p B$ via f.

Mahaney and Young [MY85] proved that two paddable sets are many-one equivalent if and only if they are p-isomorphic.

THEOREM 4.12. *If L is paddable and quasi-polynomial-close to a set in* P*, then* $L \in$ BPP.

Proof. Assume that L is a paddable set and there is a set $B \in$ P such that L is quasi-polynomial-close to B. Let $p(\cdot,\cdot)$ be a padding function for L. Given a string x, $|x| = n$, consider the following set:

$$P_x = \{p(x,y) \,|\, |x| = |y|\}.$$

Let q be a polynomial such that all strings in P_x have length $\leq q(n)$. Let k be a constant such that $\|(L \Delta B)^{\leq q(n)}\| \leq 2^{\log^k n}$. Note that $\|P_x\| = 2^n$.

If $x \in L$, then $P_x \subseteq L$. Therefore, at least $2^n - 2^{\log^k n}$ strings from P_x belong to B. On the other hand, if $x \notin L$, then $P_x \cap L = \emptyset$, and so at least $2^n - 2^{\log^k n}$ strings from P_x are not in B. Therefore,

$$x \in L \Rightarrow \Pr_{y \in \Sigma^n}[p(x,y) \in B] \geq 1 - \frac{2^{\log^k n}}{2^n},$$

$$x \notin L \Rightarrow \Pr_{y \in \Sigma^n}[p(x,y) \in B] \leq \frac{2^{\log^k n}}{2^n}.$$

Therefore, $L \in$ BPP. □

COROLLARY 4.13. *If* SAT *is quasi-polynomial-close to a set in* P*, then* NP = RP.

This follows immediately from the result of Ko [Ko82] that NP $\subseteq$ BPP implies NP = RP. Hypothesis A implies that BPP = P. Therefore, we have the following corollaries.

COROLLARY 4.14. *If Hypothesis A holds, then no paddable set $L \notin$ P can be quasi-polynomial-close to any set in* P.

COROLLARY 4.15. *If Hypothesis A holds, then no set p-isomorphic to* SAT *can be quasi-polynomial-close to any set in* P*, unless* P = NP.

Next we are interested primarily in Theorems 4.16 and 4.18 and their immediate consequence, Corollary 4.19. Theorem 4.16 follows directly from the statement of Hypothesis B.

THEOREM 4.16. *Hypothesis B implies that*

$$\text{NP} \not\subseteq \bigcup_{k>0} \text{DTIME}(2^{\log^k n}).$$

Proof. Hypothesis B asserts the existence of a 2^{n^ϵ}-secure one-way permutation π for some $0 < \epsilon < 1$. No 2^{n^ϵ}-size circuit can compute the inverse of π. So the set

$$B = \{\langle y, i\rangle \,|\, i\text{th bit of } \pi^{-1}(y) = 0\}$$

belongs to NP and cannot have a quasi-polynomial-size family of circuits. However, if $B \in \text{DTIME}(2^{\log^k n})$ for some $k > 0$, then B has a family of circuits of size $(2^{\log^k n})^2 < 2^{\log^{2k} n}$, which is a contradiction. □

We require the following proposition, which follows from Homer and Longpré's study of Ogiwara–Watanabe pruning [HL94].

PROPOSITION 4.17. *If there exists a set S that has a quasi-polynomially bounded census function and that is $\leq^p_{btt}$-hard for* NP*, then* NP $\subseteq \bigcup_{k>0} \text{DTIME}(2^{\log^k n})$.

THEOREM 4.18. *If* NP $\not\subseteq \bigcup_{k>0} \mathrm{DTIME}(2^{\log^k n})$, *then no* NP*-complete set is quasi-polynomial-close to a set in* P.

Proof. Assume there exists an NP-complete A that is quasi-polynomial-close to some $B \in \mathrm{P}$. Let $S \stackrel{df}{=} A \Delta B$. So S has a quasi-polynomially bounded census function. $A \leq^p_{1-tt} S$, and therefore S is $\leq^p_{1-tt}$-hard for NP. By Proposition 4.17, $\mathrm{NP} \subseteq \bigcup_{k>0} \mathrm{DTIME}(2^{\log^k n})$. □

As an immediate consequence, we have the following corollary, which has a stronger consequence than Corollary 4.15.

COROLLARY 4.19. *If Hypothesis B holds, then no* NP*-complete set is quasi-polynomial-close to any set in* P.

It is interesting to note that Corollary 4.19 has a short proof that does not depend on Theorems 4.16 and 4.18. We present that proof now.

Proof. We begin as the proof of Theorem 4.16 begins: Hypothesis B asserts the existence of a 2^{n^ϵ}-secure one-way permutation π. No 2^{n^ϵ}-size circuit can compute the inverse of π. So the set $B = \{\langle y, i\rangle \mid i\text{th bit of } \pi^{-1}(y) = 0\}$ belongs to NP and cannot have a quasi-polynomial-size family of circuits.

Let us assume that L is an NP-complete set such that there is some set $S \in \mathrm{P}$ and some $k > 0$ such that for every n, $\|L \Delta S\| \leq 2^{\log^k n}$. This implies that $L \in \mathrm{P}/(2^{\log^k n})$, where the advice for any length n is the set of strings in $L \Delta S$. On an input x, accept x if and only if $x \in S$ and x is not in the advice set or $x \notin S$ and x belong to the advice set.

Therefore, L has a family of quasi-polynomial-size circuits. Since L is NP-complete, it follows that every set in NP has a quasi-polynomial-size family of circuits. By the above discussion, this contradicts Hypothesis B. □

Finally, we state another consequence of Theorem 4.18, which has the same conclusion as Corollary 4.19. Buhrman and Homer [BH92] proved that NP-complete sets do not have circuits of quasi-polynomial size unless the exponential hierarchy collapses to $\mathrm{NEXP}^{\mathrm{NP}}$, the second level.

COROLLARY 4.20. *If the exponential hierarchy does not collapse to its second level, then no* NP*-complete set is quasi-polynomial-close to any set in* P.

5. Disjoint pairs. Recall that if $\mathrm{NP} \cap \mathrm{coNP} \neq \mathrm{P}$, then there exist disjoint sets A and B in NP such that $A \not\leq^p_T A \cup B$. Our first result derives the same consequence under the assumption that UEE $\neq$ EE.

THEOREM 5.1. *If* UEE $\neq$ EE, *then there exist two disjoint sets* A *and* B *in* UP *such that* $A \not\leq^p_T A \cup B$.

Proof. Beigel et al. [BBFG91] showed that if NEE $\neq$ EE, then there exists a language in $\mathrm{NP} - \mathrm{P}$ for which search does not reduce to decision. Their proof also shows that if UEE $\neq$ EE, then there exists a language S in $\mathrm{UP} - \mathrm{P}$ for which search does not reduce to decision. Let M be an unambiguous Turing machine that accepts S, and for every word $x \in S$, let a_x be the unique accepting computation of M on x. Let p be a polynomial such that for all $x \in S$, $|a_x| = p(|x|)$. Define

$$A = \{\langle x, y\rangle \mid x \in S, |y| = p(|x|), \text{ and } y \leq a_x\}$$

and

$$B = \{\langle x, y\rangle \mid x \in S, |y| = p(|x|), \text{ and } y > a_x\}.$$

Both A and B belong to UP and are disjoint. Let

$$A \cup B = S' = \{\langle x, y\rangle \mid x \in S \text{ and } |y| = p(|x|)\}.$$

Note that S' is many-one reducible to S. Now assume $A \leq_T^p S'$. Since S' is many-one reducible to S, it follows that $A \leq_T^p S$. However, we can compute the witness a_x for $x \in S$ by using a binary search algorithm with oracle A. Therefore, replacing A with S, we see that search reduces to decision for S, contradicting our choice of S. □

Next we study the question of whether there exist two disjoint Turing complete sets for NP whose union is not Turing complete. We consider the following hypothesis.

UP-machine hypothesis. There is a UP-machine M that accepts 0^* such that for some $0 < \epsilon < 1$, no 2^{n^ϵ} time-bounded machine can correctly compute infinitely many accepting computations of M.

The UP-machine hypothesis and its variant, the NP-machine hypothesis (obtained by replacing the UP-machine with an NP-machine), have been studied previously and shown to have several believable consequences. For example, Pavan and Selman [PS02] showed that the UP-machine hypothesis implies the existence of a Turing complete language for NP that is not truth-table complete, and the NP-machine hypothesis implies the existence of a Turing complete language for NP that is not many-one complete. Hitchcock and Pavan [HP04] showed that the NP-machine hypothesis implies AM = NP, $\mathrm{BPP}^{\mathrm{NP}} = \mathrm{P}^{\mathrm{NP}}$, and NEXP $\not\subseteq$ P/*poly*. Hemaspaandra, Rothe, and Wechsung [HRW97] showed several equivalent characterizations of variants of the NP- and UP-machine hypotheses. Fenner et al. [FFNR96] studied the infinite-often version of the NP-machine hypothesis.

Hitchcock and Pavan showed that if UE ∩ coUE is not included in $\mathrm{DTIME}(2^{2^{\epsilon n}})$ infinitely often, then the UP-machine hypothesis holds. They also related this hypothesis to the question of UE ∩ coUE having high circuit complexity: Let K be the standard complete set for EXP. If UE ∩ coUE does not have K-oracle circuits of size $2^{\epsilon n}$ infinitely often, then the UP-machine hypothesis holds. They also showed that these hypotheses are weaker than some hypotheses used in the context of uniform derandomization.

We show that if UP ∩ coUP has bi-immune languages, then the UP-machine hypothesis holds.

THEOREM 5.2. *If there is a* $\mathrm{DTIME}(2^{n^\epsilon})$*-bi-immune language in* UP ∩ coUP*, then the* UP*-machine hypothesis is true.*

Proof. Let $L \in$ UP ∩ coUP be the $\mathrm{DTIME}(2^{n^\epsilon})$-bi-immune set, and let N and N' be the UP-machines for L and $\overline{L}$. Consider the following machine M that accepts 0^*: On input 0^n, M guesses an accepting computation of N and of N' on 0^n, and accept 0^n if either guess is right. Note that for every 0^n, exactly one of the guesses will be correct, and therefore $L(M) = 0^*$. If there is a 2^{n^ϵ} time-bounded machine T that can correctly compute infinitely many accepting computation of M, then either $X = \{0^i \mid T(0^i)$ outputs an accepting computation of $N\}$ or $X' = \{0^i \mid T(0^i)$ outputs an accepting computation of $N'\}$ is an infinite subset of L or $\overline{L}$, contradicting the bi-immunity of L. □

THEOREM 5.3. *If the* UP*-machine hypothesis is true, then there exist two disjoint Turing complete sets for* NP *whose union is not Turing complete.*

Proof. Let a_n be the accepting computation of M on 0^n. Let $p(n)$ be the polynomial that bounds $|a_n|$. Note that a deterministic machine can verify in polynomial time whether a string of length $p(n)$ is an accepting path of M. Consider the following sets:

$$A = \{\langle x, a_m + 1\rangle \mid |x| = n, x \in \mathrm{SAT}, m = (2n)^{1/\epsilon}\} \\ \oplus \{\langle 0^n, i\rangle \mid i \leq p(n), \text{ bit } i \text{ of } a_n = 1\}$$

and

$$B = \{\langle x, a_m - 1\rangle \,\big|\, |x| = n, x \in \text{SAT}, m = (2n)^{1/\epsilon}\} \oplus \{\langle 0^n, i\rangle \,\big|\, i \le p(n), \text{ bit } i \text{ of } a_n = 0\}.$$

It is easy to see that both A and B are Turing-complete for NP. They can be made disjoint by choosing an appropriate pairing function. Note that

$$A \cup B = \{\langle x, a\rangle \,\big|\, |x| = n, x \in \text{SAT}, a = a_m - 1 \text{ or } a_m + 1, m = (2n)^{1/\epsilon}\} \oplus \{\langle 0^n, i\rangle \,\big|\, i \le p(n)\}.$$

Assume that $A \cup B$ is Turing complete for NP. Since the set $\{\langle 0^n, i\rangle \,\big|\, i \le p(n)\}$ is in P, the following set is Turing complete:

$$C = \{\langle x, a\rangle \,\big|\, |x| = n, x \in \text{SAT}, a = a_m - 1 \text{ or } a_m + 1, m = (2n)^{1/\epsilon}\}.$$

Consider the set

$$S = \{\langle 0^n, i\rangle \,\big|\, \text{ bit } i \text{ of } a_n = 1\}.$$

Since $S \in$ NP, $S \le_T^p C$ via some oracle Turing machine U.

We describe the following procedure $\mathcal{A}$:

1. input 0^n.
2. Simulate U on strings $\langle 0^n, i\rangle$, where $1 \le i \le p(n)$.
3. Let $q = \langle x, y\rangle$ be a query that is generated. If $y \ne a_t + 1$ or $y \ne a_t - 1$ for some t, then continue the simulation with answer "No."
4. Else, $q = \langle x, y\rangle$, $|x| = t^\epsilon/2$, and $y = a_t + 1$ or $y = a_t - 1$.
5. If $t \ge n^\epsilon$, then output "Unsuccessful," print a_t, and Halt.
6. Otherwise, check whether $x \in$ SAT; this takes at most $2^{|x|} \le 2^{n^{\epsilon^2}/2}$ time. Answer the query appropriately and continue the simulation of U.

Now we consider two cases.

CLAIM 5.4. *If $\mathcal{A}(0^n)$ does not output "Unsuccessful" for infinitely many n, then there is a 2^{n^ϵ}-time bounded machine that correctly outputs infinitely many accepting computations of M.*

Proof. Assume $\mathcal{A}(0^n)$ does not output "Unsuccessful." This implies that $\mathcal{A}$ is able to decide membership of $\langle 0^n, i\rangle$, $1 \le i \le p(n)$, in S. Therefore, $\mathcal{A}$ can compute a_n. The most expensive step of the above procedure is step 6, where $\mathcal{A}$ decides the membership of x in SAT. However, this occurs only if $|x| \le n^{\epsilon^2}/2$ and hence takes at most $2^{n^{\epsilon^2}/2}$ time. Thus the total time is bounded by $O(p(n) \times q(n) \times 2^{n^{\epsilon^2}/2})$, where $q(n)$ is the running time of U on $\langle 0^n, i\rangle$. Since $\epsilon < 1$, this is bounded by 2^{n^ϵ}. □

CLAIM 5.5. *If $\mathcal{A}(0^n)$ outputs "Unsuccessful" for all but finitely many n, then there is a 2^{n^ϵ}-time bounded machine that outputs infinitely many accepting computations of M.*

Proof. If $\mathcal{A}(0^n)$ is unsuccessful, then it outputs a string a_t such that $t \ge n^\epsilon$. Hence, if $\mathcal{A}(0^n)$ is unsuccessful for all but finitely many strings, then for infinitely many t there exists an n, where $n \le t^{1/\epsilon}$, and $\mathcal{A}(0^n)$ outputs a_t. Thus the following procedure computes infinitely many accepting computations of M:

input 0^t
for $i = 1$ to $t^{1/\epsilon}$ do
  if $\mathcal{A}(0^j)$ outputs a_t
    output a_t and halt.
  endif
end for

Note that $\mathcal{A}(0^i)$ runs in time $O(p(i) \times q(i) \times 2^{i^{\epsilon^2}/2})$. Thus the total running time of the above procedure is $O(2^{t^\epsilon})$. □

Claims 5.4 and 5.5 show that if C is Turing complete for NP, then there is a 2^{n^ϵ}-time bounded Turing machine that computes infinitely many accepting computations of M. This contradicts the UP-machine hypothesis, and therefore $A \cup B$ cannot be Turing complete for NP. □

5.1. Many-one complete languages. Here we consider the analogous questions for many-one reductions. We first show under two different hypotheses that there exist disjoint sets A and B in NP such that $A \not\leq_m^p A \cup B$. Also we study the question for NP-complete sets. One of our results will show a relation between our question and propositional proof systems. We refer the reader to Glasser et al. [GSS04] for definitions about proof systems and reductions between disjoint NP-pairs.

THEOREM 5.6. *If* $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$*, then there exist disjoint* $A, B \in \mathrm{NP}$ *such that*
1. A *and* B *are many-one equivalent and*
2. $A \not\leq_m^p A \cup B$.

Proof. Let $b \in \{0,1\}$, and let $L \in \mathrm{NP} \cap \mathrm{coNP} - \mathrm{P}$. Define

$$A = \{bw \,|\, b = \chi_L(w)\}$$

and

$$B = \{bw \,|\, b \neq \chi_L(w)\}.$$

Both A and B belong to $\mathrm{NP} \cap \mathrm{coNP} - \mathrm{P}$. Note that $A \cup B = \{0,1\} \circ \Sigma^*$. However, note that $A \leq_m^p B$ via $f(bw) = \bar{b}w$, and the same reduction reduces B to A. Also note that $w \to 1w$ reduces L to A, and hence A cannot be in P. Therefore, $A \not\leq_m^p A \cup B$. □

THEOREM 5.7. *If* $\mathrm{UE} \neq \mathrm{E}$*, then there exist disjoint sets* A *and* B *in* NP *such that* $A \not\leq_m^p A \cup B$.

Proof. Hemaspaandra et al. [HNOS96] showed that if $\mathrm{NE} \neq \mathrm{E}$, then there exists a language S in NP for which search does not reduce to decision nonadaptively. Essentially the same proof shows that if $\mathrm{UE} \neq \mathrm{E}$, then there exists a language S in UP for which search does not reduce to decision nonadaptively. Since $S \in \mathrm{UP}$, for each $x \in S$, there is a unique witness v_x, where $|v_x| = p(|x|)$, for some polynomial p. Define

$$A = \{\langle x, i\rangle \,|\, x \in S, i \leq p(|x|), \text{ and the } i\text{th bit of the witness } v_x \text{ of } x \text{ is } 0\}$$

and

$$B = \{\langle x, i\rangle \,|\, x \in S, i \leq p(|x|), \text{ and the } i\text{th bit of the witness } v_x \text{ of } x \text{ is } 1\}.$$

It is clear that both A and B are in NP and are disjoint. Then

$$A \cup B = S' = \{\langle x, i\rangle \,|\, x \in S, i \leq p(|x|)\}.$$

Observe that $S' \leq_m^p S$. Assume $A \leq_m^p S'$; then $A \leq_m^p S$. Therefore, we can compute the ith bit of the witness of x by making one query to S. This implies that search nonadaptively reduces to decision for S, which is a contradiction. □

Two disjoint sets A and B are P-*separable* if there is a set $S \in \mathrm{P}$ such that $A \subseteq S \subseteq \overline{B}$. Otherwise, they are P-*inseparable*. Let us say that (A, B) is a *disjoint* NP-*pair* if A and B are disjoint sets that belong to NP. If (A, B) is a disjoint NP-pair such that A and B are P-separable, then $A \leq_m^p A \cup B$ follows easily: On input x, the reduction outputs x if $x \in S$ and outputs some fixed string $w \notin A \cup B$ if $x \notin S$. This observation might lead one to conjecture that $A \cup B$ is not $\leq_m^p$-complete if A and B are disjoint, P-inseparable, $\leq_m^p$-complete NP sets. The following theorem shows that this would be false, assuming $\mathrm{P} \neq \mathrm{UP}$.

THEOREM 5.8. *If* $\mathrm{P} \neq \mathrm{UP}$*, then there exist disjoint* NP*-complete sets* A *and* B *such that*

1. (A, B) *is* P*-inseparable and*
2. $A \cup B$ *is many-one complete for* NP.

Proof. Under the assumption that $\mathrm{P} \neq \mathrm{UP}$, Grollmann and Selman [GS88] constructed a P-inseparable disjoint NP-pair (A', B') such that A' and B' are NP complete. Let

$$A \stackrel{df}{=} 0A' \cup 1\mathrm{SAT}$$

and

$$B \stackrel{df}{=} 0B'.$$

Therefore, $A \cap B = \emptyset$. Also, $\mathrm{SAT} \leq_m^p A \cup B$ via $f(\phi) = 1\phi$. Therefore, $A \cup B$ is NP complete. If (A, B) is P-separable, then so is (A', B'). □

Similar arguments show that in Theorem 5.8, we can use hypothesis $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ instead of $\mathrm{P} \neq \mathrm{UP}$: If $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$, then there exist sets A and B with the same properties as in Theorem 5.8.

We learn from the next theorem that if there exist disjoint NP-complete sets whose union is not NP-complete, then this happens already for paddable NP-complete sets.

THEOREM 5.9. *The following are equivalent:*

1. *There exist an* NP*-complete set* A *and a set* $B \in \mathrm{NP}$ *such that* $A \cap B = \emptyset$ *and* $A \cup B$ *is not* NP*-complete.*
2. *There exist disjoint,* NP*-complete sets* A *and* B *such that* $A \cup B$ *is not* NP*-complete.*
3. *There exist paddable, disjoint,* NP*-complete sets* A *and* B *such that* $A \cup B$ *is not* NP*-complete.*
4. *For every paddable* NP*-complete set* A*, there is a paddable* NP*-complete set* B *such that* $A \cap B = \emptyset$ *and* $A \cup B$ *is not* NP*-complete. Furthermore, there is a polynomial-time-computable permutation* π *on* Σ^* *such that*
   a. *for all* x, $\pi(\pi(x)) = x$ *and*
   b. $A \leq_m^p B$ *and* $B \leq_m^p A$*, both via* π.
5. *For every* NP*-complete set* A*, there is a set* $B \in \mathrm{NP}$ *such that* $A \cap B = \emptyset$ *and* $A \cup B$ *is not* NP*-complete.*

By Theorem 5.9, if there exist disjoint, NP-complete sets whose union is not complete, then there is a set B in NP that is disjoint from SAT such that $\mathrm{SAT} \cup B$ is not NP-complete. Moreover, in that case, there exists such a set B so that B is p-isomorphic to SAT. It is even the case that SAT and B are $\leq_m^p$-reducible to one another via the same polynomial-time computable permutation.

Proof. $1 \Rightarrow 2$: Let $A' \stackrel{df}{=} 0A \cup 1B$ and $B' \stackrel{df}{=} 1A \cup 0B$. Since A is NP-complete, both sets A' and B' are NP-complete. However, $A' \cup B' = \{0,1\} \cdot (A \cup B)$, and hence is not NP-complete.

$2 \Rightarrow 3$: Choose A and B according to item 2. Let $A' \stackrel{df}{=} A \times \Sigma^*$ and $B' \stackrel{df}{=} B \times \Sigma^*$. A' and B' are disjoint, paddable, and NP-complete. $A' \cup B' = (A \cup B) \times \Sigma^*$. Hence $A' \cup B' \leq_m^p A \cup B$, and therefore $A' \cup B'$ is not NP-complete.

$3 \Rightarrow 4$: Choose A and B according to item 3. We may assume that there exists a polynomial-time computable permutation π on Σ^* such that

- for all x, $\pi(\pi(x)) = x$ and
- $A \leq_m^p B$ and $B \leq_m^p A$, both via π.

Otherwise, we use $0A \cup 1B$ and $1A \cup 0B$ instead of A and B; and π is the permutation on Σ^* that flips the first bit.

Let A' be any paddable NP-complete set. So A' and A are paddable and many-one equivalent. Therefore, A' and A are p-isomorphic; i.e., there exists f, a polynomial-time computable, polynomial-time invertible permutation on Σ^*, such that $A' \leq_m^p A$ via f.

Let $B' \stackrel{df}{=} f^{-1}(B)$. $B' \leq_m^p B$ via f and, therefore B' and B are p-isomorphic. It follows that B' is paddable and NP-complete. $A' \cap B' = \emptyset$, since $A \cap B = \emptyset$. Moreover, $A' \cup B' \leq_m^p A \cup B$ via f, and hence $A' \cup B'$ is not NP-complete. Let $\pi'(x) \stackrel{df}{=} f^{-1}(\pi(f(x)))$. So π' is a polynomial-time computable permutation on Σ^*. For all x,

$$\pi'(\pi'(x)) = f^{-1}(\pi(f(f^{-1}(\pi(f(x)))))) = x.$$

Moreover, for all x,

$$x \in A' \iff f(x) \in A \iff \pi(f(x)) \in B \iff \pi'(x) \in B'.$$

Therefore, $A' \leq_m^p B'$ via π', and analogously, $B' \leq_m^p A'$ via π'.

$4 \Rightarrow 1$: The proof follows immediately, since SAT is paddable and NP-complete.

$1 \Rightarrow 5$: Choose A and B according to item 1, and let A' be an arbitrary NP-complete set. Let $f \in \mathrm{PF}$ such that $A' \leq_m^p A$ via f. $B' \stackrel{df}{=} \{x \mid f(x) \in B\}$. Clearly, $B' \in \mathrm{NP}$ and $A' \cap B' = \emptyset$, since $A \cap B = \emptyset$. For all x,

$$x \in A' \cup B' \Leftrightarrow f(x) \in A \vee f(x) \in B \Leftrightarrow f(x) \in A \cup B.$$

So $A' \cup B' \leq_m^p A \cup B$ via f and, therefore $A' \cup B'$ is not NP-complete.

$5 \Rightarrow 1$: The proof is trivial. □

Next we state relations between our question and propositional proof systems [CR79]. The recent paper of Glasser et al. [GSSZ03] contains definitions of the relevant concepts: propositional proof systems (pps), optimal pps for a pps f, the canonical disjoint NP-pair $(\mathrm{SAT}^*, \mathrm{REF}_f)$ of f, and reductions between disjoint NP-pairs. If a propositional proof system f is optimal, then Razborov [Raz94] has shown that the canonical disjoint NP-pair of f is $\leq_m^{pp}$-complete. Therefore, it is natural to ask, for any proof system f, whether the union $\mathrm{SAT}^* \cup \mathrm{REF}_f$ of the canonical pair is complete for NP. However, this always holds. It holds for trivial reasons, because SAT reduces to $\mathrm{SAT}^* \cup \mathrm{REF}_f$ by mapping every x to (x, ϵ). Since x does not have a proof of size 0, we never map to REF_f. However, $x \in \mathrm{SAT} \Leftrightarrow (x, \epsilon) \in \mathrm{SAT}^*$. Nevertheless, it is interesting to inquire, as we do in the following theorem, whether some perturbation of the canonical proof system might yield disjoint sets in NP whose union is not complete.

THEOREM 5.10. *Assume* P $\neq$ NP *and there exist disjoint sets* A *and* B *in* NP *such that* A *is* NP*-complete but* $A \cup B$ *is not* NP*-complete. Then there exists a pps* f *and a set* $X \in$ P *such that*

1. $\mathrm{SAT}^* \cap X$ *is* NP*-complete and*
2. $(\mathrm{SAT}^* \cap X) \cup (\mathrm{REF}_f \cap X)$ *is not* NP*-complete.*

Proof. If NP = coNP, then $\overline{\mathrm{SAT}}$ has a polynomially bounded pps f. Let p be the bound, and let $X \stackrel{df}{=} \{(x, y) \,\big|\, y = 0^p(|x|)\}$. Clearly, $\mathrm{SAT}^* \cap X$ is NP-complete. Observe that

$$(\mathrm{SAT}^* \cap X) \cup (\mathrm{REF}_f \cap X) = X.$$

Since the latter set is in P and P $\neq$ NP, it cannot be NP-complete. So in this case we are done.

From now on, let us assume that NP $\neq$ coNP. By Theorem 5.9, there exists $B' \in$ NP such that $B' \subseteq \overline{\mathrm{SAT}}$ and $\mathrm{SAT} \cup B'$ is not NP-complete. Let $C \in$ P and p be a polynomial such that for all x,

$$x \in B' \iff \exists y \in \Sigma^{p(|x|)}[(x, y) \in C].$$

Choose a polynomial-time-computable, polynomial-time-invertible pairing function $\langle \cdot, \cdot \rangle$ such that for all x and y, $|\langle x, y \rangle| = 2|xy|$. Define the following pps:

$$f(z) \stackrel{df}{=} \begin{cases} x & \text{if } z = \langle x, y \rangle,\ |y| = p(|x|), \text{ and } (x, y) \in C, \\ x & \text{if } z = \left\langle x, 0^{2^{2^{|x|}}} \right\rangle \text{ and } x \in \overline{\mathrm{SAT}}, \\ \text{false} & \text{otherwise.} \end{cases}$$

Observe that f is a pps. Define

$$X \stackrel{df}{=} \{(x, 0^m) \,\big|\, m = 2(|x| + p(|x|))\}.$$

$X \in$ P. Let $\mathrm{SAT}' \stackrel{df}{=} \mathrm{SAT}^* \cap X$ and $\mathrm{REF}' \stackrel{df}{=} \mathrm{REF}_f \cap X$. $\mathrm{SAT}' \in$ NP and $\mathrm{REF}' \in$ NP. Moreover, SAT' is NP-complete.

It remains to show that $\mathrm{SAT}' \cup \mathrm{REF}'$ is not NP-complete. Let α be a fixed element in $\overline{\mathrm{SAT} \cup B'}$. (Such an element exists, because otherwise NP = coNP.) We show $\mathrm{SAT}' \cup \mathrm{REF}' \leq_m^p \mathrm{SAT} \cup B'$ via the following reduction function:

$$h(x, y) \stackrel{df}{=} \begin{cases} x & \text{if } (x, y) \in X, \\ \alpha & \text{otherwise.} \end{cases}$$

Assume $(x, y) \in \mathrm{SAT}' \cup \mathrm{REF}'$. Hence $(x, y) \in X$ and therefore $y = 0^{2(|x|+p(|x|))}$ and $h(x, y) = x$. If $(x, y) \in \mathrm{SAT}'$, then $h(x, y) = x \in \mathrm{SAT}$. If $(x, y) \in \mathrm{REF}'$, then there exists $z \in \Sigma^{\leq 2(|x|+p(|x|))}$ such that $f(z) = x$. By the definition of f, there exists $z \in \Sigma^{2(|x|+p(|x|))}$ such that $z = \langle x, y \rangle$, $|y| = p(|x|)$, and $(x, y) \in C$. Hence $h(x, y) = x \in B'$.

Now assume $(x, y) \notin \mathrm{SAT}' \cup \mathrm{REF}'$. If $(x, y) \notin X$, then $h(x, y) = \alpha \notin \mathrm{SAT} \cup B'$, and we are done. Otherwise, $(x, y) \in X$. First, $h(x, y) = x \notin \mathrm{SAT}$, since $(x, y) \notin \mathrm{SAT}'$. Second, if $x \in B'$, then there exists $y \in \Sigma^{p(|x|)}$ such that $(x, y) \in C$. Therefore, if $x \in B'$, then there exists $z \in \Sigma^{\leq 2(|x|+p(|x|))}$ such that $f(z) = x$. The latter is not possible, since $(x, y) \notin \mathrm{REF}'$. It follows that $h(x, y) = x \notin B'$.

This shows $\mathrm{SAT}' \cup \mathrm{REF}' \leq_m^p \mathrm{SAT} \cup B'$ via h. Hence, $\mathrm{SAT}' \cup \mathrm{REF}'$ is not NP-complete. □

In Theorem 5.11 we show that if there are sets A and B belonging to NP such that $A \cap B = \emptyset$ and $A \cup B$ is not NP-complete, then (A, B) cannot be a $\leq_{sm}^{pp}$-complete disjoint NP-pair.

THEOREM 5.11. *If (A, B) is a $\leq_{sm}^{pp}$-complete disjoint* NP*-pair, then A, B, and $A \cup B$ are* NP*-complete.*

Proof. Since the disjoint NP-pair $(\mathrm{SAT}, \{z \wedge \bar{z}\})$ $\leq_{sm}^{pp}$-reduces to (A, B), $\mathrm{SAT} \leq_m^p A$; i.e., A is NP-complete. Similarly, B is NP-complete as well. Assume that $(\mathrm{SAT}, \{z \wedge \bar{z}\})$ $\leq_{sm}^{pp}$-reduces to (A, B) via some reduction function f. Let

$$f'(x) \stackrel{df}{=} \begin{cases} f(x) & \text{if } x \neq z \wedge \bar{z}, \\ f(y \wedge z \wedge \bar{z}) & \text{if } x = z \wedge \bar{z}. \end{cases}$$

We obtain $f'(\mathrm{SAT}) \subseteq A \cup B$ and $f'(\overline{\mathrm{SAT}}) \subseteq \overline{A \cup B}$. Hence $A \cup B$ is NP-complete. □

According to the comments after Theorem 5.8, the converse of Theorem 5.11 does not hold if either $\mathrm{P} \neq \mathrm{UP}$ or $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$. Since we know that there exists a $\leq_{sm}^{pp}$-complete disjoint NP-pair if and only if there is a $\leq_m^{pp}$-complete disjoint NP-pair [GSS04], we obtain the following corollary.

COROLLARY 5.12. *If $\leq_m^{pp}$-complete disjoint NP-pairs exist, then there is a $\leq_m^{pp}$-complete disjoint NP-pair such that both components and their union are* NP*-complete.*

5.2. Relativizations. We have been considering the following questions:

1. Do there exist disjoint sets A and B in NP such that both A and B are $\leq_T^p$-complete but $A \cup B$ is not $\leq_T^p$-complete?
2. Do there exist disjoint sets A and B in NP such that both A and B are NP-complete but $A \cup B$ is not NP-complete?

We observe here that there exist oracles relative to which both of these questions have both "yes" and "no" answers. This implies that resolving these questions would require nonrelativizable techniques.

PROPOSITION 5.13. *If the union of every two disjoint $\leq_T^p$-complete sets for* NP *is $\leq_T^p$-complete for* NP*, then* $\mathrm{P} \neq \mathrm{NP} \implies \mathrm{NP} \neq \mathrm{coNP}$.

Proof. Let us assume that $\mathrm{NP} = \mathrm{coNP}$. Then $\mathrm{SAT} \cup \overline{\mathrm{SAT}} = \Sigma^*$, which is $\leq_T^p$-complete if and only if $\mathrm{P} = \mathrm{NP}$. □

Therefore, relative to an oracle for which $\mathrm{P} \neq \mathrm{NP} = \mathrm{coNP}$ holds [BGS75], the answer to question 1 is "yes." Also, it is obvious that relative to an oracle for which $\mathrm{P} = \mathrm{NP}$, the answer to this question is "no" [BGS75].

Now we consider question 2.

PROPOSITION 5.14. *If the union of every two disjoint* NP*-complete sets is* NP*-complete, then* $\mathrm{NP} \neq \mathrm{coNP}$.

Therefore, an oracle relative to which $\mathrm{NP} = \mathrm{coNP}$ holds will answer "yes" to question 2. We learned already that if A and B are disjoint, NP-complete, P-separable sets, then $A \cup B$ is NP-complete. Homer and Selman [HS92] construct an oracle relative to which all disjoint NP-pairs are P-separable, yet $\mathrm{P} \neq \mathrm{NP}$. Therefore, relative to this oracle, the answer to question 2 is "no." Indeed, relative to this oracle, the answer to question 1 is "no" also.

Acknowledgments. The authors are appreciative of enlightening conversation with M. Agrawal. The authors thank H. Buhrman for informing them of his work on

quasi-polynomial density and thank M. Ogiwara for informing them of his results on polynomial closeness. Also, we received helpful suggestions from L. Hemaspaandra.

REFERENCES

[Agr02] M. Agrawal, *Pseudo-random generators and structure of complete degrees*, in Proceedings of the 17th IEEE Conference on Computational Complexity, IEEE Computer Society Press, Los Alamitos, CA, 2002, pp. 139–147.

[BBFG91] R. Beigel, M. Bellare, J. Feigenbaum, and S. Goldwasser, *Languages that are easier than their proofs*, in Proceedings of the 32nd Annual Symposium on Foundations of Computer Science, 1991, pp. 19–28.

[Ber76] L. Berman, *On the structure of complete sets: Almost everywhere complexity and infinitely often speedup*, in Proceedings of the 17th IEEE Symposium on Foundations of Computing, IEEE Computer Society Press, Los Alamitos, CA, 1976, pp. 76–80.

[BGS75] T. Baker, J. Gill, and R. Solovay, *Relativizations of the $\mathscr{P} =? \mathscr{NP}$ question*, SIAM J. Comput., 4 (1975), pp. 431–442.

[BH77] L. Berman and J. Hartmanis, *On isomorphisms and density of NP and other complete sets*, SIAM J. Comput., 6 (1977), pp. 305–322.

[BH92] H. Buhrman and S. Homer, *Superpolynomial circuits, almost sparse oracles, and the exponential hierarchy*, in Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Sci. 652, Springer-Verlag, Berlin, 1992, pp. 116–127.

[BHT98] H. Buhrman, A. Hoene, and L. Torenvliet, *Splittings, robustness, and structure of complete sets*, SIAM J. Comput., 27 (1998), pp. 637–653.

[BM84] M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudo-random bits*, SIAM J. Comput., 13 (1984), pp. 850–864.

[CR79] S. Cook and R. Reckhow, *The relative efficiency of propositional proof systems*, J. Symbolic Logic, 44 (1979), pp. 36–50.

[FFNR96] S. Fenner, L. Fortnow, A. Naik, and J. Rogers, *On inverting onto functions*, in Proceedings of the 11th Conference on Computational Complexity, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 213–223.

[FPS01] L. Fortnow, A. Pavan, and A. Selman, *Distributionally hard languages*, Theory Comput. Syst., 34 (2001), pp. 245–261.

[Fu93] B. Fu, *On lower bounds of the closeness between complexity classes*, Math. Systems Theory, 26 (1993), pp. 187–202.

[GL89] O. Goldreich and L. Levin, *A hardcore predicate for all one-way functions*, in Proceedings of the 21st Annual ACM Sympositum on Theory of Computing, ACM, New York, 1989, pp. 25–32.

[Gol01] O. Goldreich, *Foundations of Cryptography–Volume* 1, Cambridge University Press, New York, 2001.

[GS88] J. Grollmann and A. L. Selman, *Complexity measures for public-key cryptosystems*, SIAM J. Comput., 17 (1988), pp. 309–335.

[GSS04] C. Glasser, A. Selman, and S. Sengupta, *Reductions between disjoint NP-pairs*, Inform. and Comput., 200 (2005), pp. 247–267.

[GSSZ03] C. Glasser, A. L. Selman, S. Sengupta, and L. Zhang, *Disjoint NP-pairs*, SIAM J. Comput., 33 (2004), pp. 1369–1416.

[HL94] S. Homer and L. Longpré, *On reductions of np sets to sparse sets*, J. Comput. System Sci., 48 (1994), pp. 324–336.

[HNOS96] E. Hemaspaandra, A. Naik, M. Ogiwara, and A. Selman, *P-selective sets and reducing search to decision vs. self-reducibility*, J. Comput. System Sci., 53 (1996), pp. 194–209.

[HP04] J. Hitchcock and A. Pavan, *Hardness hypotheses, derandomization, and circuit complexity*, in Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Comput. Sci. 3328, Springer-Verlag, Berlin, 2004, pp. 184–197.

[HRW97] L. Hemaspaandra, J. Rothe, and G. Wechsung, *Easy sets and hard certificate schemes*, Acta Inform., 34 (1997), pp. 859–879.

[HS92] S. Homer and A. Selman, *Oracles for structural properties: The isomorphism problem and public-key cryptography*, J. Comput. System Sci., 44 (1992), pp. 287–301.

[HW94] S. Homer and J. Wang, *Immunity of complete problems*, Inform. Comput., 110 (1994), pp. 119–129.

[Ko82] K.-I. Ko, *Some observations on probabilistic algorithms and NP-hard problems*, Inform. Process. Lett., 14 (1982), pp. 39–43.

[Mah82] S. Mahaney, *Sparse complete sets for NP: Solution of a conjecture of Berman and Hartmanis*, J. Comput. System Sci., 25 (1982), pp. 130–143.

[MY85] S. Mahaney and P. Young, *Reductions among polynomial isomorphism types*, Theoret. Comput. Sci., 39 (1985), pp. 207–224.

[NW94] N. Nisan and A. Wigderson, *Hardness vs. randomness*, J. Comput. System Sci., 49 (1994), pp. 149–167.

[Ogi91] M. Ogiwara, *On P-Closeness of Polynomial-Time Hard Sets*, manuscript, 1991.

[OW91] M. Ogiwara and O. Watanabe, *On polynomial-time bounded truth-table reducibility of NP sets to sparse sets*, SIAM J. Comput., 20 (1991), pp. 471–483.

[PS01] A. Pavan and A. Selman, *Separation of NP-completeness notions*, in Proceedings of the 16th IEEE Conference on Computational Complexity, IEEE Computer Society Press, Los Alamitos, CA, 2001.

[PS02] A. Pavan and A. L. Selman, *Separation of NP-completeness notions*, SIAM J. Comput., 31 (2002), pp. 906–918.

[Raz94] A. Razborov, *On provably disjoint NP-pairs*, Technical report TR94-006, Electronic Colloquium on Computational Complexity, 1994.

[Sch86] U. Schöning, *Complete sets and closeness to complexity classes*, Math. Systems Theory, 19 (1986), pp. 24–41.

[Sel79] A. Selman, *P-selective sets, tally languages, and the behavior of polynomial-time reducibilities on NP*, Math. Systems Theory, 13 (1979), pp. 55–65.

[Sel88] A. L. Selman, *Natural self-reducible sets*, SIAM J. Comput., 17 (1988), pp. 989–996.

[Sho76] J. R. Shoenfield, *Degrees of classes of RE sets*, J. Symbolic Logic, 41 (1976), pp. 695–696.

[TFL93] S. Tang, B. Fu, and T. Liu, *Exponential-time and subexponential-time sets*, Theoret. Comput. Sci., 115 (1993), pp. 371–381.

[Tod91] S. Toda, *On polynomial-time truth-table reducibilities of intractable sets to P-selective sets*, Math. Systems Theory, 24 (1991), pp. 69–82.

[Yao82] A. C. C. Yao, *Theory and applications of trapdoor functions*, in Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1982, pp. 80–91.

[Yes83] Y. Yesha, *On certain polynomial-time truth-table reducibilities of complete sets to sparse sets*, SIAM J. Comput., 12 (1983), pp. 411–425.

SIAM J. COMPUT.
Vol. 36, No. 2, pp. 543–561

THE DIRECTED STEINER NETWORK PROBLEM IS TRACTABLE FOR A CONSTANT NUMBER OF TERMINALS*

JON FELDMAN† AND MATTHIAS RUHL‡

Abstract. We consider the DIRECTED STEINER NETWORK problem, also called the POINT-TO-POINT CONNECTION problem. Given a directed graph G and p pairs $\{(s_1, t_1), \ldots, (s_p, t_p)\}$ of nodes in the graph, one has to find the smallest subgraph H of G that contains paths from s_i to t_i for all i. The problem is NP-hard for general p, since the DIRECTED STEINER TREE problem is a special case. Until now, the complexity was unknown for constant $p \geq 3$. We prove that the problem is polynomially solvable if p is any constant number, even if nodes and edges in G are weighted and the goal is to minimize the total weight of the subgraph H. In addition, we give an efficient algorithm for the STRONGLY CONNECTED STEINER SUBGRAPH problem for any constant p, where given a directed graph and p nodes in the graph, one has to compute the smallest strongly connected subgraph containing the p nodes.

Key words. Steiner tree, Steiner network, network design, directed graphs, polynomial-time algorithms

AMS subject classifications. 05C20, 05C85, 68Q25, 68W05

DOI. 10.1137/S0097539704441241

1. Introduction.

The Steiner problem is one of the classic problems in computational geometry and theoretical computer science and is fundamental to the study of network design. The basic problem is the following: Given a set of points, what is the most efficient way to connect them? The first combinatorial formulation of a Steiner problem was the classic STEINER TREE problem, independently due to Hakimi [Hak71] and Levin [Lev71]. This was followed by extensive research on different variants of Steiner problems in the last 30 years.

In this paper we address one of the most general Steiner problems, the DIRECTED STEINER NETWORK problem, also called the POINT-TO-POINT CONNECTION problem.

DIRECTED STEINER NETWORK (p-DSN). *Given a directed graph $G = (V, E)$, with weights on the edges and p pairs of nodes in the graph $\{(s_1, t_1), \ldots, (s_p, t_p)\}$, find the minimum weight subgraph H of G that contains paths from s_i to t_i for $1 \leq i \leq p$.*

When p is arbitrary, this problem is NP-hard, since it generalizes STEINER TREE. The case $p = 2$ was solved in 1992 by Li, McCormick, and Simchi-Levi [LMS92]. However, when p is a fixed quantity greater than two, the question of whether there exists a polynomial-time algorithm for p-DSN was open until now.

In this paper we give the first polynomial-time algorithm for the case where p is any fixed constant, resolving this open question. In fact, the difficulty of the p-DSN problem is contained within a special case of p-DSN for strongly connected components:

STRONGLY CONNECTED STEINER SUBGRAPH (p-SCSS). *Given a directed graph*

*Received by the editors February 24, 2004; accepted for publication (in revised form) January 5, 2006; published electronically August 7, 2006. Preliminary versions of this paper have appeared as [FR99, Fel00]. This work was performed while both authors were students at the Massachusetts Institute of Technology, Cambridge, MA.

http://www.siam.org/journals/sicomp/36-2/44124.html

†Columbia University, New York, NY 10027 (jonfeld@ieor.columbia.edu).

‡Google, Mountain View, CA 94043 (ruhl@google.com).

$G = (V, E)$ and p terminal vertices $\{s_1, \ldots, s_p\}$ in V, find the smallest strongly connected subgraph H of G that contains $s_1, \ldots, s_p$.

In this paper we also give the first polynomial-time algorithm for p-SCSS for any constant p and use it as a subroutine in our algorithm for p-DSN.

1.1. Previous results.

Constant number of terminals. When $p = 1$, there is only one terminal pair (s, t); thus the solution to p-DSN is simply the shortest path from s to t. Finding a shortest path in a directed graph is a well-known problem solvable in polynomial time [CLRS01].

When all s_i are the same node, the p-DSN problem becomes the Directed Steiner Tree problem. This is solvable using an algorithm due to Dreyfus and Wagner [DW71] (and discovered independently by Levin [Lev71]). This algorithm exploits the fact that a solution to the Directed Steiner Tree problem can be described by the topology of the solution tree, i.e., the relative order in which the paths to the terminals diverge. Since the solution is always a tree, once two paths diverge, they never meet again; this makes the number of different topologies independent of the size of the graph. In contrast, topologies of solutions to the p-DSN problem can be arbitrary dags, which makes enumerating topologies infeasible.

When $p = 2$, the p-DSN problem becomes more difficult but still admits a reasonably simple solution. To motivate the solution for the general case, we present a solution for $p = 2$ in section 2. This problem was first solved by Li, McCormick, and Simchi-Levi [LMS92], who called the problem the Point-to-Point Connection problem. The running time of their algorithm is $\mathcal{O}(n^5)$. They also state the case for all $p \geq 3$ as an interesting open question.

Natu and Fang [NF95, NF97] improved the running time for the case where $p = 2$ first to $\mathcal{O}(n^4)$, and then to $\mathcal{O}(mn+n^2 \log n)$. They also propose an algorithm for $p = 3$ and conjecture that a variant of their algorithm works for all constant p. However, the conference version of the present paper [FR99] gives a counterexample to the correctness of their algorithm for $p = 3$, and thus to their conjecture.

Arbitrary number of terminals. There has been some work on the approximability of p-DSN for arbitrary p. The best positive result obtained so far is by Charikar et al. [CCC+98], who achieve an approximation ratio of $\mathcal{O}(p^{2/3} \log^{1/3} p)$ for any p. On the negative side, Dodis and Khanna [DK99] prove that p-DSN is $\Omega(2^{\log^{1-\varepsilon} p})$-hard; that is, unless $P = NP$, no algorithm exists for p-DSN that has an approximation ratio of $o(2^{\log^{1-\varepsilon} p})$.

The method of Charikar et al. [CCC+98] seems to be fundamentally different from what we use for the fixed-parameter version. Their algorithm is based on greedily finding low-cost "bunches" of vertices that connect a certain subset of the terminal pairs. A bunch has a simple structure; it consists of a path from a node u to a node v, and shortest paths from each of the sources to u and from v to each of the terminals. In this paper, we show structural properties of optimal solutions of p-DSN that are more complex than these bunches. These properties also exist for arbitrary p (but unfortunately cannot easily be found in polynomial time). It is our hope that the ideas presented here can be used to obtain a better approximation ratio for an arbitrary number of terminals.

1.2. Applications. Algorithms for Steiner problems are fundamental tools used for designing networks [MW84]. When many points in a network need to be connected and there is a cost associated with connecting them, the minimum Steiner tree between

the points represents the cheapest way of building the network. Examples of such networks include transportation, communication, or shipping. Leung, Magnanti, and Singhal [LMS90] discuss these applications in more detail.

Until recently, the directed version of the Steiner problem was of mostly theoretical interest, since networks were usually symmetric. However, with the increasing diversity of network links such as satellite and radio, link costs are becoming less symmetric [Ram96]. Therefore, the proper model for designing some types of networks is one where the underlying graph is directed [SRV97].

The problem of multicast tree generation is one example of the use of directed Steiner problems in network design that has received some attention recently [Ram96, SRV97]. A multicast tree is used for point-to-multipoint communication in high-bandwidth applications. For example, in video-conferencing, a single source must be broadcast to many different destinations. The routes taken to each destination are encoded in the multicast tree. Finding a low-cost multicast tree makes for more efficient use of resources. Ramanathan [Ram96] uses the DIRECTED STEINER TREE problem to find low-cost multicast trees.

1.3. Our contributions. In this paper, we give an exact algorithm for p-DSN for any constant p with a running time of $\mathcal{O}(mn^{4p-2} + n^{4p-1} \log n)$, where $n = |V|$ and $m = |E|$. (Throughout this paper, we will assume that the input graphs are connected, so that we have $m \geq n - 1$.) We also give an exact algorithm for p-SCSS for any constant p with a running time of $\mathcal{O}(mn^{2p-3} + n^{2p-2} \log n)$. If the input graphs are unweighted, these bounds can be slightly improved. For details on running times, see section 6.

For clarity, we present an algorithm for a version of p-DSN where there are no weights and the goal is to minimize the number of *nodes* in the subgraph H. Section 5.2 shows how to generalize our algorithm to weights and edge-minimization through simple modifications.

Our algorithm for p-DSN can best be understood in terms of a game, where a player moves tokens around the graph. Initially, p tokens are placed on the starting nodes $s_1, \ldots, s_p$, one token per node. The player is then allowed to make certain types of moves with the tokens, and his goal is to perform a series of these moves to get the tokens to their respective destinations $t_1, \ldots, t_p$ (the token from s_1 to t_1, the token from s_2 to t_2, etc.).

Every possible move has a cost associated with it: the number of nodes that are visited by the moving tokens. We define the moves carefully so that the lowest cost move sequence to get the tokens from $s_1, \ldots, s_p$ to $t_1, \ldots, t_p$ will visit only nodes from an optimal subgraph H and will visit each node in H exactly once. The difficulty of the construction is to ensure that such a sequence exists for every optimal H. For $p = 2$ this is easy to do, since the two involved paths can share vertices only in a very restricted manner. However, for $p \geq 3$ the relationships between the paths become significantly more complex. Critical to our argument is a structural lemma analyzing how these paths may overlap.

1.4. Outline of the paper. In section 2, we give a simple algorithm that solves p-SCSS for $p = 2$, while also defining the token game in more detail. We generalize this approach to any constant p and state the algorithm solving p-SCSS in section 3, making use of a token game similar to the one described above. The correctness proof is given in sections 3.4 and 4.

Using the algorithm for p-SCSS, in section 5 we then give the algorithm for the p-DSN problem and prove its correctness. We analyze the running times of our

algorithms in section 6, and conclude the paper in section 7 by summarizing our results and discussing possible future research directions.

2. A solution for 2-SCSS. We begin by solving unweighted 2-SCSS, the problem of finding a minimum (in terms of the number of nodes) strongly connected subgraph H of a graph $G = (V, E)$ that includes two specified nodes s and t. This is equivalent to finding the smallest H that contains paths from s to t and from t to s. Considering this simple problem allows us to introduce the notation and methodology used in the following sections. The algorithm described here is similar to the one given by Natu and Fang [NF97].

Figure 1 illustrates some of the difficulties of this problem. Let s, t be our terminals. The optimal subgraph consists of the six nodes $s, x_7, x_8, x_9, x_{10}, t$. The paths from s to t and t to s share vertex x_7 and the vertex sequence $x_8 \to x_9 \to x_{10}$. Note that the optimal subgraph includes neither the shortest path from s to t nor the shortest path from t to s.

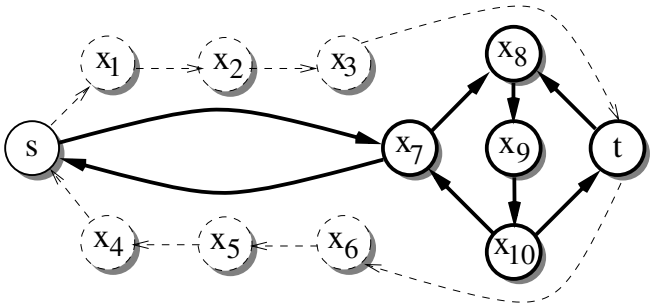


FIG. 1. *A sample graph for 2-SCSS, with terminals s and t. The optimal subgraph is in solid lines.*

2.1. The token game. To compute the optimal subgraph H, we will place two tokens, called f and b, on vertex s. We then move the tokens along edges, f moving forward along edges, and b moving backward along edges, until they both reach t. Then the set of nodes visited during the sequence of moves will contain paths $s \rightsquigarrow t$ and $t \rightsquigarrow s$.

To find the smallest subgraph H containing those paths, we will charge for the moves. The cost of a move will be the number of new vertices entered by the tokens during that move. Clearly we can never find a move sequence that gets both tokens from s to t with a total cost lower than $|H| - 1$, the size of the optimal solution H minus 1 (since we never charge for s). In fact, we will show that the lowest cost move sequence to get the tokens from s to t will have cost exactly $|H| - 1$, and thus corresponds to an optimal solution.

The three kinds of moves we allow are given below. The notation $\langle x, y \rangle$ refers to the situation where token f is on vertex x, and token b is on vertex y. The expression "$\langle x_1, y_1 \rangle \xrightarrow{c} \langle x_2, y_2 \rangle$" means that it is legal to move token f from x_1 to x_2 and token b from y_1 to y_2 (at the same time), and that this move has cost c. We want to find a move sequence from $\langle s, s \rangle$ to $\langle t, t \rangle$ with minimal cost.

(i) *Token f moving forward*: For every edge $(u, v) \in E$ and all $x \in V$, we allow
  (a) the move $\langle u, x \rangle \xrightarrow{1} \langle v, x \rangle$ and
  (b) the move $\langle u, v \rangle \xrightarrow{0} \langle v, v \rangle$.

(ii) *Token b moving backward*: For every edge $(u, v) \in E$ and all $x \in V$, we allow
  (a) the move $\langle x, v\rangle \xrightarrow{1} \langle x, u\rangle$ and
  (b) the move $\langle u, v\rangle \xrightarrow{0} \langle u, u\rangle$.

(iii) *Tokens switching places*: For every pair of vertices $a, b \in V$ for which there is a path from a to b in G, we allow the move $\langle a, b\rangle \xrightarrow{c} \langle b, a\rangle$, where c is the length of the shortest path from a to b in G. By length we mean the number of vertices besides a and b on that path.

Type (i) and (ii) moves allow the tokens f and b to move forward along a single edge and backward along an edge, respectively. Usually the cost is 1, accounting for the new vertex that the token visits. Only in the case where a token reaches a vertex with a token already on it, the cost is 0, since no "new" vertices are visited.

Type (iii) moves allow the two tokens to switch places. We call this type of move a "flip" and say that the vertices on the shortest path from a to b are *implicitly* traversed by the tokens. The cost c of the move accounts for all of these vertices.

Let us return to the example in Figure 1 to see how these moves are used. The lowest cost way to move both tokens from s to t is the following (we use subscripts to denote the type of the move):

$$\langle s, s\rangle \xrightarrow[\text{(i)}]{1} \langle x_7, s\rangle \xrightarrow[\text{(ii)}]{0} \langle x_7, x_7\rangle \xrightarrow[\text{(i)}]{1} \langle x_8, x_7\rangle$$
$$\xrightarrow[\text{(ii)}]{1} \langle x_8, x_{10}\rangle \xrightarrow[\text{(iii)}]{1} \langle x_{10}, x_8\rangle \xrightarrow[\text{(ii)}]{1} \langle x_{10}, t\rangle \xrightarrow[\text{(i)}]{0} \langle t, t\rangle.$$

The weight of this sequence is 5, which is $|H| - 1$, and the nodes visited by the tokens are exactly the nodes in the optimal solution H.

2.2. The algorithm. Let us phrase the preceding discussion in an algorithmic form. To compute H, we first construct a "game-graph" $\widetilde{G}$. The nodes of the graph correspond to token positions $\langle x, y\rangle$, the edges to legal moves between positions. In our case, the nodes are just $V \times V$, and the edges are the ones given above as legal moves.

The solution H is found by computing a lowest cost path from $\langle s, s\rangle$ to $\langle t, t\rangle$ in $\widetilde{G}$. The graph H then consists of all the vertices from V that are mentioned along that path, including the vertices that are implied by type (iii) moves.

Running time. Clearly, this game-graph can be computed in polynomial time. First, we perform an all-pairs-shortest-paths computation on G to obtain the information we need to set up the type (iii) moves. This takes time $\mathcal{O}(mn)$ using breadth-first search, where n and m are the number of vertices and edges in the original graph G. Then, we add to $\widetilde{G}$ all the edges corresponding to legal moves one at a time. The number of vertices in $\widetilde{G}$ is $\mathcal{O}(n^2)$. The number of type (i) and type (ii) edges in $\widetilde{G}$ is $\mathcal{O}(mn)$, since there are a type (i) and a type (ii) move for every edge and vertex combination in G. The number of type (iii) edges is $\mathcal{O}(n^2)$, making the total time to compute $\widetilde{G}$ equal to $\mathcal{O}(mn + n^2)$.

Now we just have to perform a single-source-shortest-paths computation from $\langle s, s\rangle$ to obtain H. Since we now have weights on the type (iii) edges, we cannot use breadth-first search. For a simple implementation, we could use Dijkstra's algorithm (which has a running time of $\mathcal{O}(|\widetilde{V}|^2)$) and achieve a running time of $\mathcal{O}(n^4)$. Using Fibonacci heaps [FT87], which allow single-source shortest paths in time $\mathcal{O}(|\widetilde{E}| + |\widetilde{V}| \log |\widetilde{V}|)$, we achieve a running time of $\mathcal{O}(mn + n^2 \log n)$.

As an aside, this algorithm can also be used to solve 2-DSN. Given a graph G and two node-pairs (s_1, t_1), (s_2, t_2), add two nodes s, t and edges $s \to s_1$, $t_1 \to t$, $t \to s_2$, $t_2 \to s$ to the graph and solve 2-SCSS for the two terminals s, t. The solution for this problem is also an optimal solution for the original 2-DSN problem (if we omit s and t). This leads to a running time of $\mathcal{O}(mn + n^2 \log n)$ for 2-DSN, which is the same as the running time obtained by Natu and Fang [NF95].

In fact, we can do a little better for unweighted graphs G by using more advanced shortest-paths algorithms. For a more detailed analysis, see section 6.

2.3. Correctness. The proof that our algorithm actually solves 2-SCSS can be split into two claims. We provide a proof here that motivates the techniques used in the general case. An alternate proof can be found in [NF95, NF97].

CLAIM 2.1. *If there is a legal move sequence from $\langle s, s\rangle$ to $\langle t, t\rangle$ with cost c, then there is a subgraph H of G of size $\leq c + 1$ that contains paths $s \rightsquigarrow t$ and $t \rightsquigarrow s$.*

Proof. If we follow a move sequence from $\langle s, s\rangle$ to $\langle t, t\rangle$, then f and b trace out paths $s \rightsquigarrow t$ and $t \rightsquigarrow s$, and this becomes our solution H. Moreover, the tokens traverse at most $c + 1$ vertices, since each vertex (except s) that we visit adds one to the cost of the move sequence. ∎

CLAIM 2.2. *Let H^* be an optimal subgraph containing paths $s \rightsquigarrow t$ and $t \rightsquigarrow s$. Then there exists a move sequence from $\langle s, s\rangle$ to $\langle t, t\rangle$ with total cost $|H^*| - 1$.*

Proof. We prove the claim by constructing a move sequence $\langle s, s\rangle \rightsquigarrow \langle t, t\rangle$, using H^* to tell us how to move the tokens.

When moving the tokens from $\langle s, s\rangle$ to $\langle t, t\rangle$, we "pay" each time we reach a new vertex in H^*. In order to achieve total cost $|H^*| - 1$ we must make sure that we pay only once for each vertex in H^*. To ensure this, we enforce one rule: After a token moves off a vertex, no other token will ever move to that vertex again. We say that a vertex becomes "dead" once a token moves from it, so that tokens are allowed to move only to vertices in H^* that are "alive." Note that the notion of dead and alive vertices is used only for the analysis; the algorithm itself never explicitly keeps track of them.

As we construct the move sequence, we maintain the invariant that there exist paths of "alive" vertices in H^* from the position of token f to t, and from t to the position of token b. This will ensure that we do not have to violate our rule to proceed with the sequence. When we reach $\langle t, t\rangle$, we will have constructed a legal move sequence and paid for each vertex in $|H^*|$ at most once. This will immediately imply the claim.

We construct the sequence in a greedy fashion. We start at position $\langle s, s\rangle$, and perform the following steps:

(a) *Move token f with type* (i) *moves.* Choose some path of alive vertices in H^* from the position of token f to t. We move token f forward along edges in that path (killing vertices along the way) using type (i) moves, until we reach t, or we reach a node x where moving token f would leave token b stranded; i.e., all paths of alive vertices from t to the position of token b go through x. We cannot move token f off of x; otherwise we would kill x and lose our invariant that there is a path of alive vertices from t to the position of token b (see Figure 2).

(b) *Move token b with type* (ii) *moves.* Choose some path of alive vertices from t to the position of token b and move the token b backward along edges of that path toward t using type (ii) moves. We proceed until we reach t, x, or some node $y \neq x$ that would leave the token f stranded if we killed it (all paths from x to t go through y).

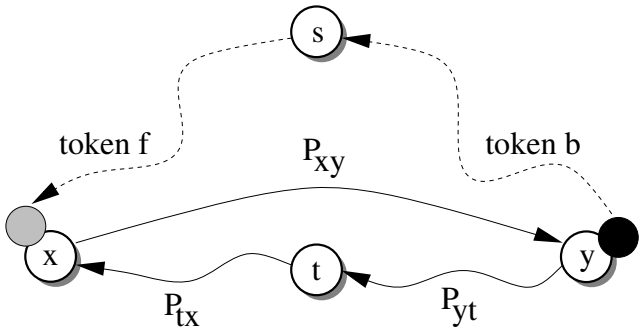


FIG. 2. *Deadlock in the construction of an optimal move sequence. Token f has moved forward onto node x, and token b has moved backward onto node y. All paths from x to t go through y, and all paths from t to y go through x. Therefore, moving either one of the tokens individually would leave the other token stranded. We resolve this deadlock using a flip: a type* (iii) *move exchanging the positions of the tokens.*

If the tokens are both on t, we are done. If the tokens are now on the same node x, go back to step (a). Note that in both these cases the last move of token b is free. If token b reaches some node $y \neq x$, we have a "deadlock" (see Figure 2).

(c) *Resolve the deadlock with a flip: A type* (iii) *move.* To resolve the deadlock, we will use a type (iii) move. We know that all paths from x to t go through y; thus there must be two disjoint simple paths P_{xy} and P_{yt} in H^* (see Figure 2). Likewise, since all paths from t to y go through x, there must be a simple path P_{tx}. Path P_{tx} must also be disjoint from P_{xy}, since if it were not, token b would be able to get from y to t without going through x.
We apply the type (iii) move $\langle x, y\rangle \rightarrow \langle y, x\rangle$. The cost of the move is at most the size of P_{xy} (not including x and y), and we kill only nodes on P_{xy}. Since P_{xy} is disjoint from both P_{tx} and P_{yt}, we maintain our invariant that there are paths of "alive" vertices from the position of token f to t and from t to the position of token b. We continue with step (a).

We never move a token onto a dead vertex, and each step maintains the invariant that there are paths of alive vertices from the position of token f to t and from t to the position of token b. Therefore both tokens reach t, and we pay only once for each vertex we visit. Since we visit only vertices in H^* and do not pay for s, it must be the case that the cost of the move sequence is at most $|H^*| - 1$. □

This claim immediately implies that the shortest path in $\widetilde{G}$ will correspond to an optimal solution, since by Claim 2.1 and the minimality of H^* there are no paths in $\widetilde{G}$ of length less than $|H^*| - 1$.

The token movements for the $p = 2$ case essentially describe the way in which paths are shared in the optimal solution. Path sharing for $p \geq 3$ is more complex, and thus we will need a richer set of token moves and a more involved proof.

3. Strongly connected Steiner subgraphs. In this section we give an algorithm for p-SCSS that is a generalization of the algorithm for 2-SCSS given in the previous section.

Again, we will use token movements to trace out the solution H. The way the tokens move is motivated by the following observation. Consider any strongly connected H containing $\{s_1, \ldots, s_p\}$. This H will contain paths from each $s_1, \ldots, s_{p-1}$ to

s_p, and these paths can be chosen to form an in-tree rooted at s_p; we will call this tree the *forward tree*. The graph H will also contain paths from s_p to each $s_1, \ldots, s_{p-1}$, forming an out-tree that we call the *backward tree*. Moreover, every H that is the union of two such trees is a feasible solution to our p-SCSS instance.

For ease of notation, we set $q := p - 1$ for the remainder of this section and let $r := s_p$, as s_p plays the special role of "root" in the two trees.

3.1. Token moves for *p*-SCSS. To trace out the two trees, we will have q "F-tokens" moving forward along edges in the forward tree from $\{s_1, \ldots, s_q\}$ to r, and q "B-tokens" moving backward along edges from $\{s_1, \ldots, s_q\}$ to r. Given a set of legal moves, we will again look for the lowest cost move sequence that moves all tokens to r. This will then correspond to the smallest subgraph containing paths $s_i \rightsquigarrow r$ and $r \rightsquigarrow s_i$ for all $i \leq q$, which is the graph we are looking for.

Since both sets of tokens trace out a tree, once two tokens of the same kind reach a vertex, they will travel the same way to the root. In that case, we will simply merge them into one token. It is therefore enough to describe the positions of the tokens by a pair of sets $\langle F, B \rangle$, where F and B are the sets of nodes currently occupied by the F- and B-tokens, respectively.

Again, we have three types of legal token moves. Type (i) moves correspond to F-tokens moving forward along an edge, and type (ii) moves correspond to B-tokens moving backward along an edge. We do not charge for entering a vertex if another token is already on it.

Since there are at most q tokens of each type, the possible token positions for a particular type are the subsets of V of size at most q. For the following, let $\mathcal{P}_q(V)$ be the set of subsets of V of size at most q.

(i) *Single moves for F-tokens*: For every edge $(u, v) \in E$ and all token sets $F \in \mathcal{P}_q(V)$, $B \in \mathcal{P}_q(V)$ such that $u \in F$, the following is a legal move:

$$\langle F, B \rangle \xrightarrow{c} \langle (F \setminus \{u\}) \cup \{v\}, B \rangle,$$

where the cost c of the move is 1 if $v \notin F \cup B$, and 0 otherwise.

(ii) *Single moves for B-tokens*: For every edge $(u, v) \in E$ and all token sets $F \in \mathcal{P}_q(V)$, $B \in \mathcal{P}_q(V)$, such that $v \in B$, the following is a legal move:

$$\langle F, B \rangle \xrightarrow{c} \langle F, (B \setminus \{v\}) \cup \{u\} \rangle,$$

where the cost c of the move is 1 if $u \notin F \cup B$, and 0 otherwise.

Type (iii) moves allow tokens to pass each other, similar to the type (iii) moves in the previous section, except that this time the "flip" is more complex (see Figure 3). We have two "outer" tokens, f and b, trying to pass each other. Between f and b there are other F-tokens moving forward and trying to pass b, and B-tokens moving backward and trying to pass f. These tokens, sitting on node sets F' and B', are "picked up" during the flip.

(iii) *Flipping*: For every pair of vertices f, b and vertex sets F, B, $F' \subset F$, $B' \subset B$, such that

- $f \in F$, $F \in \mathcal{P}_q(V)$,
- $b \in B$, $B \in \mathcal{P}_q(V)$, and
- there is a path in G from $f \rightsquigarrow b$ going through all vertices in $F' \cup B'$,

the following is a legal token move:

$$\langle F, B \rangle \overset{|M|}{\rightarrow} \langle (F \setminus (\{f\} \cup F')) \cup \{b\}, (B \setminus (\{b\} \cup B')) \cup \{f\} \rangle,$$

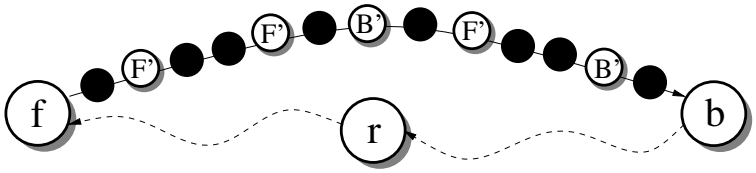


FIG. 3. *Flipping f and b, with tokens F' and B' that need to be "picked up." The black nodes are the set M.*

where M is the set of vertices on a shortest path from f to b in G going through all vertices in $F' \cup B'$, excluding f, b, and the vertices in $F' \cup B'$.

3.2. The algorithm for p-SCSS. We can now state the algorithm for p-SCSS:

1. Construct a game-graph $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ from $G = (V, E)$. Set $\widetilde{V} := \mathcal{P}_q(V) \times \mathcal{P}_q(V)$, the possible positions of the token sets, and $\widetilde{E} :=$ all legal token moves defined above.
2. Find a shortest path P in $\widetilde{G}$ from $\langle \{s_1, \ldots, s_q\}, \{s_1, \ldots, s_q\} \rangle$ to $\langle \{r\}, \{r\} \rangle$.
3. Let H be the union of $\{s_1, \ldots, s_q, r\}$ and all nodes given by P (including those in sets M for type (iii) moves).

The difficult part of constructing the game-graph $\widetilde{G}$ is computing the costs for the type (iii) moves that flip f and b. We need to know the size of the shortest path between f to b in G going through all vertices in $F' \cup B'$. Note that we do not require this path to be simple. Thus, if we knew the order in which the vertices in F' and B' occurred on the path, computing the shortest path becomes easy: It is just the union of the shortest paths between consecutive nodes in that order. Since the number of tokens in $F' \cup B'$ is bounded by $2(q-1)$, which is a constant, we can simply try all possible permutations of the nodes in $F' \cup B'$.

The total running time of the algorithm is $\mathcal{O}(mn^{2p-3} + n^{2p-2} \log n)$. For more details on the running time, see section 6.

3.3. An example. As an example we look at how the token game works on the graph in Figure 4. Our terminals are s_1, s_2, s_3, s_4, s_5, and so we set s_5 to be the root vertex and put a forward and a backward token on each of the other terminals (Figure 4(a)). In Figures 4(b)–(i) we see the following move sequence:

$$
\begin{aligned}
&\langle \{s_1, s_2, s_3, s_4\}, \{s_1, s_2, s_3, s_4\} \rangle \\
&\xrightarrow[\text{(i)}]{1} \langle \{s_1, s_2, s_3, x_3\}, \{s_1, s_2, s_3, s_4\} \rangle \\
&\xrightarrow[\text{(ii)}]{0} \langle \{s_1, s_2, s_3, x_3\}, \{s_1, s_2, s_3\} \rangle \xrightarrow[\text{(iii)}]{2} \langle \{s_2\}, \{x_3\} \rangle \\
&\xrightarrow[\text{(i)}]{1} \langle \{x_5\}, \{x_3\} \rangle \xrightarrow[\text{(ii)}]{1} \langle \{x_5\}, \{x_4\} \rangle \xrightarrow[\text{(iii)}]{0} \langle \{x_4\}, \{x_5\} \rangle \\
&\xrightarrow[\text{(i)}]{1} \langle \{s_5\}, \{x_5\} \rangle \xrightarrow[\text{(ii)}]{0} \langle \{s_5\}, \{s_5\} \rangle.
\end{aligned}
$$

The total cost of the moves is 6 and therefore equal to $|H| - q = 10 - 4 = 6$, as expected. The solution $\{s_1, s_2, s_3, s_4, s_5, x_1, x_2, x_3, x_4, x_5\}$ (Figure 4(j)) is made up of the terminals $\{s_1, s_2, s_3, s_4, s_5\}$, the nodes $\{x_3, x_4, x_5\}$ mentioned in the sequence of moves, and the nodes $\{x_1, x_2\}$ in the set M for the first type (iii) move. This is optimal.

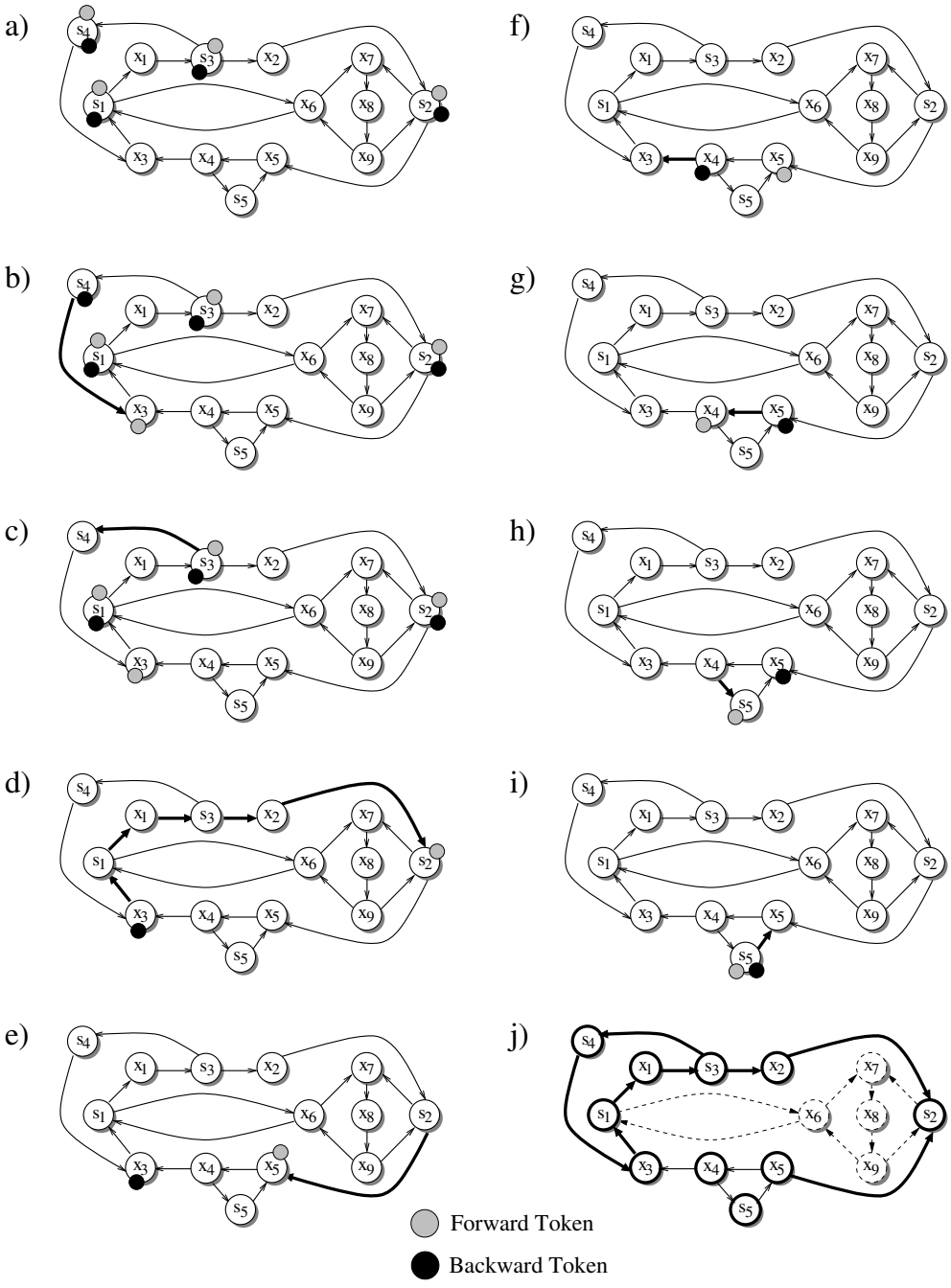


FIG. 4. *An example for the p-SCSS algorithm. Bold lines indicate the path of the previous token move.*

3.4. Correctness of the *p*-SCSS algorithm. The correctness proof for our p-SCSS algorithm can be split into the same two parts we used for 2-SCSS.

LEMMA 3.1. *Suppose there is a move sequence from* $\langle\{s_1,\ldots,s_q\},\{s_1,\ldots,s_q\}\rangle$ *to* $\langle\{r\},\{r\}\rangle$ *with total cost* c*. Then there exists a solution* H *to this* p-SCSS *instance of*

size $\leq c + q$. *Moreover, given the move sequence, it is easy to construct such an* H.

Proof. This follows directly from the definition of the moves. The cost of any move sequence is an upper bound on the number of vertices traversed by that sequence. Given the constructive nature of the moves, it is also easy to actually find H. ∎

Together with the following, much more involved lemma, the correctness of the algorithm is proved.

LEMMA 3.2. *Suppose* $H^* = (V^*, E^*)$ *is any minimal feasible solution. Then there is a move sequence from* $\langle \{s_1, \ldots, s_q\}, \{s_1, \ldots, s_q\} \rangle$ *to* $\langle \{r\}, \{r\} \rangle$ *with weight equal to* $|H^*| - q$.

Proof. To prove this lemma, we will effectively construct such a move sequence, where all intermediate positions of the tokens will be in H^*.

When moving the F- and B-tokens from $\{s_1, \ldots, s_q\}$ to r, we "pay" each time we reach a new vertex. To account for each move and achieve a total cost of $|H^*| - q$, we will use the same method as for 2-SCSS. We enforce the same rule: Once a token moves off a vertex, no other token will ever move to that vertex again. As before, we say that a vertex becomes "dead" once a token moves from it, so that tokens are allowed to move only to vertices in H^* that are "alive." This also ensures that our move sequence will be finite, since no token can return to a vertex it has already visited.

We say that a token t *requires* a vertex $v \in V^*$ if all legal paths for t to get to r pass through v. By "legal paths" we mean paths that are within H^*, go in the appropriate direction for the token t, and do not include any dead vertices. We will sometimes speak of tokens requiring tokens; in this case we mean that the first token requires the vertex on which the second token is sitting. Note that the requirement relation among tokens moving in the same direction is transitive; i.e., if f_1 requires f_2, and f_2 requires x, then f_1 also requires x.

We will construct our move sequence in a greedy fashion. That is, we will move tokens toward r using type (i) and (ii) moves, until each token sits on a vertex that is required by some other token to get to r. In this case we cannot apply any more type (i) or (ii) moves—doing so would leave some other token stranded as it is not allowed to move onto the now dead vertex.

In this case we need to use a type (iii) move to resolve the deadlock. Showing that this is always possible is the core of the correctness proof, the "flip lemma" shown in section 4. To state this lemma and see how it implies the correctness of the algorithm, we have to introduce some additional notation.

Let the "F_0-tokens" be the F-tokens that are not required by any other F-token. Similarly, let the "B_0-tokens" be the B-tokens that are not required by any other B-token. Note that since the requirements among the F-tokens are not cyclic, there will be at least one F_0-token and, similarly, at least one B_0-token.

LEMMA 3.3 (the flip lemma). *Suppose every token is required by some other token. Then there is an* F_0*-token* f *and a* B_0*-token* b *such that*

- f *requires* b*, and no other* F_0*-token requires* b*,*
- b *requires* f*, and no other* B_0*-token requires* f*.*

We will prove this lemma in the next section. Let us now see how it concludes the proof of Lemma 3.2.

Let f and b be chosen according to the flip lemma. Fix any simple path from f to r that uses only live vertices. Call P the portion of this path between f and b, and Q the portion between b and r (see Figure 5). By definition, P and Q are disjoint.

CLAIM 3.4. *All tokens that require a vertex on* P *are on* P *themselves.*

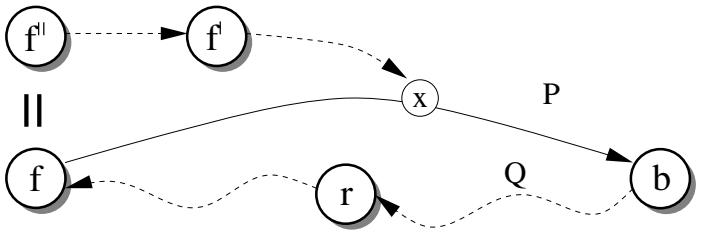


FIG. 5. *Showing Claim* 3.4*, that all tokens that require a vertex on* P *are on* P *themselves.*

Proof. We prove the claim for F-tokens; a symmetric argument applies to B-tokens. Suppose some F-token $f' \neq f$ requires a vertex x on P. Every path $x \rightsquigarrow r$ must include b; otherwise f could move to x and then to r, without visiting b. Therefore, f' also requires b (see Figure 5). The token f' cannot be an F_0-token, since the flip lemma tells us that f is the only F_0-token that requires b. Note that due to transitivity, every F-token is either an F_0-token or required by some F_0-token, so f' must be required by some F_0-token f''. By transitivity, f'' requires b, and thus $f'' = f$, by the flip lemma.

Since $f = f''$ requires f', the token f' is either on P or Q. If f' is on Q, then x is also on Q, since f' requires x. This contradicts the fact that P and Q are disjoint, and thus f' must be on P. □

Let F' be the set of F-tokens that are on the path P, and let B' be the set of B-tokens on P. We can apply a type (iii) move that switches f and b and picks up F' and B' along the way. All vertices on P become dead. No token is stranded, since by Claim 3.4, all tokens that required a vertex on P were on P and therefore were picked up by the flip.

We have shown, pending the flip lemma, that each step of the construction of our move sequence preserves paths of "alive" vertices from the F-tokens to r and from r to the B-tokens, and never moves a token onto a dead vertex. This shows that we can always continue the construction of our move sequence until all tokens reach r, and that the cost of the move sequence will be no more than $|H^*| - q$. □

4. The flip lemma. To complete the proof of correctness, it remains to prove Lemma 3.3, the flip lemma. We prove the flip lemma by making a graph out of the requirement relationships between the F_0- and B_0-tokens. We show that there is a two-cycle in this graph, consisting of an F_0- and a B_0-token, that does not have any other incoming requirements. This proves the lemma.

During the discussion, keep in mind that transitivity holds only among requirements for the same type of token; if an F-token f requires a node (or a token) x and some F-token f' requires f, then f' requires x. However, if some B-token b requires f, it is not necessarily the case that b also requires x, since b is moving backward along edges.

4.1. Proof of Lemma 3.3 (the flip lemma). Let $G_{req} = (V_{req}, E_{req})$ be a new directed graph, whose nodes are the F_0- and B_0-tokens. The edges in E_{req} correspond to requirements: G_{req} has an edge $x \to y$ if and only if the token x requires the token y.

This graph has a lot of structure. First of all, G_{req} is bipartite, since no two F_0-tokens (and no two B_0-tokens) have a requirement relationship (by the definition of an F_0- and a B_0-token). For the remainder of the discussion, we will use f to denote a node on the F_0 side of the bipartition and b to denote a node on the B_0 side.

By assumption (every token is required by some other token) and by definition (an F_0-token is not required by any F-token), we know that every F_0-token is required by at least one B-token. We know that either that B-token is a B_0-token, or there is another B_0-token that requires that B-token. Therefore, by transitivity for B-tokens, every F_0-token is required by at least one B_0-token. By symmetry, every B_0-token is required by at least one F_0-token. Therefore, every node in G_{req} has at least one incoming edge.

We want to find a two-cycle in G_{req} with no incoming edges, since the two tokens in such a cycle would require each other but would not be required by any other token, proving the lemma. We can view G_{req} as a dag (directed acyclic graph) of strongly connected components and sort the strongly connected components topologically. Let C be the first component in that ordering. This means that no token outside of C requires any token in C. Furthermore, C cannot consist of only one node, since then that token would be required by no other token, in contradiction to our assumption that every token is required by at least one token.

We will now prove that C cannot consist of more than two nodes. This gives us the desired two-cycle and shows the flip lemma. The proof rests on the observation that G_{req} satisfies a kind of transitivity, which we call the *projection* property. After showing his property, we prove the claim by contradiction by applying the projection property across the requirement graph.

CLAIM 4.1 (projection). *Suppose for three nodes f_1, f_2, b_1 ($f_1 \neq f_2$) in G_{req} we have edges $f_1 \to b_1$ and $b_1 \to f_2$ in G_{req}. Then the following holds: All nodes b that have an edge $b \to f_1$ also have an edge $b \to f_2$.*

Proof. By definition of F_0, there is a legal path in H^* from f_1 to r avoiding f_2. Since f_1 requires b_1, this path goes through b_1. Therefore, there is a path P_1 from f_1 to b_1 avoiding f_2 (see Figure 6).

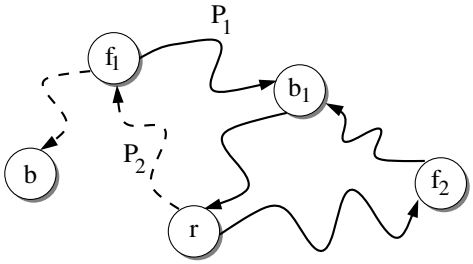


FIG. 6. *Proving the projection property in G_{req}. The solid lines are paths in H^* corresponding to edges $f_1 \to b_1$ and $b_1 \to f_2$ in G_{req}; the dashed line corresponds to the edge $b \to f_1$.*

Suppose that some node b in G_{req} has an edge $b \to f_1$. We show that $b \to f_2$ is also in G_{req} by contradiction. If $b \to f_2$ is not in the requirement graph, then there is a legal path in H^* from r to b avoiding f_2. Since b requires f_1, this path goes through f_1. Therefore, there is a path P_2 in H^* from r to f_1 avoiding f_2. Combining P_2 and P_1, we obtain a path from r to b_1 that does not visit f_2, in contradiction to $b_1 \to f_2$ being in G_{req}. ∎

A symmetric property holds by exchanging f's and b's; i.e., for any triple f_1, b_1, b_2 ($b_1 \neq b_2$), if there are edges $b_1 \to f_1$ and $f_1 \to b_2$ in G_{req}, then for every node f in G_{req}, if there is an edge $f \to b_1$, then there must also be an edge $f \to b_2$.

The projection property has a profound effect on the structure of G_{req}. In fact, it shows that if there is a path in G_{req} from a node b to a node f, then there is an edge

from b to f. To see this, consider the last four nodes on the path from b to f in G_{req}, $b'' \to f' \to b' \to f$. By projection, there must be an edge $b'' \to f$. This shortens the length of the path by 2. This argument can be repeated along the path until it shows that there must be an edge $b \to f$.

A symmetric argument (using the symmetric projection property) shows that if there is a path in G_{req} from a node f to a node b, then there is an edge from f to b. We further conclude that every strongly connected component in G_{req} is a complete bipartite graph (every pair of nodes on opposite sides of the bipartition is connected in both directions). Now we are ready to finish the flip lemma.

CLAIM 4.2. *No strongly connected component C of G_{req} has more than two nodes.*

Proof. We prove the claim by contradiction. Assume that a strongly connected component C in G_{req} has at least three elements. Either there are two F_0 nodes in C, or two B_0 nodes in C. We assume (without loss of generality) that there are two F_0 nodes (a symmetric argument shows the other case). Since C is a complete bipartite graph, there must be a complete bipartite subgraph of C consisting of nodes f_1, b_1, f_2.

We turn back to H^* to show that this structure cannot exist. Since b_1 requires both f_1 and f_2, there is a legal path in H^* from r to b_1 that visits both f_1 and f_2 (solid lines in Figure 7). Without loss of generality assume that f_1 is the first node on that path, so that there is a path P_1 from r to f_1 that avoids f_2. Since f_1 requires b_1, but f_1 does not require f_2, there must also be a path P_2 from f_1 to b_1 that avoids f_2 (dashed lines in Figure 7). Combining P_1 and P_2, we obtain a legal path in H^* from r to b_1 that avoids f_2, in contradiction to the fact that b_1 requires f_2.

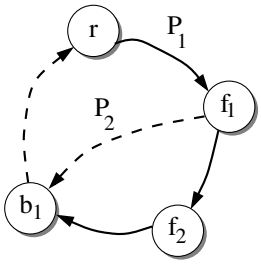


FIG. 7. *Components with more than two elements are impossible.*

This contradiction shows that C cannot have more than two nodes. This shows the claim and thereby the flip lemma. ∎

5. The directed Steiner network problem. In this section we show how to apply the algorithm developed in the previous sections to solve the DIRECTED STEINER NETWORK problem (p-DSN), for any constant p.

p-DSN (unweighted, node-minimizing). *Given a directed graph $G = (V, E)$ and p pairs of nodes in the graph $\{(s_1, t_1), \ldots, (s_p, t_p)\}$, find the subgraph H of G with the smallest number of nodes that contains paths from s_i to t_i for $1 \le i \le p$.*

We use the same general model of a token game, but now we have tokens moving from each source s_i to its destination t_i. This time, we have no backward moving tokens, and also tokens do *not* merge when they reach the same node. We describe the positions of the tokens by a p-tuple $\langle f_1, f_2, \ldots, f_p \rangle$. We have two kinds of moves for the tokens. The first kind of move allows a single token to move one step along an edge.

(i) For each edge (u, v) we include the moves $\langle —, u, — \rangle \xrightarrow{c} \langle —, v, — \rangle$, meaning that one token moves from u to v, and all others remain where they are. The cost c of the move is 0 if v already has a token on it, and 1 otherwise.

Using only type (i) token moves, it is easy to see that we can exactly trace out any dag. But since the optimal solution to p-DSN may contain cycles (or other more complex strongly connected components), we need to define additional token moves so as not to overcount.

Consider the optimal solution to p-DSN and contract every strongly connected component to a single node; the resulting graph is a dag (see Figure 8). Now imagine a single token moving through this graph along a path to its destination; for each strongly connected component it encounters along the way, it has an entrance point and an exit point. A particular strongly connected component in this decomposition has some number $k \leq p$ such entrance/exit point pairs.

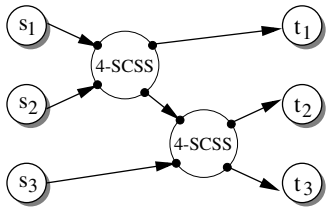


FIG. 8. *A solution to* p-DSN *is a dag of strongly connected components.*

Using this intuition, we define a special token move that allows a group of k tokens ($k \leq p$) to move from any k or fewer specific entrance points to any k or fewer specific exit points in a strongly connected component (if such a component exists). The cost of the move will be (roughly) the size of the smallest component containing these special entrance/exit points. This cost can be computed using our algorithm for SCSS from section 3.

Formally, our token moves are defined as follows.

(ii) For all $k \leq p$ and for every set of k node-pairs $\{(f_1, x_1), (f_2, x_2), \ldots, (f_k, x_k)\}$ for which there is a strongly connected subgraph of G containing the f_i and x_i for all $1 \leq i \leq k$, we allow the move

$$(— f_1 — f_2 — \cdots — f_k —)$$
$$\xrightarrow{c} (— x_1 — x_2 — \cdots — x_k —).$$

The cost c of this move is the size of the smallest strongly connected component containing the vertices $\{f_1, f_2, \ldots, f_k, x_1, x_2, \ldots, x_k\}$ minus the size of the set $\{f_1, \ldots, f_k\}$. We can use the algorithm developed in section 3 to compute this cost.

Similar in structure to our algorithm for p-SCSS in section 3, the algorithm for p-DSN consists of the following steps.

1. Compute the game-graph $\mathcal{G}$, where the vertices in $\mathcal{G}$ are p-tuples of vertices in the input graph G, and edges are included for each legal token move (type (i) or (ii)).
2. Find the minimum-weight path P in $\mathcal{G}$ from $\langle s_1, \ldots, s_p \rangle$ to $\langle t_1, \ldots, t_p \rangle$.
3. Output the subgraph H of G induced by P, i.e., the subgraph containing

- all vertices of G explicitly "mentioned" by vertices in P, and
- for all type (ii) moves used in P, all the vertices making up the smallest strongly connected component containing the f_i's and x_i's used to define that move.

5.1. Correctness. As for the previous algorithms, it is easy to see that for any move sequence from $\langle s_1, \ldots, s_p \rangle$ to $\langle t_1, \ldots, t_p \rangle$ of cost c, there is a feasible solution H of size at most $c + |\{s_1, \ldots, s_p\}|$. It is also easy to find this H, given the move sequence. The following lemma then implies the correctness of the algorithm.

LEMMA 5.1. *Let H^* be a minimum size subgraph of G that contains paths $s_i \rightsquigarrow t_i$ for all $i \in \{1, \ldots, p\}$. Then there is a legal sequence of token moves from $\langle s_1, \ldots, s_p \rangle$ to $\langle t_1, \ldots, t_p \rangle$ with cost $|H^*| - |\{s_1, \ldots, s_p\}|$.*

Proof. We again give a constructive proof. We start with tokens $f_1, \ldots, f_p$ at $s_1, \ldots, s_p$ and move them to their respective destinations $t_1, \ldots, t_p$.

Regard each strongly connected component in H^* as a single node and topologically sort this dag of strongly connected components. Let $C_1, \ldots, C_m$ be the resulting order of strongly connected components. We now consider each component in order and move each token in the component either to its destination (if its destination is in the component) or to some later component in the ordering on a path to its destination. After doing so, all nodes in the component are dead. This ensures that we pay only once for every node.

For each component C_i containing some k tokens ($k \leq p$), we perform the following moves. We execute (a) and (b) if C_i consists of more than one node, and only (b) if C_i consists of a single node.

(a) We apply a type (ii) move. For each token f_ℓ in C_i we define a node x_ℓ in C_i to which it moves. For tokens f_ℓ whose destination t_ℓ is in C_i, we set x_ℓ to that destination. For all other tokens f_ℓ we choose any legal path to its destination t_ℓ and let x_ℓ be the last node of that path that is in C_i. Using a type (ii) move we simultaneously move all the tokens f_ℓ to their respective x_ℓ.

(b) We apply a type (i) move for each token f_ℓ in C_i that is not yet at its destination t_ℓ. We move along one edge of a path to t_ℓ into a new component C_j ($j > i$). □

5.2. Weights and edges. The algorithms provided for p-DSN and p-SCSS can easily be modified to handle weighted nodes: Just make the cost of a move the total weight of the unoccupied nodes entered during the move instead of just their number.

It is also easy to minimize the total edge weight in H by reducing it to the weighted-node case. To do this, we give every vertex in G weight 0 and replace every edge e by a new vertex having the weight of e. We connect this new vertex to the two vertices incident to e. Naturally, it is also possible to combine vertex weights and edge weights.

6. Runtime analysis. In this section we provide the running time analysis for our algorithms solving p-SCSS (from section 3) and p-DSN (from section 5).

The aim of this section is mainly to give an idea as to how the running time is distributed over the different parts of the algorithms (game-graph construction and shortest-path computation).

6.1. The p-SCSS algorithm. The algorithm consists of two parts: the generation of the game-graph $\widetilde{G}$ from the input $G = (V, E)$ and the computation of a shortest path from $\langle \{s_1, \ldots, s_q\}, \{s_1, \ldots, s_q\} \rangle$ to $\langle \{r\}, \{r\} \rangle$ in $\widetilde{G}$.

The size of the game-graph $\widetilde{G}$. In the following, n and m are always the number of vertices and edges, respectively, of the input graph G. Recall that we assume that G is connected, and thus $m \geq n-1$. The number of vertices in the game-graph $\widetilde{G}$ is

$$|\mathcal{P}_q(V) \times \mathcal{P}_q(V)| = \left(\sum_{i=0}^{q} \binom{n}{i}\right)^2 = \mathcal{O}(n^{2q}).$$

The number of type (i) edges can be computed as follows. If we fix an edge $(u,v) \in E$, then there are $|\mathcal{P}_{q-1}(V \setminus \{u\})|$ choices for F and $|\mathcal{P}_q(V)|$ choices for B, and thus the total number of type (i) edges is

$$m \cdot |\mathcal{P}_{q-1}(V \setminus \{u\})| \cdot |\mathcal{P}_q(V)| = \mathcal{O}(m \cdot n^{q-1} \cdot n^q) = \mathcal{O}(mn^{2q-1}).$$

By symmetry, the number of type (ii) edges is the same.

For the type (iii) edges, we can also obtain an upper bound on their number by multiplying the number of choices for f and b ($\mathcal{O}(n)$ each), F and B ($\mathcal{O}(n^{q-1})$ each), and F' and B' ($\mathcal{O}(2^{q-1})$ each after choosing F and B). This yields a bound of $\mathcal{O}(n^{2q})$. Since $m \geq n-1$, the total number of edges is $\mathcal{O}(mn^{2q-1})$. The number of edges in $\widetilde{G}$ therefore is not much larger than the number of nodes.

Constructing the edge weights of the game-graph $\widetilde{G}$. Computing the edge weights takes constant time for type (i) and (ii) edges but is slightly more expensive for type (iii) edges. We need to compute, for each type (iii) edge, the length of a shortest path from a node f to a node b going through at most $2q-2$ specific intermediate nodes (the ones in $F' \cup B'$). Since this path does not have to be simple, it will be the union of the shortest paths between the consecutive intermediate vertices. Since there are only a constant number of intermediate vertices, we can guess their order.

More formally, we run an all-pairs-shortest-paths algorithm on the input graph G; this takes time at most $\mathcal{O}(n^2 \log n + mn)$. Now, for each type (iii) edge, we go through all possible sequences in which the vertices in $F' \cup B'$ could appear on the path. For each sequence, we add together the shortest path distances for consecutive vertices to compute the total cost of the best path for that sequence. The shortest path length among all sequences is kept as the weight of the type (iii) edge. There are $\mathcal{O}((2q-2)!)$ possible sequences of the vertices in $F' \cup B'$, and adding together the distances takes time $\mathcal{O}(2q)$. So as long as p (and therefore q) is constant, the time to compute the weight of a type (iii) edge is also constant.

To summarize, we spend a constant amount of time to compute the weight of each edge in the graph, which leads to a total time of $\mathcal{O}(mn^{2q-1})$ for the game-graph construction—subsuming the time for the all-pairs-shortest-path computation. Note that this bound also holds for the case where the original graph has arbitrary weights.

Computing the shortest path in the game-graph $\widetilde{G}$. The second part of the algorithm is to compute a single-source-shortest-path query in the game-graph $\widetilde{G} = (\widetilde{V}, \widetilde{E})$. Since all known shortest-paths algorithms might have to look at every edge, this step will dominate the running time of the algorithm. There exist a variety of different shortest-paths algorithms [AMO93], and the best one to use for a given graph depends on n, m, and C, where C is the largest weight in the graph if all weights are positive integers. For the case where the input graph G is weighted with positive real numbers, we want an algorithm that does not depend on C. Thus, the best choice is to use Fibonacci heaps [FT87], which allow us to compute single-source shortest paths in time $\mathcal{O}(|\widetilde{E}| + |\widetilde{V}| \log |\widetilde{V}|)$, and the total running time of our algorithm is

$$\mathcal{O}(n^{2q} + mn^{2q-1} + n^{2q} \log n) = \mathcal{O}(mn^{2p-3} + n^{2p-2} \log n).$$

If the input graph has no weights, as is the case for most of this paper, the weights in the game-graph will be positive integers bounded by n. So, we can use an algorithm of Ahuja et al. [AMOT90] that performs a shortest-path query in time $\mathcal{O}(|\widetilde{E}| + |\widetilde{V}|\sqrt{\log(|\widetilde{V}|C)})$, where C is the largest weight in the game-graph. Therefore the running time for the unweighted p-SCSS algorithm is

$$\mathcal{O}(mn^{2q-1} + n^{2q}\sqrt{\log n}) = \mathcal{O}(mn^{2p-3} + n^{2p-2}\sqrt{\log n}).$$

6.2. The *p*-DSN algorithm. For this algorithm, the game-graph $\mathcal{G}$ consists of $\mathcal{O}(n^p)$ nodes and can therefore have up to $\mathcal{O}(n^{2p})$ edges. This means that the final shortest-path computation will take time at most $\mathcal{O}(n^{2p})$, using Dijkstra's algorithm. We are deliberately rough with this calculation since it turns out that for the p-DSN algorithm, the time to construct the game-graph actually overshadows this shortest-path computation.

The most time-consuming part of the game-graph construction is to determine the weights of the type (ii) edges. Obviously, it would be very inefficient (though still polynomial) to call our k-SCSS algorithm for every type (ii) edge in the game-graph. Fortunately, a simple observation makes it possible to avoid that.

Solving the instances of k-SCSS *efficiently.* Every instance of k-SCSS that we wish to solve constructs a game-graph from the same underlying graph G. Furthermore, this game-graph does not depend on which vertices are terminals but only on G and the number k. This is because the game-graph describes legal token moves, which are independent of the starting positions of the tokens and their goals. Therefore, all instances of k-SCSS that operate on the same underlying graph G and have the same number of terminals use the same game-graph. Let us call this game-graph $\widetilde{G}_k$.

We need to solve instances of k-SCSS for all $k \leq 2p$. But, notice that $\widetilde{G}_k$ is a subgraph of $\widetilde{G}_{2p}$ if $k \leq 2p$. Moreover, there are no edges from this subgraph $\widetilde{G}_k$ to any other vertices in $\widetilde{G}_{2p}$, since all other vertices $\langle F, B\rangle$ in $\widetilde{G}_{2p}$ have more than k nodes in either F or B. Thus, we need only construct a single game-graph $\widetilde{G}_{2p}$ to determine all the weights of the type (ii) edges. However, we need to perform several different shortest-path computations on this graph for the different terminal sets.

Fortunately, all these computations have something in common. Solving a k-SCSS instance requires computing a shortest path in $\widetilde{G}_{2p}$, to a node of the form $\langle\{r\}, \{r\}\rangle$. This suggests the following strategy: Run n single-destination shortest-path computations, one for each possible destination $\langle\{r\}, \{r\}\rangle$ $(r \in V)$. Now the weights of type (ii) edges can then be computed in constant time by looking up the appropriate shortest path length.

The number of nodes in $\widetilde{G}_{2p}$ is $\mathcal{O}(n^{4p-2})$, and the number of edges is $\mathcal{O}(mn^{4p-3})$. Using Fibonacci heaps to perform each of the n shortest-path computations, we obtain a running time of $\mathcal{O}(mn^{4p-2} + n^{4p-1}\log n)$. In the unweighted case, the maximum weight in $\widetilde{G}_{2p}$ is again at most n. Thus, we use the algorithm of Ahuja et al. [AMOT90] and obtain a running time of $\mathcal{O}(mn^{4p-2} + n^{4p-1}\sqrt{\log n})$.

7. Conclusion. We have developed a polynomial time algorithm that computes the smallest subgraph containing paths between $p = O(1)$ pairs of nodes in a directed graph. It is an interesting open question whether p-DSN is fixed-parameter tractable, i.e., whether there is an algorithm whose running time is $O(n^k)$ for some k that does not depend on the constant p. We also wonder whether the tools developed to obtain our result can be used to construct improved approximation algorithms for arbitrary p, or for the closely related Directed Steiner Tree and Minimum Equivalent Digraph problems.

Acknowledgments. We would like to thank David Karger for helpful suggestions and Andras Frank for asking about the 2-SCSS problem, which started our research on this topic. We are grateful to Marshall Bern, Yevgeniy Dodis, John Dunagan, and Matt Levine for their comments on a previous version of this paper. We also thank the anonymous reviewers for their helpful comments.

REFERENCES

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice–Hall, Englewood Cliffs, NJ, 1993.

[AMOT90] R. K. Ahuja, K. Mehlhorn, J. B. Orlin, and R. E. Tarjan, *Faster algorithms for the shortest path problem*, J. ACM, 37 (1990), pp. 213–223.

[CCC+98] M. Charikar, C. Chekuri, T.-y. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li, *Approximation algorithms for directed Steiner problems*, in Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), San Francisco, 1998, pp. 192–200.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed., MIT Press, Cambridge, MA, 2001.

[DK99] Y. Dodis and S. Khanna, *Designing networks with bounded pairwise distance*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC), 1999, pp. 750–759.

[DW71] S. E. Dreyfus and R. A. Wagner, *The Steiner problem in graphs*, Networks, 1 (1971), pp. 195–207.

[Fel00] J. Feldman, *The Directed Steiner Network Problem Is Tractable for a Constant Number of Terminals*, Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2000.

[FR99] J. Feldman and M. Ruhl, *The directed Steiner network problem is tractable for a constant number of terminals*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1999, pp. 299–308.

[FT87] M. L. Fredman and R. E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. ACM, 34 (1987), pp. 596–615.

[Hak71] S. L. Hakimi, *Steiner's problem in graphs and its implications*, Networks, 1 (1971), pp. 113–133.

[Lev71] A. Levin, *Algorithm for the shortest connection of a group of graph vertices*, Soviet Math. Dokl., 12 (1971), pp. 1477–1481.

[LMS90] J. M. Y. Leung, T. L. Magnanti, and V. Singhal, *Routing in point-to-point delivery systems: Formulations and solution heuristics*, Transportation Sci., 24 (1990), pp. 245–260.

[LMS92] C.-L. Li, S. T. McCormick, and D. Simchi-Levi, *The point-to-point delivery and connection problems: Complexity and algorithms*, Discrete Appl. Math., 36 (1992), pp. 267–292.

[MW84] T. L. Magnanti and R. T. Wong, *Network design and transportation planning: Models and algorithms*, Transportation Sci., 18 (1984), pp. 1–55.

[NF95] M. Natu and S.-C. Fang, *On the point-to-point connection problem*, Inform. Process. Lett., 53 (1995), pp. 333–336.

[NF97] M. Natu and S.-C. Fang, *The point-to-point connection problem—Analysis and algorithms*, Discrete Appl. Math., 78 (1997), pp. 207–226.

[Ram96] S. Ramanathan, *Multicast tree generation in networks with asymmetric links*, IEEE/ACM Trans. Networking, 4 (1996), pp. 558–568.

[SRV97] H. F. Salama, D. S. Reeves, and Y. Viniotis, *Evaluation of multicast routing algorithms for real-time communication on high-speed networks*, IEEE J. Select. Areas Commun., 15 (1997), pp. 332–345.

SIAM J. COMPUT.
Vol. 36, No. 3, pp. 563–594

TIME-SPACE LOWER BOUNDS FOR THE POLYNOMIAL-TIME HIERARCHY ON RANDOMIZED MACHINES*

SCOTT DIEHL† AND DIETER VAN MELKEBEEK†

Abstract. We establish the first polynomial-strength time-space lower bounds for problems in the linear-time hierarchy on randomized machines with two-sided error. We show that for any integer $\ell > 1$ and constant $c < \ell$, there exists a positive constant d such that QSAT_ℓ cannot be computed by such machines in time n^c and space n^d, where QSAT_ℓ denotes the problem of deciding the validity of a quantified Boolean formula with at most $\ell - 1$ quantifier alternations. Moreover, d approaches $1/2$ from below as c approaches 1 from above for $\ell = 2$, and d approaches 1 from below as c approaches 1 from above for $\ell \geq 3$. In fact, we establish the stronger result that for any constants $a \leq 1$ and $c < 1 + (\ell - 1)a$, there exists a positive constant d such that linear-time alternating machines using space n^a and $\ell - 1$ alternations cannot be simulated by randomized machines with two-sided error running in time n^c and space n^d, where d approaches $a/2$ from below as c approaches 1 from above for $\ell = 2$, and d approaches a from below as c approaches 1 from above for $\ell \geq 3$. Corresponding to $\ell = 1$, we prove that there exists a positive constant d such that the set of Boolean tautologies cannot be decided by a randomized machine with one-sided error in time $n^{1.759}$ and space n^d. As a corollary, this gives the same lower bound for satisfiability on deterministic machines, improving on the previously best known such result.

Key words. time-space lower bounds, randomized algorithms, polynomial-time hierarchy, satisfiability

AMS subject classifications. 68Q17, 68Q10, 68Q15

DOI. 10.1137/050642228

1. Introduction.

Satisfiability, the problem of deciding if a propositional formula has at least one satisfying assignment, is among the most fundamental NP-complete problems. Proving lower bounds for satisfiability remains an open problem of paramount importance to the field of computational complexity. Although it is widely conjectured that any deterministic algorithm requires exponential time to solve satisfiability, a proof of this belief seems far out of reach. The trivial linear-time lower bound, which follows from the observation that the machine must look at its entire input formula in the worst case, is the state-of-the-art bound for random-access machines. Despite several decades of effort, there has been no success in proving superlinear time lower bounds for satisfiability.

A few years ago, Fortnow [8] established nontrivial time lower bounds for satisfiability on machines which are restricted to using a small amount of workspace. Fortnow's technique has its roots in earlier work by Kannan [14] and has been further developed in recent years [17, 19, 24]. For example, for machines using a subpolynomial amount of space, Fortnow and van Melkebeek [9] derived a time lower bound of $n^{\phi - o(1)}$, where $\phi \approx 1.618$ denotes the golden ratio. Recently, Williams [24] improved this lower bound to $n^{1.732}$, and we can further boost it to $n^{1.759}$ as a corollary to one of our results.

*Received by the editors October 9, 2005; accepted for publication (in revised form) May 30, 2006; published electronically August 25, 2006. A preliminary version of this work appeared as an extended abstract in [7].

http://www.siam.org/journals/sicomp/36-3/64222.html

†University of Wisconsin, Madison, WI 53706 (sfdiehl@cs.wisc.edu, dieter@cs.wisc.edu). The first author was supported by NSF Career Award CCR-0133693. The second author was partially supported by NSF Career Award CCR-0133693.

However, the main focus of this paper is not on lower bounds for deterministic machines, but rather on lower bounds for *randomized machines with two-sided error* (bounded away from 1/2). While it is conjectured that satisfiability requires exponential time even on such machines, proving lower bounds in the presence of randomness becomes a more difficult task than in the deterministic setting. No nontrivial lower bounds for satisfiability have been established on randomized random-access machines with two-sided error, even when the workspace of such machines is restricted to be logarithmic. In fact, previous to this work, no nontrivial time-space lower bounds have been shown for any complete problems in the polynomial-time hierarchy on randomized random-access machines with two-sided error.

1.1. Results. In this paper, we establish such time-space lower bounds. We consider the problem QSAT_ℓ of deciding the validity of a given quantified Boolean formula with at most $\ell-1$ quantifier alternations (beginning with existential). For any integer $\ell \geq 1$, QSAT_ℓ is complete for the ℓth level of the polynomial-time hierarchy. These problems are generalizations of satisfiability—for $\ell = 1$, QSAT_ℓ is precisely the satisfiability problem. Time-space lower bounds for QSAT_ℓ have been previously considered for deterministic machines. For example, Fortnow and van Melkebeek show an $n^{\ell-o(1)}$ time lower bound for deterministic machines solving QSAT_ℓ in subpolynomial space, for $\ell \geq 2$. We match these bounds for randomized machines running in subpolynomial space, and a more careful analysis yields lower bounds for small polynomial space bounds.

THEOREM 1 (main theorem). *For any integer $\ell \geq 2$ and constant $c < \ell$, there exists a positive constant d such that QSAT_ℓ cannot be solved by randomized random-access machines with two-sided error running in time n^c and space n^d. Moreover, d approaches 1/2 from below as c approaches 1 from above for $\ell = 2$, and d approaches 1 from below as c approaches 1 from above for $\ell \geq 3$.*

The randomized machines in Theorem 1 and in the rest of this paper refer to the natural coin flip model, in which the machine has one-way read-only access to a tape with random bits. Viola [23] recently extended Theorem 1 to the model in which the randomized machines have two-way access to the random bit tape, although his approach yields weaker lower bounds and works only for $\ell \geq 3$.

We note that Theorem 1 establishes the first polynomial-strength time-space lower bounds for problems in the polynomial-time hierarchy on two-sided error randomized machines. By time-space lower bounds of "polynomial strength" we mean time lower bounds of the form $\Omega(n^c)$ for some constant $c > 1$ under nontrivial space upper bounds. Previous works establish randomized time-space lower bounds, but either they consider problems believed not to be in the polynomial-time hierarchy, or the time lower bounds involved are only slightly superlinear. Allender et al.'s [1] time-space lower bounds for problems in the counting hierarchy on probabilistic machines with unbounded error fall within the first category. On the other hand, Beame et al.'s [3] nonuniform time-space lower bound for a binary quadratic form problem in P falls within the second category.

Because of the tight connection between QSAT_ℓ and the ℓth level of the linear-time hierarchy, Theorem 1 can be stated equivalently as a time-space lower bound for simulations of the ℓth level of the linear-time hierarchy on randomized machines with two-sided error. In fact, we can strengthen Theorem 1 and establish time-space lower bounds for simulations of linear-time alternating machines which use only space n^a for constant $a \leq 1$.

THEOREM 2. *For any integer $\ell \geq 2$ and any constants $a \leq 1$ and $c < 1 + (\ell - 1)a$, there exists a positive constant d such that linear-time alternating machines*

using space n^a and $\ell - 1$ alternations cannot be simulated by randomized random-access machines with two-sided error running in time n^c and space n^d. Moreover, d approaches $a/2$ from below as c approaches 1 from above for $\ell = 2$, and d approaches a from below as c approaches 1 from above for $\ell \geq 3$.

Note that when $a = 1$, the space restriction on the alternating machines disappears and Theorem 2 becomes Theorem 1.

The $\ell \geq 2$ restriction in Theorem 1 implies that it does not give any bounds for the first level of the polynomial-time hierarchy, i.e., for satisfiability or its complement, tautology, the problem of deciding if a propositional formula is true under all assignments. However, we are able to strengthen the known lower bounds for tautology on randomized machines with *one-sided error*. Previously, Fortnow and van Melkebeek showed a lower bound of $n^{\sqrt{2}-o(1)}$ for the tautology problem on nondeterministic machines using subpolynomial space. Since randomized machines with one-sided error (on the "yes" side) are special cases of nondeterministic machines, these bounds also apply to this setting. Using ideas from the proof of our main result, we manage to take advantage of the extra structure provided by the randomized machine and improve the known lower bounds for tautology on randomized machines with one-sided error.

THEOREM 3. *There exists a positive constant d such that tautology cannot be solved by randomized random-access machines with one-sided error running in time $n^{1.759}$ and space n^d.*

Notice that the lower bound of Theorem 3 applies to deterministic machines as a special case. Therefore, by the closure of deterministic classes under complement, Theorem 3 implies the improved lower bound for satisfiability on deterministic machines mentioned earlier.

COROLLARY 4. *There exists a positive constant d such that satisfiability cannot be solved by deterministic random-access machines running in time $n^{1.759}$ and space n^d.*

1.2. Techniques. Our proofs follow the paradigm of indirect diagonalization. This technique establishes a desired separation by contradiction—assuming the separation does not hold, we derive a sequence of progressively unlikely inclusions of complexity classes until we reach one that contradicts a known diagonalization result. Kannan [14] used the paradigm "avant la lettre" to investigate the relationship between deterministic linear time and nondeterministic linear time. All of the recent work on time-space lower bounds for satisfiability and problems higher up in the polynomial-time hierarchy [8, 17, 9, 19, 24] follow it as well. Allender et al. [1] employed the technique to establish time-space lower bounds for problems in the counting hierarchy.

At first glance, it might seem that current techniques from space-bounded derandomization let us derive time-space lower bounds on randomized machines as immediate corollaries to time-space lower bounds on deterministic machines. In particular, assuming that we can solve satisfiability on a randomized machine in logarithmic space and time n^c, Nisan's deterministic simulation [21] yields a deterministic algorithm for satisfiability that runs in polylogarithmic space and polynomial time. However, even for $c = 1$, the degree of the latter polynomial is far too large for this simulation to yield a contradiction with known time-space lower bounds for deterministic machines. Thus, we need a more delicate approach for the randomized setting.

The critical ingredient in this approach is a time- and space-efficient simulation of randomized computations in the second level of the polynomial-time hierarchy with very few guess bits. The latter follows from a careful combination of Nisan's partial space-bounded derandomization [20], deterministic amplification by a random walk on an expander [5, 13], and a version of Lautemann's proof that randomized machines

with two-sided error can be simulated by an alternating machine at a polynomial time-overhead [15]. It gives us (i) an unconditional way to speed up small-space randomized computations with two-sided error in higher levels of the polynomial-time hierarchy, and (ii) a conditional efficient complementation of computations within the ℓth level of the polynomial-time hierarchy for $\ell \geq 2$. The condition on the latter is the hypothesis of the indirect diagonalization argument that the ℓth level of the linear-time hierarchy can be simulated by randomized machines with two-sided error that run in time n^c and small space. Combining that hypothesis with (i) and (ii), we conclude that computations in the ℓth level of the polynomial-time hierarchy that run in time T (where T is some sufficiently large polynomial) can be complemented in time $g(T)$, where g is some function depending on c. For sufficiently small values of c, $g(T)$ becomes $o(T)$, which yields a contradiction with a known diagonalization result. For somewhat larger values of c, we do not obtain a contradiction right away but we obtain a more efficient complementation within the ℓth level of the polynomial-time hierarchy for larger polynomials T. We then run the argument again using the new efficient complementation in step (ii), yielding an even more efficient complementation. This allows us to rule out larger values of c and further improve the efficiency of complementation. Bootstrapping this way leads to Theorem 1.

A careful analysis shows that we can handle space bounds of the form n^d, where d is a positive constant depending on c. For $\ell = 2$, the above argument yields a constant d approaching $1/2$ from below when c approaches 1 from above. For $\ell \geq 3$, we achieve a better value of d for such small values of c by deriving a more efficient simulation of randomized computations in the *third* level of the polynomial-time hierarchy.[1] This follows by exploiting the structure of the second-level simulation described above and adding an alternation to reduce the time overhead. The savings in time are more substantial for large values of d. When c approaches 1 from above, the modified argument can handle values of d approaching 1 from below. For larger values of c, the cost of the additional alternation obviates the savings in running time and makes the earlier argument the better one. By paying close attention to the space used by the simulations involved, we obtain the strengthening given in Theorem 2.

For our tautology lower bounds, we extend Williams' recent lower bounds for deterministic machines [24] to nondeterministic machines with few guess bits and add a new component to the argument. The proof in [24] contains two ingredients. The first one is a bootstrapping argument similar to the one we just described. Instead of yielding more and more efficient complementations within the ℓth level of the polynomial-time hierarchy for some fixed ℓ at larger and larger polynomial time bounds T, it gives more and more efficient complementations of linear time within higher and higher levels of the polynomial-time hierarchy. The second ingredient is an improved starting point for the bootstrapping argument. This involves using the hypothesis to obtain a—conditional—better speedup of small-space deterministic computations in the second level of the polynomial-time hierarchy. To obtain the quantitative improvement stated in Corollary 4, we take the idea of exploiting the hypothesis of the indirect diagonalization argument further and show how to improve the conditional speedup of small-space deterministic computations in higher levels of the polynomial-time hierarchy. In order to establish the lower bounds on randomized machines with one-sided error (as in Theorem 3), we show that the above argument extends from deterministic machines to nondeterministic machines that use few guess

[1] Viola [23] independently obtained the same simulation using a somewhat more complicated argument.

bits. We exploit Nisan's partial derandomization again, this time to transform one-sided error randomized machines into equivalent nondeterministic machines with few guess bits at a marginal cost in time and space.

1.3. Organization. Section 2 introduces the notation and the machine models we use for this paper. Additionally, we state some useful complexity results which are fundamental to our techniques.

In section 3, we describe the general framework of our proofs. This includes a tight connection between QSAT_ℓ and linear time on an alternating machine with $\ell - 1$ alternations, which allows us to focus on proving lower bounds for the latter from there on. We also describe the paradigm of indirect diagonalization, which our lower bound proofs follow, and give a concrete example.

Section 4 shows how we can leverage Nisan's space-bounded derandomization, deterministic amplification, and Lautemann's proof that randomized machines with two-sided error can be simulated by alternating machines with one alternation at a polynomial time-overhead. Specifically, we obtain simulations of space-bounded randomized machines by alternating machines which use few guess bits and only marginally more time and space than the randomized machine. We exploit these simulations in section 5 to establish the lower bounds given by Theorem 1.

Section 6 contains our other results. Specifically, we show the more general time-space lower bounds for space-bounded linear-time alternating machines given by Theorem 2, as well as the time-space lower bounds for tautology on randomized machines with one-sided error given by Theorem 3.

Finally, we conclude in section 7 by discussing some open problems that remain directions for further research. We also include an appendix in which we prove some results regarding the running time of Nisan's generator and of deterministic amplification based on a random walk on the Gabber–Galil expander. We could not find these results in the literature and they may be of independent interest.

2. Preliminaries. While much of the notation we use is standard [2, 22], we introduce some conventions and additional notation in this section. We also state a few results which we use throughout the rest of the paper.

2.1. Machine model. Our lower bounds are robust with respect to the choice of machine model. In particular, they hold for random-access machines. We refer to [19] for the details of the specific model we use for our derivations.

We adopt the convention that the time and space bounds of these machines are constructible functions from natural numbers to natural numbers which are at least logarithmic, and refer to them as time and space functions. We often discuss "subpolynomial" space functions, which refer to space functions in $n^{o(1)}$. Our results ultimately apply to computations with polynomial time and space bounds, which certainly meet the required constructibility conditions. Note that machines running in sublinear time or sublogarithmic space trivially cannot solve problems like satisfiability, where the answer can depend on the entire input.

2.2. Notation. We introduce some additional terminology to represent randomized computation. In particular, we use the notation $\mathrm{BPTISP}[T, S]$ to refer to the class of languages recognized by randomized machines using time T and space S with error bounded by $\frac{1}{3}$ on both sides. Similarly, $\mathrm{RTISP}[T, S]$ refers to randomized machines with one-sided error on the membership side bounded by $\frac{1}{2}$.

As is standard, we assume that the random bits are presented to such machines on a one-way read-only worktape. If the machine wishes to reread random bits, it

must copy them down onto a worktape at the cost of space, as opposed to the more powerful model which has two-way access to the random tape. Except where stated otherwise, our results about randomized machines hold only for the former machine model.

Our arguments involve alternating computations in which the numbers of bits guessed at each stage are bounded by explicitly given small functions. To this end, we use the following notation to describe such computations.

DEFINITION 5. *Given a complexity class $\mathcal{C}$ and a function f, we define the class $\exists^f\mathcal{C}$ to be the set of languages that can be described as*

$$\{x|\exists y \in \{0,1\}^{O(f(|x|))} P(x,y)\},$$

where P is a predicate accepting a language in the class $\mathcal{C}$ when its complexity is measured in terms of $|x|$ (not $|x|+|y|$). We analogously define $\forall^f\mathcal{C}$.

For example, $\exists^f\text{DTIME}[n]$ and $\forall^f\text{DTIME}[n]$ are subsets of NP and coNP for $f(n) = n^{O(1)}$. The requirement that the complexity of P be measured in terms of $|x|$ allows us to express the running times in terms of the original input length, which is a more natural convention for our arguments.

A subtlety arises when we consider space-bounded classes $\mathcal{C}$. Computations corresponding to $\exists^f\mathcal{C}$ and $\forall^f\mathcal{C}$ explicitly write down their guess bits y and then run a space-bounded machine on the combined input consisting of the original input x and the guess bits y. Thus, the space-bounded machine effectively has two-way access to the guess bits y. For example, although machines corresponding to $\exists^n\text{DTISP}[n, n^{o(1)}]$ and to $\text{NTISP}[n, n^{o(1)}]$ both use only a subpolynomial amount of space to verify their guesses, they do not necessarily have the same computational power. This is because the former machines have two-way access to the guess bits, which are written down on a separate tape that does not count towards its space bound, whereas the latter machines have only one-way access to these bits and do not have enough space to write them down on their worktape.

2.3. Speedup of space-bounded computations. We also make use of the standard divide-and-conquer approach for speeding up space-bounded computations by introducing alternations. This technique is described in detail in [19]. By splitting up the computation tableau of a $\text{DTISP}[T,S]$ computation into $B \geq 1$ equal-sized blocks, we obtain

$$\text{DTISP}[T,S] \subseteq \exists^{BS}\forall^{\log B}\text{DTISP}[T/B,S] \subseteq \Sigma_2\text{TIME}[BS + T/B]. \tag{1}$$

By the closure of DTISP under complement, this inclusion can also be stated for the Π-side of the polynomial-time hierarchy, which will be more convenient to use in some of our arguments. If we choose B to optimize the running time of the resulting Π_2-computation, the result is a square-root speedup for small space bounds S:

$$\text{DTISP}[T,S] \subseteq \forall^{\sqrt{TS}}\exists^{\log T}\text{DTISP}[\sqrt{TS},S] \subseteq \Pi_2\text{TIME}[\sqrt{TS}]. \tag{2}$$

Recursively applying (1) while exploiting DTISP's closure under complementation to conserve alternations yields

$$\begin{aligned}\text{DTISP}[T,S] &\subseteq \underbrace{\forall^{BS}\exists^{BS}\cdots\overline{Q}^{BS}}_{k-1}\ Q^{\log B}\text{DTISP}[T/B^{k-1}+BS,S] \\ &\subseteq \Pi_k\text{TIME}[T/B^{k-1}+BS]\end{aligned} \tag{3}$$

for any integer $k \geq 2$, where $Q = \exists$ if k is even and $Q = \forall$ otherwise, and $\overline{Q}$ denotes the quantifier complementary to Q. Choosing B to optimize the resulting running time, (3) achieves a kth-root speedup for small space bounds S:

$$\mathrm{DTISP}[T,S] \subseteq \underbrace{\forall^{(TS^{k-1})^{1/k}}\exists^{(TS^{k-1})^{1/k}}\cdots\overline{Q}^{(TS^{k-1})^{1/k}}}_{k-1} Q^{\log(T/S)}\mathrm{DTISP}[(TS^{k-1})^{1/k},S]$$

$$\subseteq \Pi_k\mathrm{TIME}[(TS^{k-1})^{1/k}]. \tag{4}$$

2.4. Diagonalization results. Finally, we need a standard diagonalization result from which we can derive contradictions. The following lemma states that we cannot speed up the computation of every language in $\Sigma_\ell\mathrm{TIME}[T]$ by switching to Π_ℓ.

LEMMA 6 (folklore). *Let ℓ be a positive integer and T a time function. Then*

$$\Sigma_\ell\mathrm{TIME}[T] \not\subseteq \Pi_\ell\mathrm{TIME}[o(T)].$$

Our lower bounds for space-bounded alternating linear time require a stronger diagonalization result which is both time- and space-sensitive.

LEMMA 7 (see [9]). *Let T be a time function and S a space function. Then for any integer $\ell > 0$,*

$$\Sigma_\ell\mathrm{TISP}[T,S] \not\subseteq \Pi_\ell\mathrm{TISP}[o(T),o(S)].$$

3. Earlier techniques. We now outline some of the techniques common to many time-space lower bound arguments. We also use them for our results.

3.1. Alternating linear time versus QSAT_ℓ. The first such result involves a tight connection between QSAT_ℓ and linear time on an alternating Turing machine with ℓ alternating stages, $\Sigma_\ell\mathrm{TIME}[n]$. At the first level, we know that satisfiability can be solved in nondeterministic quasilinear time, so that time-space lower bounds for satisfiability imply the same lower bounds for $\mathrm{NTIME}[n]$ up to polylogarithmic factors. Conversely, Fortnow and van Melkebeek [9] show that a sufficient strengthening of the Cook–Levin theorem gives a reduction from $\mathrm{NTIME}[n]$ to satisfiability which is efficient in both time *and* space, showing that if satisfiability can be solved in time n^c and space n^d, then $\mathrm{NTIME}[n]$ can be solved in time $n^c\,\mathrm{polylog}(n)$ and space $n^d\,\mathrm{polylog}(n)$. Thus, time-space lower bounds for $\mathrm{NTIME}[n]$ and for satisfiability are equivalent up to polylogarithmic factors.

At higher levels of the polynomial-time hierarchy, we know that QSAT_ℓ can be solved in quasilinear time on a machine which makes $\ell - 1$ alternations, so that time-space lower bounds for QSAT_ℓ imply the same lower bounds for $\Sigma_\ell\mathrm{TIME}[n]$ up to polylogarithmic factors. Conversely, just as the Cook–Levin theorem generalizes from NP to higher levels of the polynomial-time hierarchy and shows that QSAT_ℓ is complete for Σ_ℓ^p, the reductions given by Fortnow and van Melkebeek [9] generalize to give time- and space-efficient reductions from $\Sigma_\ell\mathrm{TIME}[n]$ to QSAT_ℓ.

THEOREM 8. *For any integer $\ell \geq 1$ and constants $c, d > 0$, if*

$$\mathrm{QSAT}_\ell \in \mathrm{DTISP}[n^c, n^d],$$

then

$$\Sigma_\ell\mathrm{TIME}[n] \subseteq \mathrm{DTISP}[n^c\,\mathrm{polylog}(n), n^d\,\mathrm{polylog}(n)].$$

This also holds if we replace DTISP *with* BPTISP *or* RTISP.

This establishes the equivalence of time-space lower bounds for $\Sigma_\ell\text{TIME}[n]$ and QSAT_ℓ up to polylogarithmic factors. In particular, polynomial-strength time-space lower bounds for $\Sigma_\ell\text{TIME}[n]$ on randomized machines yield essentially the same lower bounds for QSAT_ℓ. With this in mind, we focus on proving lower bounds for $\Sigma_\ell\text{TIME}[n]$ for the rest of the paper. We include a proof of Theorem 8 here for completeness.

Proof. Let L be a language decided by a random-access linear-time alternating Turing machine which makes $\ell - 1$ alternations, beginning in an existential stage. For such an L, let P be the predicate recognized by a random-access linear-time nondeterministic machine so that the condition $x \in L$ can be written as

$$(\exists y_1 \in \{0,1\}^{rn})(\forall y_2 \in \{0,1\}^{rn}) \cdots (Q y_{\ell-1} \in \{0,1\}^{rn}) R(x, y_1, y_2, \ldots, y_{\ell-1}),$$

where r is some constant, and $Q = \forall$, $R = P$ if ℓ is odd, and $Q = \exists$, $R = \neg P$ otherwise. All that remains to represent the acceptance condition of L as a quantified Boolean formula is to reduce P to satisfiability. The original Cook–Levin reduction produces a formula of quadratic size, which is too large for our purposes. However, Cook [6] shows how to leverage the oblivious simulations of Hennie and Stearns [12] to obtain a formula of quasilinear size. More precisely, we can construct a formula φ depending only on P and n such that φ

(i) has size $O(n \operatorname{polylog}(n))$ where each bit can be constructed in time $O(\operatorname{polylog}(n))$ and space $O(\log(n))$,

(ii) involves the bits of $x, y_1, \ldots, y_{\ell-1}$ input to P as well as $O(n \log n)$ additional Boolean variables z, and

(iii) is satisfiable in z on input $x, y_1, \ldots, y_{\ell-1}$ if and only if P accepts $x, y_1, \ldots, y_{\ell-1}$.

Defining $\varphi' \doteq \varphi$ if ℓ is odd and $\varphi' \doteq \neg\varphi$ otherwise, this shows that the Σ_ℓ-formula,

$$\psi \doteq \exists y_1 \forall y_2 \cdots Q y_{\ell-1} \overline{Q} z \, \varphi',$$

is in QSAT_ℓ if and only if $x \in L$. The size of ψ is only $O(n \log n)$ more than φ—the log factor is required to write down the bit indices of the quantified variables. The easy nature of these extensions to φ endows ψ with the same constructibility properties as φ.

In the case that $\text{QSAT}_\ell \in \text{DTISP}[n^c, n^d]$, let M be a deterministic machine deciding QSAT_ℓ in time $O(n^c)$ and space $O(n^d)$. We simulate M on input ψ to decide L. However, computing ψ and writing down the result on a worktape requires too much space. Instead, when M needs a bit of ψ, the simulation computes this bit from scratch. As φ is of size $O(n \operatorname{polylog}(n))$, M runs for time $O(n^c \operatorname{polylog}(n))$ and space $O(n^d \operatorname{polylog}(n))$ on input ψ. Computing the bits of ψ on the fly adds a multiplicative overhead of $O(\operatorname{polylog}(n))$ to the time and an additive overhead of $O(\log n)$ to the space. Thus, this deterministic simulation decides L in the desired time and space bounds.

The cases for BPTISP and RTISP follow from the same argument. □

3.2. Indirect diagonalization. We set out to prove Theorem 1, i.e., that $\Sigma_\ell\text{TIME}[n] \not\subseteq \text{BPTISP}[t, s]$ for certain interesting values of t and s. To accomplish this, we follow the same general technique that is used to prove all previous time-space lower bounds for nondeterministic linear time, namely indirect diagonalization. This paradigm follows three basic steps:

1. Assume the inclusion that we wish to show does not hold. In our case, assume that $\Sigma_\ell\text{TIME}[n] \subseteq \text{BPTISP}[t, s]$.

2. Using the hypothesis, derive inclusions of complexity classes which are increasingly unlikely.

3. Eventually, one of these inclusions contradicts a known diagonalization result, proving the desired result.

There is a myriad of ways to derive new inclusions from the hypothesis in step 2, with different approaches yielding different results. Often, the inclusions derived in step 2 are obtained by a combination of two opposing processes. These can be loosely thought of as deriving a speedup at the cost of adding alternations, and removing alternations at the cost of a small slowdown. The main intuition that guides how we apply these processes is that we wish the speedup gained by introducing alternations to outweigh the cost of eliminating them, so that overall we obtain a contradiction to Lemma 6 (or Lemma 7).

The former process, deriving a speedup, involves simulating a space-bounded machine of the type for which we are attempting to derive a lower bound (i.e., deterministic or randomized) in significantly less time. We must pay a price to achieve this task, and we choose to pay in the currency of alternations. Specifically, we introduce a small number of alternations to the computation and carry out a fast simulation on an alternating machine. In the deterministic setting, such a simulation yields an inclusion resembling

$$\mathrm{DTISP}[T, S] \subseteq \Pi_\ell \mathrm{TIME}[f(T, S)],$$

for $\ell \geq 1$ and $f(T, S) \ll T$ when $S \ll T$. Note that the Π_2-simulation of a DTISP-computation given by (2) is an example of this process.

In the other direction, the process of removing alternations involves the task of simulating an alternating machine by another alternating machine which makes fewer alternations and runs for only slightly more time. In many cases, this is accomplished by deriving a statement such as

$$\Sigma_\ell \mathrm{TIME}[T] \subseteq \Pi_\ell \mathrm{TIME}[g(T)],$$

for a small function g. When g is polynomial, such an inclusion results in a collapse of the polynomial-time hierarchy to the ℓth level, and we refer to it as an *efficient complementation*. Note that a complementation can be derived unconditionally by using an exhaustive search in lieu of an alternating step, but in this case g will be exponential in T. In our arguments, we can derive an efficient complementation which is conditional on the hypothesis of step 1, from which more efficient complementations are derived in an inductive fashion in step 2.

Notice that by Lemma 6, a complementation with $g(T) = o(T)$ is not just unlikely but impossible. Deriving such an impossibly efficient complementation provides the desired inclusion to arrive at a contradiction in step 3. In the following section, we give an example of how to accomplish this end via an appropriate combination of a speedup and an efficient complementation.

3.3. A concrete example. We step through an instantiation of the indirect diagonalization paradigm and prove the result of [17] that satisfiability cannot be solved by deterministic random-access machines running in time n^c and space $n^{o(1)}$ for constants $c < \sqrt{2}$. The first step is to assume that

$$\mathrm{NTIME}[n] \subseteq \mathrm{DTISP}[n^c, n^{o(1)}]. \tag{5}$$

This allows a simulation of $\mathrm{NTIME}[T]$ by $\mathrm{DTISP}[T^c, T^{o(1)}]$ for some polynomial T. The speedup given by the inclusion (2) then yields a square-root speedup at the cost

of two alternations. The net result is

$$\mathrm{NTIME}[T] \subseteq \mathrm{DTISP}[T^c, T^{o(1)}] \subseteq \Pi_2\mathrm{TIME}[T^{c/2+o(1)}], \tag{6}$$

which represents a speedup of NTIME$[T]$ for $c < 2$ by using one more alternating stage. To contradict Lemma 6, we need to arrive at such a speedup which uses just a universal stage. Therefore, we use an efficient complementation to remove the inner existential stage from the Π_2-speedup represented by (6). Note that hypothesis (5) gives a simulation of NTIME$[n]$ by a deterministic machine, and since any deterministic machine is trivially a conondeterministic machine, we have

$$\mathrm{NTIME}[n] \subseteq \mathrm{coNTIME}[n^c]. \tag{7}$$

In order to use this efficient complementation to remove an alternation, we write

$$\Pi_2\mathrm{TIME}[T^{c/2+o(1)}] = \forall^{T^{c/2+o(1)}} \underbrace{\mathrm{NTIME}[T^{c/2+o(1)}]}_{(\alpha)}.$$

Note that (α) represents a nondeterministic computation which takes an input of size $n + T^{c/2+o(1)}$ and runs in time $T^{c/2+o(1)}$. For $T(n) \geq n^{2/c}$, the running time is at least linear in the input size, so that (7) gives a conondeterministic simulation of (α) running in time $T^{c^2/2+o(1)}$. By merging the resulting adjacent universal stages, we obtain

$$\mathrm{NTIME}[T] \subseteq \forall^{T^{c/2+o(1)}}\mathrm{coNTIME}[T^{c^2/2+o(1)}] = \mathrm{coNTIME}[T^{c^2/2+o(1)}].$$

When $c^2/2 < 1$, we can see this process has delivered a net speedup of nondeterministic time T on conondeterministic machines, which is a contradiction to Lemma 6. This proves the desired result.

4. Lautemann's proof and derandomization. We now adopt the techniques of the previous section to prove Theorem 1, beginning with the case $\ell = 2$. By way of Theorem 8 and the indirect diagonalization paradigm, we seek to derive a contradiction from the assumption

$$\Sigma_2\mathrm{TIME}[n] \subseteq \mathrm{BPTISP}[t, s] \tag{8}$$

for some interesting functions t and s. The known proofs that BPP (the class of languages recognized by polynomial-time randomized machines with two-sided error) lies in the second level of the polynomial-time hierarchy provide a simulation of randomized machines with two-sided error by Π_2-machines with a polynomial overhead in time. Combined with hypothesis (8), this immediately gives us one of the ingredients we need to carry through the program from the previous section, namely the efficient complementation of Σ_2TIME$[n]$.

Assuming the Π_2-simulation is sufficiently time and space efficient, we can also use it for the other ingredient we need, namely the speedup. This is because the divide-and-conquer strategy for DTISP-computations from section 2.3 applies to Σ_kTISP-computations as well. However, it turns out that the known Π_2-simulations of randomized two-sided error machines are not time- and space-efficient enough to obtain any lower bounds this way. Moreover, in order to achieve the quantitative strength of our lower bounds, we need to save alternations by applying the speedup as in (1) to the final deterministic phase of the simulation as opposed to the Π_2-simulations as a whole. For that approach to yield an overall speedup, we need the number of

guess bits in the alternating phases of the simulation to be small—otherwise, the time needed for the guesses would obviate the speedup obtained in the final deterministic phase. The known simulations use too many guess bits from that perspective. Therefore, in this section, we develop a new Π_2-simulation of randomized machines with two-sided error that meets all the above efficiency requirements.

We start by analyzing Lautemann's proof that any language L in BPP is also in $\Sigma_2^p \cap \Pi_2^p$. The proof assumes a randomized algorithm using R random bits to decide L with error ϵ. When ϵ is small enough in comparison to R, there is a $v \geq 1$ so that membership of x in L can be characterized by the existence of v shifts of the set of random strings accepting x which together cover the universe of all random strings. If $x \in L$, the set of random strings accepting x is large enough to guarantee that such shifts exist as long as $\epsilon^v < 2^{-R}$. On the other hand, if $x \notin L$, the set of accepting random strings is small enough so that v shifts cannot cover the universe of random strings as long as $\epsilon < \frac{1}{v}$. For such ϵ and v, these complementary conditions are expressed by a Σ_2^p predicate. Since BPP is closed under complement, this shows that BPP $\subseteq \Sigma_2^p \cap \Pi_2^p$. Specifically, we are interested in the Π_2^p-side of the inclusion.

THEOREM 9 (Lautemann [15]). *Let L be a language recognized by a randomized machine M that runs in time T and space S and that uses R random bits with error bounded on both sides by ϵ. Then for any $v \geq 1$ such that $\epsilon < \min(2^{-R/v}, \frac{1}{v})$, we have that*

$$L \in \forall^{vR}\exists^{R}\mathrm{DTISP}[vT, S + \log v]. \tag{9}$$

In Lautemann's proof that BPP $\subseteq \Sigma_2^p \cap \Pi_2^p$, one starts from an algorithm deciding $L \in$ BPP that has error less than the reciprocal of the number R' of random bits it uses. Such an error probability can be achieved from a standard BPP algorithm deciding L that uses R random bits by taking the majority vote of $O(\log R)$ independent trials, which results in $R' = O(R \log R)$. For $\epsilon < \frac{1}{R'}$, choosing $v = R'$ satisfies the conditions of Lemma 9. This shows that if L can be decided by a BPTISP$[T, S]$ machine using R random bits, then

$$L \in \forall^{(R \log R)^2}\exists^{R \log R}\mathrm{DTISP}[RT \log R, S]. \tag{10}$$

We can further reduce the number of guess bits in the alternating stages of the simulation by using more efficient methods of amplification. With such methods, the error can be made as small as 2^{-R} while using only $O(R)$ random bits. Specifically, the algorithm runs $O(R)$ trials, which are obtained from the labels of vertices on a random walk of length $O(R)$ in an easily constructible expander graph, and accepts if a majority of these trials accept. For our purposes, we choose the Gabber–Galil family of expanders [10], a construction based on the Margulis family [18] where the vertices are connected via simple affine transformations on the labels. The easy form of the edge relations ensures that the walk is efficiently computable in time $O(R^2)$ and space $O(R)$.

THEOREM 10 (see [5, 13]). *Let M be a randomized machine with constant error bounded away from $\frac{1}{2}$ that runs in time T and space S and that uses R random bits. Then M can be simulated by another randomized machine M' that runs in time $O(RT)$ and space $O(R + S)$, while using only $O(R)$ random bits to achieve error 2^{-R}.*

When an algorithm has been amplified as in Theorem 10, $v = O(1)$ shifts suffice for Theorem 9. This shows that if L can be decided by a BPTISP$[T, S]$ machine using R random bits, then

$$L \in \forall^{R}\exists^{R}\mathrm{DTISP}[RT, R + S]. \tag{11}$$

The efficiency of this simulation depends on the number of random bits R for all the criteria we mentioned: the number of bits guessed in the alternating stages, the multiplicative time overhead, and the additive space overhead for the final deterministic stage are all $O(R)$. Since R can be as large as T, we need an additional ingredient to do better. That ingredient exploits the fact that we are dealing with *space-bounded* BPP-computations. In that setting, we know of techniques to reduce the number of random bits without increasing the time or space by much, which in turn increases the efficiency of the Π_2-simulation in (11). The means by which we achieve the needed reduction in randomness is the space-bounded derandomization of Nisan [20]. We state a version here and leave the proof to Appendix A (where we also present a brief overview of Theorem 10).

THEOREM 11. *Any randomized machine M running in time T and space S with error ϵ can be simulated by another randomized machine that runs in time $O(T\,\mathrm{polylog}(T))$ and space $O(S\log T)$ and that uses only $O(S\log T)$ random bits. The error of the simulation is $\epsilon + 2^{-S}$, and is one-sided if M has one-sided error.*

Note that we do not apply Theorem 11 to deterministically simulate the randomized machine. Instead, we use it to reduce the randomness required by a BPTISP$[T,S]$ machine to $O(S\log T)$. If we subsequently efficiently amplify using Theorem 10, then the overhead of the alternating simulation given by Theorem 9 becomes acceptable for polynomial T and small S. More precisely, we have the following lemma.

THEOREM 12.

$$\mathrm{BPTISP}[T,S] \subseteq \forall^{S\log T}\exists^{S\log T}\mathrm{DTISP}[TS\,\mathrm{polylog}(T), S\log T]. \tag{12}$$

Proof. Let M be the randomized time T, space S machine for recognizing $L \in \mathrm{BPTISP}[T,S]$. By the derandomization of Theorem 11, we obtain a simulation using $R \doteq O(S\log T)$ random bits, time $O(T\,\mathrm{polylog}(T))$, and space $O(S\log T)$, with error $\frac{1}{3} + 2^{-S}$. Theorem 10 gives a machine deciding L with error 2^{-R} while using $O(R)$ random bits. The time increases to $O(TS\,\mathrm{polylog}(T))$, while the space is still $O(S\log T)$. Applying Theorem 9 for $v = O(1)$ yields the desired alternating simulation of M. □

We point out that using the instantiation of Theorem 9 given by (10) instead of (11) in the proof of Theorem 12 yields a simulation similar to (12). The main difference is that the initial universal phase takes time $O((S\log T)^2)$ rather than $O(S\log T)$. This version is still efficient enough to yield time-space lower bounds as in Theorem 1, but the dependence of the space parameter d on c becomes worse.

4.1. Speedup. Theorem 12 has the desired nice properties that allow us to derive a speedup for BPTISP. The simulation spends only time $O(S\log T)$ in its alternating phases, which is small when S is small. In this case, the running time is dominated by the final deterministic computation, so that a speedup of this final stage results in a speedup of the computation as a whole. Since the final deterministic computation of the simulation given by (12) is space bounded, we can achieve this by applying the speedup of (1) or (3). For example, applying (1) adds two alternations, and by merging adjacent existential stages we obtain a simulation given by

$$\begin{aligned}\mathrm{BPTISP}[T,S] &\subseteq \forall^{S\log T}\exists^{S\log T}\exists^{BS\log T}\forall^{\log B}\mathrm{DTISP}[TS\,\mathrm{polylog}(T)/B, S\log T]\\ &= \forall^{S\log T}\exists^{BS\log T}\forall^{\log B}\mathrm{DTISP}[TS\,\mathrm{polylog}(T)/B, S\log T].\end{aligned}$$

Choosing B to optimize the running time of this simulation up to a $\mathrm{polylog}(T)$ factor, we get

$$\mathrm{BPTISP}[T,S] \subseteq \forall^{S\log T}\exists^{\sqrt{T}S}\forall^{\log T}\mathrm{DTISP}[\sqrt{T}S\,\mathrm{polylog}(T), S\log T]. \tag{13}$$

For small S, we obtain a speedup for space-bounded randomized machines similar to that which (2) gives for deterministic machines. However, the simulation uses three alternations rather than two to realize the same speedup by a square root.

4.2. Complementation. We also use Theorem 12 to derive an efficient complementation under the hypothesis of the indirect diagonalization argument. Note that hypothesis (8) gives a simulation of Σ_2 by BPTISP, which can be simulated in turn by the Π_2-machine given by Theorem 12. Thus, we derive

$$\Sigma_2\text{TIME}[n] \subseteq \forall^{s\log t}\exists^{s\log t}\text{DTISP}[ts\,\text{polylog}(t), s\log t], \tag{14}$$

which gives the desired complementation for small enough t and s. For example, when t is polynomial, say $t = n^c$ and s is subpolynomial, (14) becomes

$$\Sigma_2\text{TIME}[n] \subseteq \forall^{n^{o(1)}}\exists^{n^{o(1)}}\text{DTISP}[n^{c+o(1)}, n^{o(1)}] \subseteq \Pi_2\text{TIME}[n^{c+o(1)}]. \tag{15}$$

Thus, we can see that when s is small, this complementation allows us to eliminate alternations at a cost little more than raising the running time to the power of c. In this manner, (15) can be used analogously to the complementation (7) in the deterministic case of section 3.3.

We note that the bottleneck causing our lower bounds for QSAT_ℓ to hold only for $\ell \geq 2$ arises right here. In the case $\ell = 1$, the hypothesis becomes $\text{NTIME}[n] \subseteq \text{BPTISP}[t, s]$; combining with Theorem 12 as above, we obtain $\text{NTIME}[n] \subseteq \Pi_2\text{TIME}[ts\,\text{polylog}(t)]$, which is trivial and does not represent an efficient complementation.

4.3. Higher levels of the hierarchy. The previous discussion in this section developed techniques towards proving lower bounds for $\Sigma_2\text{TIME}[n]$. These readily generalize to higher levels, where hypothesis (8) becomes

$$\Sigma_\ell\text{TIME}[n] \subseteq \text{BPTISP}[t, s] \tag{16}$$

for $\ell \geq 3$. Theorem 12 then allows for an efficient simulation of $\Sigma_\ell\text{TIME}[n]$ by a Π_2-machine, which can be used to eliminate alternations. This allows us to establish Theorem 1 for values of $c < \ell$, where d approaches some small value depending on ℓ from below as c approaches 1 from above.

For $\ell \geq 3$, we can get a better dependence of d on c when c approaches 1. In this setting, a Π_3-simulation of BPTISP suffices to achieve an efficient complementation. The ability to use an additional alternation allows us to achieve a more time-efficient simulation than the one given by Theorem 12. Specifically, we add an alternation to the latter Π_2-simulation and eliminate the time blowup incurred by running the $O(S\log T)$ trials required by the amplification of Theorem 10. Rather than deterministically simulate all of these trials, we use the power of alternation to efficiently verify that a majority of these trials accept.

LEMMA 13. *Let M be a randomized machine with constant error bounded away from $\frac{1}{2}$ that runs in time T and space S and that uses R random bits. Then M can be simulated by another randomized machine M' that uses $O(R)$ random bits to achieve error 2^{-R}. Furthermore, the acceptance of M' on input x and random string r can be decided in*

$$\exists^R\forall^{\log R}\text{DTISP}[T + R\,\text{polylog}(R), R + S].$$

We defer the proof to Appendix B. Notice that the final deterministic stage of the simulation represented by (12) can be replaced with the Σ_2-verification given by

Lemma 13. Merging the resulting adjacent existential phases results in a simulation using one more alternation but running in less time.

THEOREM 14.

$$\text{BPTISP}[T, S] \subseteq \forall^{S \log T} \exists^{S \log T} \forall^{\log S} \text{DTISP}[T \operatorname{polylog}(T), S \log T]. \tag{17}$$

As in sections 4.1 and 4.2, Theorem 14 admits a speedup of BPTISP to Π_4 as well as an efficient complementation of Σ_3. When $t = n^c$ and $s = n^d$ in (16), the latter eliminates alternations essentially at the cost of raising the running time to the power of c. For values of c close to 1, this cost is small enough to alleviate the effects of the extra alternation in (17). In this case, the better dependence of the running time of the simulation on the space parameter allows us to derive contradictions for larger values of d than we can by using Theorem 12. On the other hand, for larger values of c, the extra alternation in (17) has a greater impact and eventually prevents us from reaching a contradiction. In this case, switching to the more alternation-efficient simulation given by Theorem 12 allows us to derive a contradiction for such larger values of c. However, we must restrict d to smaller values in order to counteract the worse dependence of (12) on the space bound. Therefore, to derive Theorem 1, we focus on using Theorem 12 to obtain the bounds for large values of c first and then show how Theorem 14 yields the larger values of d when c is small.

5. Main result. We now use the techniques discussed in the previous sections to formulate an indirect diagonalization argument for the case $\ell = 2$ of Theorem 1. For clarity, we present the following exposition in terms of subpolynomial space bounds and generalize these techniques to polynomial space bounds in the subsequent formal proof. Thus, to obtain a lower bound for $\Sigma_2\text{TIME}[n]$, we start with the assumption (8) for $t = n^c$ and $s = n^{o(1)}$, i.e.,

$$\Sigma_2\text{TIME}[n] \subseteq \text{BPTISP}[n^c, n^{o(1)}].$$

Consider a $\Sigma_2\text{TIME}[T]$ computation for some time function $T(n) = n^{O(1)}$. We adopt the approach outlined in section 3.3, namely speeding up $\Sigma_2\text{TIME}[T]$ at the cost of adding alternations and then removing these alternations via an efficient complementation to arrive at a $\Pi_2\text{TIME}[o(T)]$ computation. The hypothesis gives a simulation of $\Sigma_2\text{TIME}[T]$ in $\text{BPTISP}[T^c, T^{o(1)}]$. We then apply the square-root speedup of (13) to obtain a simulation in Π_3:

$$\begin{aligned}\Sigma_2\text{TIME}[T] &\subseteq \text{BPTISP}[T^c, T^{o(1)}] \\ &\subseteq \forall^{T^{o(1)}} \underbrace{\exists^{T^{\frac{c}{2}+o(1)}} \forall^{\log T} \text{DTISP}[T^{c/2+o(1)}, T^{o(1)}]}_{(\alpha)}.\end{aligned}$$

We have arrived at a simulation which makes one more alternation than we started with. To balance the number of alternations, we eliminate one alternation. Notice that the stages of the computation indicated by (α) can be seen as a computation in Σ_2 taking input of size $n + T^{o(1)}$ and running in time $T^{\frac{c}{2}+o(1)}$. When T is large enough, this running time is at least linear in the input size, and we can pad (15) to allow us to switch (α) to Π_2. Merging the resulting adjacent universal stages yields the desired Π_2-simulation:

$$\Sigma_2\text{TIME}[T] \subseteq \Pi_2\text{TIME}[T^{\frac{c^2}{2}+o(1)}]. \tag{18}$$

For $c < \sqrt{2}$, this results in a net speedup which is a contradiction to Lemma 6. Thus, we have derived a lower bound of $n^{\sqrt{2}-o(1)}$ for QSAT_2 on subpolynomial-space randomized machines. However, we can do better by observing what (18) represents for values of c that do not immediately contradict Lemma 6. Specifically, (18) gives a complementation of Σ_2 of the same form as (15), but has an exponent cost of $c^2/2$ rather than c. Thus, we have derived a *more efficient* complementation for sufficiently large polynomial time bounds T when $c < 2$.

We now reiterate the above argument, except we use (18) to eliminate alternations more efficiently than we did with (15). This yields an even more efficient complementation for sufficiently large polynomials T:

$$\Sigma_2\mathrm{TIME}[T] \subseteq \Pi_2\mathrm{TIME}[T^{\frac{c^3}{4}+o(1)}].$$

This can in turn be used to derive another more efficient complementation, and so on. In this manner, we can derive a series of complementations, each one more efficient than the previous one. Specifically, each iteration multiplies the exponent cost of the complementation by $\frac{c}{2}$, so after k iterations we obtain

$$\Sigma_2\mathrm{TIME}[T] \subseteq \Pi_2\mathrm{TIME}[T^{c\cdot e_k+o(1)}],$$

where $e_k = (\frac{c}{2})^k$. Note that for $c < 2$, $e_k \to 0$ as $k \to \infty$. Thus, by choosing k large enough so that $c \cdot e_k < 1$, we arrive at a contradiction to Lemma 6, which proves the desired lower bound of $n^{2-o(1)}$.

The following lemma precisely derives the series of complementations of Σ_2 for larger space bounds than $n^{o(1)}$. Specifically, we consider the hypothesis (8) for polynomials t and s, namely $t = n^c$ and $s = n^d$ for some constants $c \geq 1$ and $d > 0$, and derive the running time of the resulting Π_2-simulation in terms of c, d, and k, the number of times the argument is recursively applied.

LEMMA 15. *Suppose that*

$$\Sigma_2\mathrm{TIME}[n] \subseteq \mathrm{BPTISP}[n^c, n^d] \tag{19}$$

for some constants $c \geq 1$ and $d > 0$ where $c + 2d \leq 2$. Then for any time function T and integer $k \geq 0$ such that $d \leq f_k$,

$$\Sigma_2\mathrm{TIME}[T] \subseteq \Pi_2\mathrm{TIME}\left[\left((T^{f_k}+n)^{c+d}\right)\mathrm{polylog}(T+n)\right],$$

where

$$f_k = \left(\frac{c+2d}{2}\right)^k. \tag{20}$$

Proof. We give a proof by induction on k. For $k = 0$, a padded version of the initial complementation given by (14) offers a Π_2-simulation of $\Sigma_2\mathrm{TIME}[T]$ running in the desired time.

We now show the inductive step by using the kth complementation to derive the $(k+1)$st. Padding hypothesis (19) gives a simulation of $\Sigma_2\mathrm{TIME}[T]$ in $\mathrm{BPTISP}[(T+n)^c, (T+n)^d]$. Note that the addition of the term n to T ensures the validity of this step for arbitrary T, in particular for sublinear T. Applying (13) gives a speedup of the BPTISP simulation at the cost of three alternations, yielding

$$\Sigma_2\mathrm{TIME}[T] \subseteq \forall^{(T+n)^d\log(T+n)}\underbrace{\Sigma_2\mathrm{TIME}\left[(T+n)^{\frac{c+2d}{2}}\,\mathrm{polylog}(T+n)\right]}_{(\alpha)}. \tag{21}$$

The complementation given by the inductive hypothesis switches (α) to a Π_2-computation, eliminating one alternation. Specifically, (α) represents a Σ_2-machine running in time

$$\tilde{T} \doteq O\left((T+n)^{\frac{c+2d}{2}} \operatorname{polylog}(T+n)\right)$$

on inputs comprised of the original n-bit input, in addition to the $(T+n)^d \log(T+n)$ bits guessed in the preceding universal stage, for a total input size of

$$\tilde{n} \doteq O\left(n + (T+n)^d \log(T+n)\right).$$

The inductive hypothesis allows us to simulate (α) by a Π_2-machine running in time $O\big((\tilde{T}^{f_k} + \tilde{n})^{c+d} \operatorname{polylog}(\tilde{T}+\tilde{n})\big)$. Using this Π_2-machine for (α) and accounting for the time spent in the initial universal stage of the simulation given by (21) results in a Π_2-simulation of $\Sigma_2\mathrm{TIME}[T]$ running in time big-O of

$$(T+n)^d \log(T+n) + \left((\tilde{T}^{f_k} + \tilde{n})^{c+d}\right) \operatorname{polylog}(\tilde{T}+\tilde{n}).$$

All that remains is to show that the above running time is of the desired form under the conditions on c and d. To simplify this expression, note that $\operatorname{polylog}(\tilde{T}+\tilde{n}) = \operatorname{polylog}(T+n)$. Collecting all of the other polylog terms, the running time can be written as big-O of

$$\left[(T+n)^d + \left(\left((T+n)^{\frac{c+2d}{2}}\right)^{f_k} + n + (T+n)^d\right)^{c+d}\right] \operatorname{polylog}(T+n).$$

The terms depending on T in the big-O expression have exponents d, $\frac{c+2d}{2} f_k(c+d)$, and $d(c+d)$ (putting aside the $\operatorname{polylog}(T+n)$ factor for a moment). Thus, when $d \le \frac{c+2d}{2} f_k = f_{k+1}$ and $c+d \ge 1$, the dominating term is $T^{\frac{c+2d}{2} f_k(c+d)} = T^{f_{k+1}(c+d)}$.

Similarly, the terms depending on n have exponents d, $\frac{c+2d}{2} f_k(c+d), c+d$, and $d(c+d)$. Under the same conditions on c and d, the first and last terms are subsumed by the second, which can be rewritten as $f_{k+1}(c+d)$. Additionally, when $c+2d \le 2$, $f_k \le 1$ for all $k \ge 0$, so that the n^{c+d} term dominates. Thus, we simplify the running time to big-O of

$$\left((T^{f_{k+1}} + n)^{c+d}\right) \operatorname{polylog}(T+n),$$

which is of the desired form. ∎

The series of complementations given by Lemma 15 lead to a contradiction with Lemma 6 for certain values of c and d, which proves the desired lower bound.

THEOREM 16. *For any constant $c < 2$, there exists a positive constant d such that $\Sigma_2\mathrm{TIME}[n]$ cannot be simulated by randomized random-access machines with two-sided error running in time n^c and space n^d. Moreover, d approaches $1/2$ from below as c approaches 1 from above.*

Proof. For $c < 1$, the theorem holds for any d by standard techniques. Namely, take the parity function, which is surely in $\Sigma_2\mathrm{TIME}[n]$, and consider any $\mathrm{BPTIME}[n^c]$ machine M which purportedly computes it. On input 0^n, we must have that

$$\sum_{i=1}^{n} \Pr[M \text{ looks at the } i\text{th bit on input } 0^n] \le n^c.$$

For $c < 1$, this implies that there is a bit position that M looks at very rarely. Namely, there exists an i such that

$$\Pr[M \text{ looks at the } i\text{th bit on input } 0^n] = o(1).$$

From this, we can deduce that M has approximately the same probability of accepting 0^n as it does 0^n with the ith bit flipped. Therefore, M cannot compute parity.

We prove the case for $c \geq 1$ via indirect diagonalization. Suppose, by way of contradiction, that

$$\Sigma_2\text{TIME}[n] \subseteq \text{BPTISP}[n^c, n^d] \tag{22}$$

for some constant $d > 0$ to be determined later. Then for any time function $\tau(n)$, Lemma 15 gives us the complementations

$$\Sigma_2\text{TIME}[\tau] \subseteq \Pi_2\text{TIME}\left[\left((\tau^{f_k} + n)^{c+d}\right) \text{polylog}(\tau + n)\right],$$

when $c + 2d \leq 2$ and $d \leq f_k$. Choosing τ so that $\tau(n)^{f_k} \geq n$ allows us to simplify this to

$$\Sigma_2\text{TIME}[\tau] \subseteq \Pi_2\text{TIME}[\tau^{(c+d)f_k} \text{polylog}(\tau)]. \tag{23}$$

The inclusion (23) gives a contradiction with Lemma 6 for any k with $f_k < \frac{1}{c+d}$. Note that $f_k \to 0$ as $k \to \infty$ if $c + 2d < 2$. Therefore, all that remains is to show that the latter condition is compatible with the others, i.e., that we can pick a constant $d > 0$ and an integer $k > 0$ such that

$$c + 2d < 2, \tag{24}$$

$$d \leq f_k, \tag{25}$$

and

$$f_k < \frac{1}{c+d}. \tag{26}$$

For any c and d satisfying (24), consider choosing $k \geq 1$ to be the smallest integer such that (26) is satisfied. Observe that $f_k \geq \frac{c+d}{2} f_{k-1}$, and by how we chose k, $f_{k-1} \geq \frac{1}{c+d}$. This shows that $f_k \geq 1/2$, so (25) is satisfied when $d \leq 1/2$. From (24), we have $d < \frac{2-c}{2}$, which is at most $1/2$ when $c \geq 1$. Therefore, choosing d such that $d < \frac{2-c}{2}$ and then calculating k as described above yields a d and k satisfying all of the constraints, leading to the desired contradiction. As c approaches 1 from above, $\frac{2-c}{2}$ approaches $1/2$ from below, so the largest value of d that yields a contradiction approaches $1/2$ as well. This proves that $\Sigma_2\text{TIME}[n] \not\subseteq \text{BPTISP}[n^c, n^d]$ for such c and d. □

We point out that, although we can handle the same values of c as in the deterministic setting, the dependence of d on c in Theorem 16 is worse. In particular, the proofs of the time-space lower bounds for deterministic machines show that d approaches 1 from below as c approaches 1 from above [9], while in our result d approaches $1/2$ from below as c approaches 1 from above.

The proof of Theorem 16 generalizes to $\Sigma_\ell\text{TIME}[n]$ for any $\ell \geq 3$. In this setting, the hypothesis becomes

$$\Sigma_\ell\text{TIME}[n] \subseteq \text{BPTISP}[n^c, n^d]. \tag{27}$$

Along with the Π_2-simulation of BPTISP$[T, S]$ of Theorem 12, this yields a collapse of the form $\Sigma_\ell \subseteq \Pi_2$, which allows us to eliminate more than one alternation at the same cost of removing one alternation in the setting of Theorem 16, where $\ell = 2$. Therefore, we can afford to use more alternations using (4) and achieve a greater speedup. In a manner analogous to the proof of Theorem 16, we derive a series of increasingly efficient complementations of Σ_ℓ to Π_ℓ for $c < \ell$, eventually reaching a contradiction as long as $d \leq \frac{1}{\sqrt{\ell}}$.

Alternatively, we can use extra alternations to achieve a dependence of d on c, where d approaches 1 from below as c approaches 1 from above, as in the deterministic case. For example, when $\ell = 3$, this follows by deriving a complementation of Σ_3 following the same technique that derives the complementation of Σ_2 given by (18) when $\ell = 2$, modulo the replacement of the Π_2-simulation given by Theorem 12 with the Π_3-simulation given by Theorem 14. Specifically, hypothesis (27) and Theorem 14 give a complementation of Σ_3:

$$\Sigma_3\text{TIME}[n] \subseteq \forall^{n^d \log n}\exists^{n^d \log n}\forall^{\log n}\text{DTISP}[n^c \operatorname{polylog}(n), n^d \log n]. \tag{28}$$

Consider a Σ_3TIME$[\tau]$ computation for some time function τ to be determined. Applying the speedup of (2) to the final deterministic stage of the simulation given by (28) yields

$$\Sigma_3\text{TIME}[\tau] \subseteq \forall^{\tau^d \log \tau} \underbrace{\exists^{\tau^d \log \tau}\Pi_2\text{TIME}[\tau^{\frac{c+d}{2}} \operatorname{polylog}(\tau)]}_{(\alpha)}.$$

We can now use (28) to simulate (α) by a Π_3-machine, yielding a more efficient complementation than (28) for certain values of c and d:

$$\Sigma_3\text{TIME}[\tau] \subseteq \forall^{\tau^d \log \tau}\Pi_3\text{TIME}[(\tau^{\frac{c+d}{2}} + n + \tau^d)^c \operatorname{polylog}(\tau)].$$

When $\tau(n) \geq n^{\frac{2}{c+d}}$ (and $d \leq c$), this simplifies to

$$\Sigma_3\text{TIME}[\tau] \subseteq \Pi_3\text{TIME}[\tau^{c\frac{c+d}{2}} \operatorname{polylog}(\tau)], \tag{29}$$

yielding a contradiction to Lemma 6 when $c < \sqrt{2}$ and $d < \frac{2-c^2}{c}$. Therefore, as c approaches 1 from above, the upper bound on d approaches 1 from below when $\ell = 3$. Indeed, this analysis establishes the desired behavior of d as c approaches 1 for any level $\ell \geq 3$, since if (27) holds for $\ell > 3$, it must also hold for $\ell = 3$.

We point out that we can augment the strategy leading to (28) with a bootstrapping argument similar to the one in the proof of Lemma 15 and obtain increasingly efficient complementations of Σ_3. This approach results in a contradiction for $c < 2$, whereas the analogous approach using Theorem 12 leads to a contradiction for $c < 3$. More generally, at level $\ell \geq 3$, we can modify the above approach to take full advantage of the stronger hypothesis and arrive at complementations of Σ_ℓ. However, we reach a contradiction only for $c < \ell - 1$, whereas the strategy based on Theorem 12 results in a contradiction for $c < \ell$. Therefore, we need *both* approaches to prove Theorem 1—the latter achieves the lower bound for large values of c, while the former establishes the dependence of d on c for small values of c.

Since much of the proof of Theorem 1 closely follows the outline of Theorem 16, we give only a brief sketch of it here. In fact, Theorem 1 also follows from the more

general Theorem 2, which gives time-space lower bounds for simulations of space-bounded linear-time alternating machines. A complete proof of Theorem 2 appears in the next section.

Proof of Theorem 1. The case for $c < 1$ follows by standard techniques, as in the proof of Theorem 16.

For the case $c \geq 1$, assume that $\Sigma_\ell \text{TIME}[n] \subseteq \text{BPTISP}[n^c, n^d]$ for some constant $d > 0$ to be determined later. Using the speedup given by (4) rather than (2), we can step through an argument similar to that of Lemma 15 to show

$$\Sigma_\ell \text{TIME}[T] \subseteq \Pi_\ell \text{TIME}\left[\left((T^{g_k} + n)^{c+d}\right) \text{polylog}(T + n)\right], \tag{30}$$

as long as $c + \ell d \leq \ell$ and $d \leq g_k$, where

$$g_k = \left(\frac{c + \ell d}{\ell}\right)^k. \tag{31}$$

We can now use (30) as we used Lemma 15 in the proof of Theorem 16. For a time function τ such that $\tau(n)^{g_k} \geq n$, (30) gives that

$$\Sigma_\ell \text{TIME}[\tau] \subseteq \Pi_\ell \text{TIME}\left[\tau^{(c+d)g_k}\right],$$

which is a contradiction with Lemma 6 when $g_k < \frac{1}{c+d}$. Therefore, it remains to show that it is possible to choose a positive d and integer k satisfying the latter condition as well as those on (30). More specifically, for $c < \ell$ we can choose $d < \frac{\ell - c}{\ell}$ so that there exists a smallest positive integer k such that $g_k < \frac{1}{c+d}$. Since $g_k = \frac{c+\ell d}{\ell} \cdot g_{k-1} \geq \frac{c+\ell d}{\ell} \cdot \frac{1}{c+d}$, we can guarantee that the only constraint possibly left unsatisfied, namely $d \leq g_k$, is met by restricting our choice of d to $d \leq \frac{c+\ell d}{\ell} \cdot \frac{1}{c+d}$.

When $\ell = 2$, the upper bound on d approaches $\frac{1}{2}$ as c approaches 1. For $\ell \geq 3$, the upper bound on d approaches $\frac{1}{\sqrt{\ell}}$ as c approaches 1, but we can improve it using the combination of the hypothesis and Theorem 14 that leads to the complementation of Σ_3 represented by (29). For large enough τ, this gives a contradiction for values of d approaching 1 from below as c approaches 1 from above when $\ell \geq 3$.

Theorem 8 transfers the lower bounds to QSAT_ℓ. □

6. Other results. In this section, we strengthen Theorem 1 to establish time-space lower bounds for problems decidable by alternating machines that run in linear time and use only a small amount of space. We also prove Theorem 3 regarding time-space lower bounds for tautology on randomized machines with *one-sided* error.

6.1. Sublinear space on linear-time alternating machines. By paying close attention to the space used by the simulations in the proof of Theorem 1, we actually obtain time-space lower bounds for randomized simulations of linear-time alternating machines using space n^a for $a < 1$, given by Theorem 2. The main task is to use the weaker assumption that $\Sigma_\ell \text{TISP}[n, n^a] \subseteq \text{BPTISP}[n^c, n^d]$ to eliminate the alternations introduced by the speedup of (3). This requires that the Σ_k-simulation after the speedup use an amount of space which is at most the ath power of its running time. Since the simulation guesses (and stores) $O(BS)$ bits in each alternating stage, this restricts us to choose a small value of B, which in turn grants a smaller speedup. Therefore, our bounds become weaker as a becomes smaller.

To prove Theorem 2, we first prove an analogue of Lemma 15 which gives a series of complementations of $\Sigma_\ell \text{TISP}$.

LEMMA 17. *Suppose that*

$$\Sigma_\ell \mathrm{TISP}[n, n^a] \subseteq \mathrm{BPTISP}[n^c, n^d] \tag{32}$$

for some integer $\ell \geq 2$ *and constants* $0 < a \leq 1$, $c \geq 1$, *and* $d > 0$ *with* $c + \ell d \leq 1 + (\ell - 1)a$ *and* $(1 - a)d \leq ac$. *Then for any time function* T *and integer* $k \geq 0$ *such that* $d \leq h_k$,

$$\Sigma_\ell \mathrm{TISP}[T, T^a] \subseteq \Pi_\ell \mathrm{TISP}\left[\left((T^{h_k} + n)^{c+d}\right) \mathrm{polylog}(T + n), (T + n)^d \,\mathrm{polylog}(T + n)\right], \tag{33}$$

where

$$h_k = \left(\frac{c + \ell d}{1 + (\ell - 1)a}\right)^k. \tag{34}$$

Proof. The base case $k = 0$ is given by combining hypothesis (32) and the efficient Π_2-simulation of BPTISP given by Theorem 12. We now prove the inductive step $k \to k + 1$. Consider a $\Sigma_\ell \mathrm{TISP}[T, T^a]$ computation. From hypothesis (32), Theorem 12, and the speedup of (3) (to Σ_ℓ rather than Π_ℓ), we obtain a simulation which (neglecting the $\mathrm{polylog}(T + n)$ factors) is in

$$\forall^{(T+n)^d} \underbrace{\Sigma_\ell \mathrm{TISP}\left[B(T + n)^d + \frac{(T + n)^{c+d}}{B^{\ell-1}}, B(T + n)^d\right]}_{(\alpha)}.$$

In order to apply the inductive hypothesis to complete the complementation to $\Pi_\ell \mathrm{TISP}$, the space bound of (α) must be at most the ath power of its running time. This is the case when B satisfies

$$B(T + n)^d = \left(\frac{(T + n)^{c+d}}{B^{\ell-1}}\right)^a,$$

which offers the choice

$$B = (T + n)^{\frac{ac+(a-1)d}{1+(\ell-1)a}}$$

as long as $ac + (a - 1)d \geq 0$. For such a choice, (α) is a computation in

$$\Sigma_\ell \mathrm{TISP}\left[(T + n)^{\frac{c+\ell d}{1+(\ell-1)a}}, (T + n)^{a\frac{c+\ell d}{1+(\ell-1)a}}\right],$$

which takes an input of size $O(n + (T + n)^d)$, which is in $O(n + T^d)$ when $d \leq 1$. Thus, instead of achieving an ℓth root speedup, as we do when we are not concerned about the space used by the simulation of (3), we instead achieve a $(1 + (\ell - 1)a)$th root speedup.

The inductive hypothesis gives a $\Pi_\ell \mathrm{TISP}$-simulation of (α), and hence a $\Pi_\ell \mathrm{TISP}$-simulation of $\Sigma_\ell \mathrm{TISP}[T, T^a]$ running in time big-O of (neglecting the $\mathrm{polylog}(T + n)$ terms)

$$T^* \doteq (T + n)^d + \left((T + n)^{h_k \frac{c+\ell d}{1+(\ell-1)a}} + n + T^d\right)^{c+d}$$

and using space big-O of

$$S^* \doteq (T + n)^d + \left((T + n)^{\frac{c+\ell d}{1+(\ell-1)a}} + n + T^d\right)^d.$$

When $c + \ell d \leq 1 + (\ell - 1)a$ and $d \leq 1$, S^* simplifies to $(T + n)^d$. When we have the further constraint that $d \leq h_{k+1}$, T^* has the appropriate leading terms and simplifies to $(T^{h_{k+1}} + n)^{c+d}$. Accounting for the polylog factors, we have shown that the simulation is of the desired form. □

We note that the proof of Lemma 17 also yields a version in which the Π_ℓ on the right-hand side of (33) is replaced with Π_2. However, we do not see a way to exploit that fact to strengthen our final result. For the sake of clarity and consistency, we present Lemma 17 as stated.

When $c + \ell d < 1 + (\ell - 1)a$, $h_k \to 0$ as $k \to \infty$. In such a case, we can use Lemma 17 to derive a contradiction to Lemma 7 in a fashion analogous to how we used Lemma 15 to prove Theorem 16 for values of c as large as possible. For $\ell = 2$, this gives a value of d approaching $\frac{a}{2}$ from below as c approaches 1 from above. We can do better when $\ell \geq 3$ by using Theorem 14 to derive a space-bounded complementation analogous to that given by (29) in the unrestricted case.

LEMMA 18. *Suppose that*

$$\Sigma_3\text{TISP}[n, n^a] \subseteq \text{BPTISP}[n^c, n^d] \tag{35}$$

for constants $0 < a \leq 1$, $c \geq 1$, *and* $0 < d \leq ac$ *with* $c + d \leq 1 + a$. *Then for any time function* T,

$$\Sigma_3\text{TISP}[T, T^a] \subseteq \Pi_3\text{TISP}[(T^{\frac{c+d}{1+a}} + n)^c \operatorname{polylog}(T + n), (T + n)^d \operatorname{polylog}(T + n)].$$

Proof. Hypothesis (35) and Theorem 14 yield the complementation

$$\Sigma_3\text{TISP}[n, n^a] \subseteq \forall^{n^d \log n} \exists^{n^d \log^2 n} \forall^{\log n} \text{DTISP}[n^c \operatorname{polylog}(n), n^d \log n]. \tag{36}$$

Consider a $\Sigma_3\text{TISP}[T, T^a]$ computation. Padding (36) and applying the speedup of (1) (on the Π_2-side) to the final deterministic stage gives a simulation of $\Sigma_3\text{TISP}[T, T^a]$ which (neglecting the polylog terms) is in

$$\forall^{(T+n)^d} \exists^{(T+n)^d} \underbrace{\Pi_2\text{TISP}[B(T + n)^d + (T + n)^c/B, B(T + n)^d]}_{(\beta)}.$$

Choosing $B = (T + n)^{\frac{ac-d}{1+a}}$ so that the space bound of (β) is the ath power of its running time places the simulation in

$$\forall^{(T+n)^d} \underbrace{\exists^{(T+n)^d} \Pi_2\text{TISP}[(T + n)^{\frac{c+d}{1+a}}, (T + n)^{a\frac{c+d}{1+a}}]}_{(\gamma)},$$

when $d \leq ac$. Under the same condition, (γ) is a computation in $\Sigma_3\text{TISP}[(T + n)^{\frac{c+d}{1+a}}, (T+n)^{a\frac{c+d}{1+a}}]$ taking an input of size $n+(T+n)^d$, so that (36) gives a simulation of (γ) in $\Pi_3\text{TISP}$. Overall we have derived

$$\begin{aligned}
\Sigma_3\text{TISP}[T, T^a] &\subseteq \Pi_3\text{TISP} \quad \left[\left((T + n)^{\frac{c+d}{1+a}} + n + (T + n)^d\right)^c, \right. \\
&\qquad \left. (T + n)^d + \left((T + n)^{\frac{c+d}{1+a}} + n + (T + n)^d\right)^d \right] \\
&\subseteq \Pi_3\text{TISP} \quad [(T^{\frac{c+d}{1+a}} + n)^c, (T + n)^d],
\end{aligned}$$

where the last inclusion follows as long as $d \leq ac$ and $\frac{c+d}{1+a} \leq 1$. Accounting for the polylog$(T + n)$ terms finishes the proof. □

We are now ready to prove Theorem 2.

Proof of Theorem 2. The proof for $c < 1$ follows from standard techniques, as in the proof of Theorem 16.

For the case $c \geq 1$, assume by way of contradiction that $\Sigma_\ell\text{TISP}[n, n^a] \subseteq \text{BPTISP}[n^c, n^d]$ for some constant $d > 0$ to be determined later. Then for a time function τ, where $\tau(n)^{h_k} \geq n$, Lemma 17 gives the complementation

$$\Sigma_\ell\text{TISP}[\tau, \tau^a] \subseteq \Pi_\ell\text{TISP}\left[\tau^{(c+d)h_k}\,\text{polylog}(\tau), \tau^d\,\text{polylog}(\tau)\right] \tag{37}$$

when $c + \ell d \leq 1 + (\ell - 1)a$, $(1 - a)d \leq ac$, and $d \leq h_k$. When $h_k < \frac{1}{c+d}$, the time bound of the right-hand side of (37) is $o(\tau)$. Provided that $c < 1 + (\ell - 1)a$, we can choose a positive d and an integer k such that all the above conditions are met. More specifically, for any value of $d < \frac{1+(\ell-1)a-c}{\ell}$, there exists a smallest positive integer k such that $h_k < \frac{1}{c+d}$. Since $h_k = \frac{c+\ell d}{1+(\ell-1)a} \cdot h_{k-1} \geq \frac{c+\ell d}{1+(\ell-1)a} \cdot \frac{1}{c+d}$, we can guarantee that $d \leq h_k$ by imposing the condition

$$d \leq \frac{c + \ell d}{1 + (\ell - 1)a} \cdot \frac{1}{c + d}. \tag{38}$$

It follows that we can meet all constraints mentioned so far by choosing d below some positive threshold. All that remains to reach a contradiction with Lemma 7 is to ensure that the space bound of the right-hand side of (37) is $o(\tau^a)$, which can be done with the additional constraint on d that $d < a$.

For $\ell = 2$ the upper bound on d approaches $\frac{a}{2}$ from below as c approaches 1 from above. In the case $\ell \geq 3$ the upper bound on d imposed by the conditions other than (38) approaches $\frac{\ell-1}{\ell}a$ as c approaches 1; condition (38) implies an upper bound of $1/\sqrt{\ell}$ for $a = 1$ and a somewhat weaker bound for smaller values of a. However, we can achieve a contradiction for larger d as c approaches 1 and $\ell \geq 3$ using the following argument. For $\tau(n) \geq n^{\frac{1+a}{c+d}}$, Lemma 18 gives

$$\Sigma_3\text{TISP}[\tau, \tau^a] \subseteq \Pi_3\text{TISP}[\tau^{c\frac{c+d}{1+a}}\,\text{polylog}(\tau), \tau^d\,\text{polylog}(\tau)]$$

when $c + d \leq 1 + a$ and $d \leq ac$. When $c < \sqrt{1+a}$, we can choose $d < \min(\frac{(1+a)-c^2}{c}, a)$ and arrive at a contradiction with Lemma 7. As c approaches 1 from above, the upper bound on d approaches a from below, which gives the desired dependence. ∎

6.2. Tautology on randomized machines with one-sided error. In this section, we consider problems in the first level of the polynomial-time hierarchy and establish the time-space lower bounds of Theorem 3 for tautology on randomized machines with one-sided error. To do so, we actually prove time-space lower bounds for a more powerful class of machines, namely nondeterministic machines which are restricted to guess few bits.

THEOREM 19. *There exists positive constants b and d such that tautology cannot be solved by nondeterministic random-access machines which run in time $n^{1.759}\,\text{polylog}(n)$ and space n^d and that nondeterministically guess only n^b bits.*

By way of the space-bounded derandomization of Theorem 11, a space-bounded randomized machine with one-sided error can be made to use very few random bits. Since a randomized machine with one-sided error is also a special type of nondeterministic machine, we can view the one-sided error machines obtained by Theorem 11 as nondeterministic machines which guess very few random bits. Thus, the time-space lower bounds of Theorem 3 follow as a corollary to Theorem 19.

Proof of Theorem 3. Let b^* be the value of b given by Theorem 19, and let d^* be the value of d. By the derandomization of Theorem 11, if tautology can be solved in $\mathrm{RTISP}[n^c, n^d]$, then it can also be solved in $\exists^{n^d \log n}\mathrm{DTISP}[n^c \operatorname{polylog}(n), n^d]$. Thus by Theorem 19, tautology is not in $\mathrm{RTISP}[n^{1.759}, n^d]$ for any $d < \min(b^*, d^*)$. □

We now focus on proving Theorem 19 using the ideas outlined in section 1.2. In this setting, the hypothesis of the indirect diagonalization argument becomes $\mathrm{NTIME}[n] \subseteq \forall^{n^b}\mathrm{DTISP}[n^c, n^d]$. This unlikely scenario certainly yields an efficient complementation, and it is actually strong enough to allow for something more: Under this hypothesis, we can improve the space-bounded speedups of (2) and (4) for certain values of b, c, and d.

LEMMA 20. *Suppose that*

$$\mathrm{NTIME}[n] \subseteq \forall^{n^b}\mathrm{DTISP}[n^c, n^d] \tag{39}$$

for some constants b, $c \geq 1$, and d such that $c + d \leq 2$ and $b \leq c + d - 1$. Then for any time function T, space function S, and integers $i \geq 0$ and $k \geq 2$,

$$\mathrm{DTISP}[T, S] \subseteq \Pi_k\mathrm{TIME}[(TS^{k-1})^{\frac{\gamma_i}{(k-2)\gamma_i+1}} + n + S], \tag{40}$$

where $\gamma_0 = \frac{1}{2}$ and $\gamma_{i+1} = \frac{(c+d)\gamma_i}{(c+d)\gamma_i+1}$.

We point out some facts about the sequence $(\gamma_i)_i$ in order to clearly assess the speedup represented by Lemma 20. From the definition, we can see that $\gamma_{i+1} \leq \gamma_i$ if and only if $\gamma_i \leq \gamma_{i-1}$. It follows that the sequence $(\gamma_i)_i$ is monotonic. Since the transformation $x \mapsto \frac{(c+d)x}{(c+d)x+1}$ has a unique attractive fixed point at $1 - \frac{1}{c+d}$, $(\gamma_i)_i$ converges to this value. Specifically, when $c + d < 2$, this fixed point is less than $\frac{1}{2} = \gamma_0$, so in this case $(\gamma_i)_i$ decreases monotonically to $1 - \frac{1}{c+d}$.

When S is small, Lemma 20 essentially offers a $((k-2) + \frac{1}{\gamma_i})$th-root speedup of a $\mathrm{DTISP}[T, S]$ machine on a Π_k-machine, provided this running time remains at least linear. From the convergence properties of $(\gamma_i)_i$, we can see that this speedup approaches the $(k - 1 + \frac{1}{c+d-1})$th-root as i increases. Recall that the unconditional speedup offered by (4) gives a similar kth-root speedup. Thus, when hypothesis (39) holds for $c + d < 2$, Lemma 20 offers a greater speedup than we had unconditionally.

To prove Lemma 20, we start with the case $k = 2$ by inductively deriving better and better speedups of $\mathrm{DTISP}[T, S]$ into Π_2. For the case $k > 2$, we use $k - 2$ alternations for a speedup as in (3) and then one more alternation to speed up the final deterministic phase in (3) by applying (40) for $k = 2$. An optimal choice of the number of blocks B yields the result.

Proof of Lemma 20. We prove the case $k = 2$ by induction on i. In particular, we need to prove

$$\mathrm{DTISP}[T, S] \subseteq \Pi_2\mathrm{TIME}[(TS)^{\gamma_i} + n + S]. \tag{41}$$

For the base case, (41) holds unconditionally for $i = 0$ by the standard square root speedup of (2). For the inductive step $i \to i+1$, consider a $\mathrm{DTISP}[T, S]$ computation. Using the Π_2-version of the inclusion (1), we speed up this computation in Π_2:

$$\mathrm{DTISP}[T, S] \subseteq \forall^{BS}\underbrace{\exists^{\log B}\mathrm{DTISP}[T/B, S]}_{\text{(i)}}.$$

We can see that (i) represents a computation in nondeterministic time $O(T/B)$ taking an input of length $O(n + BS)$. Thus, hypothesis (39) gives a simulation of (i)

which yields

$$\mathrm{DTISP}[T,S] \subseteq \underbrace{\forall^{BS}\forall^{(T/B+n+BS)^b}}_{\text{(ii)}}\underbrace{\mathrm{DTISP}[(T/B+n+BS)^c,(T/B+n+BS)^d]}_{\text{(iii)}}.$$

Notice that the final space-bounded deterministic stage (iii) takes an input of size $O(n+BS+(T/B)^b)$ provided $b \leq 1$, so that the inductive hypothesis yields a simulation of this stage in

$$\Pi_2\mathrm{TIME}[((T/B+n+BS)^{c+d})^{\gamma_i}+n+BS+(T/B)^b+(T/B+n+BS)^d].$$

Merging the initial universal phase of this simulation with that represented by (ii) and noting that $B \geq 1$, we see that we have arrived at a simulation of $\mathrm{DTISP}[T,S]$ on a Π_2-machine running in time big-O of

$$BS+(T/B+n+BS)^{\max(b,d)}+((T/B+n+BS)^{c+d})^{\gamma_i}+n.$$

To simplify this, notice that when $c+d \leq 2$, we have that $(c+d)\gamma_i \leq 1$ (since $\gamma_i \leq \frac{1}{2}$). If $c \geq 1$, we have $d \leq (c+d)\gamma_i$ (since $\gamma_i \geq 1-\frac{1}{c+d}$). Furthermore, if $b \leq c+d-1$, we have $b \leq (c+d)\gamma_i$ (since $\gamma_i \geq 1-\frac{1}{c+d}$). Under these conditions, the above running time simplifies to big-O of

$$BS+(T/B)^{(c+d)\gamma_i}+n.$$

To minimize this running time up to a constant factor, we choose a value for B such that $BS=(T/B)^{(c+d)\gamma_i}$, namely $B^* \doteq (\frac{T^{(c+d)\gamma_i}}{S})^{1/((c+d)\gamma_i+1)}$. If $B^* \geq 1$, this choice results in a running time in big-O of

$$(TS)^{\frac{(c+d)\gamma_i}{(c+d)\gamma_i+1}}+n=(TS)^{\gamma_{i+1}}+n.$$

On the other hand, if $B^* < 1$, then $B=1$ is the best we can do. This yields a running time of $O(S+n)$. In either case, $O((TS)^{\gamma_{i+1}}+n+S)$ is an upper bound on the running time. By induction, (41) holds for all $i \geq 0$ and b, c, and d as above.

Now that we have established (41), we use it to establish (40) for $k > 2$. The first step is to use (3) to speed up a $\mathrm{DTISP}[T,S]$-machine on a Π_{k-1}-machine. This yields

$$\mathrm{DTISP}[T,S] \subseteq \underbrace{\forall^{BS}\exists^{BS}\cdots Q^{\log B}}_{k-1}\underbrace{\mathrm{DTISP}[T/B^{k-2},S]}_{(\alpha)}, \tag{42}$$

where Q is $\forall$ if k is even, and $\exists$ if k is odd.

We can see that (α) represents a computation taking an input of size $O(n+BS)$ and running in time T/B^{k-2} and space S. Provided b, c, and d satisfy the constraints of the lemma, (41) gives simulations of (α) on Π_2-machines running in time

$$O\left((TS/B^{k-2})^{\gamma_i}+n+BS\right),$$

for $B \geq 1$. Since deterministic classes are closed under complement, we also get simulations on Σ_2-machines running in the same amount of time. Choosing the former if $Q=\forall$ in (42), and the latter otherwise, the alternating stages align properly so that replacing (α) in this manner adds only one alternation. Overall, we arrive at a simulation of $\mathrm{DTISP}[T,S]$ by a Π_k-machine running in the above time bound. To

minimize this running time up to a constant factor, we choose B so that the two terms depending on B are equal. This occurs at the value

$$B^\dagger \doteq (T^{\gamma_i} S^{\gamma_i - 1})^{\frac{1}{(k-2)\gamma_i + 1}}.$$

When $B^\dagger \geq 1$, such a choice yields the running time $O\left((TS^{k-1})^{\frac{\gamma_i}{(k-2)\gamma_i+1}} + n\right)$. If $B^\dagger < 1$, then $B = 1$ is the best we can do and the running time becomes $O(S + n)$. In both cases, we obtain an upper bound of

$$O\left((TS^{k-1})^{\frac{\gamma_i}{(k-2)\gamma_i+1}} + n + S\right)$$

on the running time, which proves the claim. □

We now use a bootstrapping argument to derive a series of increasingly efficient complementations of the linear-time hierarchy at higher and higher levels. At each level, the improved speedup granted by Lemma 20 for sufficiently small b and d allows a more efficient complementation than the unconditional speedup of (4), which is key to obtaining quantitatively stronger lower bounds.

LEMMA 21. *For any constants $1 \leq c < 2$, $\epsilon > 0$, and integer $\ell \geq 2$, there exist positive constants b and d such that if*

$$\text{NTIME}[n] \subseteq \forall^{n^b}\text{DTISP}[n^c, n^d], \tag{43}$$

then

$$\Sigma_\ell\text{TIME}[n] \subseteq \Pi_\ell\text{TIME}[n^{c_\ell + \epsilon}], \tag{44}$$

where

$$c_\ell = \begin{cases} c(c-1) & \text{for } \ell = 2, \\ \frac{c^2(c-1)\prod_{j=2}^{\ell-1} c_j}{(\ell-1)(c-1)+1} & \text{otherwise.} \end{cases} \tag{45}$$

Closed forms for the exponent c_ℓ defined by (45) become rather complex. One can show by induction that

$$c_\ell = \frac{c^{3\cdot 2^{\ell-3}}(c-1)^{2^{\ell-2}}}{((\ell-1)(c-1)+1)\cdot\prod_{k=3}^{\ell-1}((k-1)(c-1)+1)^{2^{\ell-k-1}}}$$

for $\ell > 2$.

Proof of Lemma 21. Let c and ϵ be given. We argue by induction that we can choose b and d appropriately so that (43) yields the desired inclusion. For $\ell = 2$, we use the hypothesis to obtain a DTISP simulation of $\Sigma_2\text{TIME}[n]$ when $b \leq c$:

$$\begin{aligned}\Sigma_2\text{TIME}[n] = \exists^n\forall^n\text{DTIME}[n] &\subseteq \exists^n\exists^{n^b}\text{DTISP}[n^c, n^d] \subseteq \text{NTIME}[n^c] \\ &\subseteq \forall^{n^{bc}}\text{DTISP}[n^{c^2}, n^{cd}].\end{aligned}$$

The input to the final deterministic stage is of size $n + n^{bc}$, so applying Lemma 20 to simulate this stage yields

$$\Sigma_2\text{TIME}[n] \subseteq \forall^{n^{bc}}\text{DTISP}[n^{c^2}, n^{cd}] \subseteq \Pi_2\text{TIME}[(n^{c(c+d)})^{\gamma_i} + n + n^{bc} + n^{cd}].$$

As $i \to \infty$, the exponent $c(c+d)\gamma_i \to c(c+d-1)$. For d such that this limit is at most $c(c-1)+\frac{\epsilon}{2}$, we can choose i large enough so that the exponent is at most $c(c-1)+\epsilon$. Under the additional constraints $b, d \leq c-1$, the running time becomes $O(n^{c(c-1)+\epsilon}+n)$. Therefore, when $c(c-1)+\epsilon \geq 1$, we have the desired inclusion. In fact, this is the only case we need to consider, for if $c(c-1)+\epsilon < 1$, we can find $a > 1$ such that $ac(c-1)+\epsilon = 1$ and apply the above argument to $\Sigma_2\text{TIME}[n^a]$. This yields the inclusion $\Sigma_2\text{TIME}[n^a] \subseteq \Pi_2\text{TIME}[n]$, which contradicts Lemma 6. Therefore, the claim holds for $\ell = 2$.

Now suppose that for $2 \leq k < \ell$, $\Sigma_k\text{TIME}[n] \subseteq \Pi_k\text{TIME}[n^{c_k+\epsilon'}]$ for an ϵ' to be determined later. Let $b_{k,\epsilon'}$ and $d_{k,\epsilon'}$ denote the appropriate values of b and d given by the inductive hypothesis for the complementation at Σ_k to hold. To show that the desired complementation holds for Σ_ℓ, we first derive a simulation of such a computation in $\forall^n\text{DTISP}[n^{O(1)}, n^d]$ and then use Lemma 20 to achieve a faster simulation in Π_ℓ.

We accomplish the former by iteratively deriving simulations one level lower in the polynomial-time hierarchy. At step $j = 0, \ldots, \ell-3$, we start with a $\Sigma_{\ell-j}$-machine and use a complementation given by the inductive hypothesis to obtain a simulation by a $\Sigma_{\ell-j-1}$-machine. This follows by complementing the computation following the initial existential stage of the $\Sigma_{\ell-j}$-machine, which is a $\Pi_{\ell-j-1}$-computation. Thus, when the hypothesis holds for $b \leq b_{\ell-j-1,\epsilon'}$ and $d \leq d_{\ell-j-1,\epsilon'}$, we derive a simulation with one less alternation while raising the running time to the power of $c_{\ell-j-1}+\epsilon'$. This lets us write

$$\begin{aligned}\Sigma_\ell\text{TIME}[n] &\subseteq \Sigma_{\ell-1}\text{TIME}[n^{c_{\ell-1}+\epsilon'}]\\ &\subseteq \Sigma_{\ell-2}\text{TIME}[n^{(c_{\ell-1}+\epsilon')(c_{\ell-2}+\epsilon')}]\\ &\cdots\\ &\subseteq \Sigma_2\text{TIME}[n^{(c_{\ell-1}+\epsilon')(c_{\ell-2}+\epsilon')\cdots(c_2+\epsilon')}].\end{aligned}$$

Defining $C_{\ell,\epsilon'} \doteq (c_{\ell-1}+\epsilon')(c_{\ell-2}+\epsilon')\cdots(c_2+\epsilon')$ and applying hypothesis (43) twice to the latter simulation (as in the base case), we obtain the desired simulation:

$$\Sigma_\ell\text{TIME}[n] \subseteq \underbrace{\forall^{n^{bcC_{\ell,\epsilon'}}}}_{(\alpha)}\underbrace{\text{DTISP}[n^{c^2C_{\ell,\epsilon'}}, n^{cdC_{\ell,\epsilon'}}]}_{(\beta)}.$$

The input to (β) is of size $O(n+n^{bcC_{\ell,\epsilon'}})$, so applying Lemma 20 to this stage and absorbing the universal stage (α) gives

$$\Sigma_\ell\text{TIME}[n] \subseteq \Pi_\ell\text{TIME}[\underbrace{(n^{c(c+(\ell-1)d)C_{\ell,\epsilon'}})^{\frac{\gamma_i}{(\ell-2)\gamma_i+1}}}_{(*)} + n + \underbrace{n^{bcC_{\ell,\epsilon'}}+n^{cdC_{\ell,\epsilon'}}}_{(**)}], \tag{46}$$

for small enough b and d and any integer $i \geq 0$. When d is yet further restricted, the exponent of the term $(*)$ approaches $\frac{c^2(c-1)C_{\ell,\epsilon'}}{(\ell-1)(c-1)+1}+\frac{\epsilon}{4}$ as i grows. This allows the choice of i large enough, so this exponent is at most $\frac{\epsilon}{4}$ away from its limit point, namely at most $\frac{c^2(c-1)C_{\ell,\epsilon'}}{(\ell-1)(c-1)+1}+\frac{\epsilon}{2}$. We next choose ϵ' small enough so that all of the terms in the exponent of $(*)$ involving ϵ' sum to at most $\frac{\epsilon}{2}$. Under these circumstances, an upper bound for the exponent of $(*)$ is

$$\frac{c^2(c-1)C_{\ell,\epsilon'}}{(\ell-1)(c-1)+1}+\frac{\epsilon}{2} \leq \frac{c^2(c-1)\prod_{i=2}^{\ell-1}c_i}{(\ell-1)(c-1)+1}+\epsilon = c_\ell+\epsilon.$$

When we also have $b, d \leq \frac{c(c-1)}{(\ell-1)(c-1)+1}$, the term $(*)$ dominates $(**)$, so the Π_ℓ-simulation represented by (46) runs in time $O(n^{c_\ell+\epsilon} + n)$. When $c_\ell + \epsilon \geq 1$, this shows that (44) holds for b and d small enough to meet all of the above constraints. This is the only case we need to consider, since $c_\ell + \epsilon < 1$ results in a contradiction to Lemma 6 by applying the above argument to $\Sigma_\ell \mathrm{TIME}[n^a]$ for an appropriate $a > 1$ (as in the step for $\ell = 2$). □

Lemma 21 gives a series of complementations which are increasingly unlikely and eventually contradict Lemma 6 for certain values of c. We obtain Theorem 19 by analyzing the behavior of the sequence of exponents c_ℓ defined by (45).

Proof of Theorem 19. (See page 22 for the statement.) For any integer $\ell \geq 2$, consider c_ℓ defined by (45) as a function of c. One can show by induction on ℓ that c_ℓ monotonically grows from $c_\ell = 0$ at $c = 1$ to infinity. By continuity, there exists a unique value c_ℓ^* at which c_ℓ equals 1. For values $c < c_\ell^*$, Lemma 21 gives a contradiction to Lemma 6 for a choice of ϵ small enough such that $c_\ell + \epsilon < 1$. Thus, we can rule out simulations of NTIME$[n]$ in $\forall^{n^b}\mathrm{DTISP}[n^c, n^d]$ for all $c < c_\ell^*$ and b, d given by Lemma 21.

By (45), we have that at $c = c_\ell^*$,

$$c_{\ell+1} = \frac{c_{\ell+1}}{c_\ell} = \left(1 - \frac{1}{\ell + \frac{1}{c-1}}\right) \cdot c_\ell < 1.$$

The monotonicity of $c_{\ell+1}$ then implies that $c_{\ell+1}^* > c_\ell^*$; i.e., the sequence $(c_\ell^*)_\ell$ increases. Numerical calculations show that $c_{14}^* \approx 1.759708$ and $c_{15}^* \approx 1.759719$, which is enough to prove the claimed lower bound of $n^{1.759}\,\mathrm{polylog}(n)$. □

7. Further research. The techniques discussed in this work allow us to establish time-space lower bounds for QSAT_ℓ on two-sided error randomized machines for $\ell \geq 2$. They do not seem to extend to the first-level problems of satisfiability or tautology in a straightforward way. This is due only to our inability to exploit the assumption NTIME$[n] \subseteq \mathrm{BPTISP}[t, s]$ to obtain an efficient complementation at some level of the polynomial-time hierarchy. Thus, establishing time-space lower bounds for satisfiability on randomized machines with two-sided error remains open.

We employed and further developed a technique from [24] to improve the known lower bounds for satisfiability on deterministic machines. The original technique also leads to improved lower bounds for QSAT_ℓ with $\ell \geq 2$ on deterministic machines. For example, it allows the boosting of the time lower bound for QSAT_2 on deterministic subpolynomial-space machines from n^2 [9] to $n^{2.761}$ [24]. Although we were able to find purchase in adopting these techniques to establish better lower bounds for randomized machines with one-sided error, we have been unable to adopt them to improve our results for machines with two-sided error. As a next step, we suggest finding a way to extend the improved lower bounds for QSAT_ℓ with $\ell \geq 2$ on deterministic machines to randomized machines with two-sided error.

Appendices. We now prove some results on the complexity of Nisan's generator (Theorem 11) and on deterministic amplification by random walks on a Gabber–Galil expander graph (Lemma 13).

Appendix A. Nisan's generator. Theorem 11 follows from an analysis of a time- and space-efficient implementation of Nisan's pseudorandom generator using fast Fourier transform (FFT) multiplication methods to quickly evaluate and invert linear hash functions. In fact, we prove a somewhat stronger version of Theorem 11.

THEOREM 22. *Any randomized machine M running in time T and space S with error ϵ can be simulated by another randomized machine that runs in time $O(T \log^2 S \log\log S)$ and space $O(S \log T)$ and that uses only $O(S \log T)$ random bits. If two-way access to the random bits is allowed, the space requirement is reduced to $O(S)$. The error of the simulation is $\epsilon + 2^{-S}$ and is one-sided if M has one-sided error.*

Theorem 22 gives a tighter time bound than the bound of $O(T \operatorname{polylog}(T))$ stated in Theorem 11. Our arguments in sections 5 and 6 are not noticeably improved by using the tighter bound stated in Theorem 22, so we use the simpler bounds of Theorem 11 there for clarity. We state the tighter bound here because it may be of independent interest.

Before proving Theorem 22, we introduce Nisan's pseudorandom generator [20] and discuss some of its properties. Define

$$G_{m,k} : \{0,1\}^m \times H_m^k \to (\{0,1\}^m)^{2^k},$$

where H_m is a family of two-universal hash functions $h : \{0,1\}^m \to \{0,1\}^m$ [4]. The evaluation of $G_{m,k}$ is defined recursively as

$$G_{m,k}(y, h_1, \ldots, h_k) = \begin{cases} y & \text{if } k = 0, \text{ else,} \\ G_{m,k-1}(y, h_1, \ldots, h_{k-1}) \circ G_{m,k-1}(h_k(y), h_1, \ldots, h_{k-1}), \end{cases}$$

where "$\circ$" denotes concatenation. Given a randomized machine M running in time T and space S, we define $G \doteq G_{m,k}$ for $k = \log \frac{T}{m}$, where $m = \Theta(S)$ will be determined later. The simulation of M proceeds with the output of G as the random string, one block of length m at a time. Nisan proves that G fools M in the following sense.

THEOREM 23 (Nisan [20]). *There exists a constant ν such that if M is a randomized machine running in time T and space S on input x, then for $m \geq \nu \cdot S$ and $k = \log \frac{T}{m}$,*

$$\left| \Pr_r[M(x,r) \text{ accepts}] - \Pr_{y,h_1,\ldots,h_k}[M(x, G_{m,k}(y, h_1, \ldots, h_k)) \text{ accepts}] \right| \leq 2^{-S},$$

where $M(x,r)$ denotes the outcome of running M on input x and random string r.

This satisfies the error requirements of Theorem 22, so all that remains to prove Theorem 22 is to show how to simulate M on the random string $G(y, h_1, \ldots, h_k)$ within the correct bounds on the time, space, and randomness.

We start with the standard simulation that computes G in a blockwise fashion, where each subsequent m-bit block is computed after m simulation steps of M using the current block. One way to do this is to compute each block from scratch, namely, apply the appropriate sequence of at most k hash functions to the m-bit seed y. Although this technique is good enough to derive our main results, we can do slightly better, namely by a factor of $O(\frac{\log S}{\log(T/S)})$. This improvement follows by computing each block in a recursive manner, which avoids the calculations that the "from scratch" method does over and over again. We now work out the details of our simulation to complete the proof of Theorem 22.

Proof of Theorem 22. To accommodate the approach described above, we choose $m = \Theta(S)$ such that $m \geq \nu \cdot S$ and is of the form $2 \cdot 3^q$ for some integer $q \geq 0$. The latter guarantees a simple explicit formula for an irreducible polynomial of degree m over GF(2), namely $z^{2 \cdot 3^q} + z^{3^q} + 1$ [16, Thm. 1.1.28, p. 13]. We choose H_m to be the set of all invertible linear mappings from GF(2^m) to GF(2^m), i.e., all functions of the

form $x \mapsto ax+b$ where $a, b \in \mathrm{GF}(2^m)$ and $a \neq 0$ [4].[2] Such functions can be described by $2m$ bits, so that the input to G can be described by $(2k+1)m = O(S \log T)$ bits. This meets the requirements on the randomness.

To reach the desired time and space bounds, we must be able to evaluate the functions in H_m much faster than the naive bound of $O(m^2)$. Using FFT multiplication techniques based on those of Schönhage and Strassen and exploiting the sparseness of the above irreducible polynomial, we can evaluate $h \in H_m$ in time $O(m \log m \log\log m)$ and space $O(m)$. These fast multiplication techniques can be combined with the extended Euclidean algorithm to invert $h \in H_m$ in time $O(m \log^2 m \log\log m)$ and space $O(m)$. See [11, Cor. 11.8, p. 319] for more details. We use these algorithms to output the blocks of $G_{m,k}(y, h_1, \ldots, h_k)$ recursively with small space overhead. Specifically, we define the procedure P_m which uses global registers containing $k \geq 0$, $y \in \{0,1\}^m$, and $h_1, h_2, \ldots, h_k \in H_m$ to perform the following steps:

1. **If** $k > 0$:
2. $k \leftarrow k-1$;
3. Recursively **call** P_m;
4. $y \leftarrow h_{k+1}(y)$;
5. Recursively **call** P_m;
6. $y \leftarrow h_{k+1}^{-1}(y)$; $k \leftarrow k+1$ and **return**.
7. **Else output** y and **return**.

The output of $P_m(k, y, h_1, \ldots, h_k)$ is exactly $G_{m,k}(y, h_1, \ldots, h_k)$. Evaluating P_m takes 2^k recursive calls, each accompanied by an evaluation of h_i or h_i^{-1}. Thus, the overall time complexity to output $G_{m,k}$ is $O(2^k \cdot m \log^2 m \log\log m)$. By replacing y with $h_k(y)$ instead of writing down $h_k(y)$ separately on the worktape, only a constant amount of space overhead is required for each level. Thus, the space requirement is $O(m+k)$. For the settings of G, $m = O(S)$, and $k = \log \frac{T}{S} + O(1)$, the time becomes $O(T \log^2 S \log\log S)$, while the space is $O(S + \log \frac{T}{S})$, which is $O(S)$ since $T \leq 2^S$ without loss of generality. This assumes that the hash functions can be accessed repeatedly, so there is an additional space cost of $O(S \log T)$ to copy the hash functions from the random tape to the worktape, bringing the space bound to $O(S \log T + S) = O(S \log T)$. However, if the simulation has two-way access to the random tape, this cost is avoided.

Our simulation runs M for $O(S)$ steps every time P_m outputs a block of G, for a total of T steps while using space S. Thus, the above time and space bounds for computing G hold for the simulation as a whole. □

Appendix B. Deterministic amplification. We now prove Lemma 13, beginning with a brief discussion of some properties of the amplification given by Theorem 10. Let M be a randomized machine that runs in time T and space S and that uses R random bits. We may assume that M has an error of at most some small constant δ to be determined later, since this can be achieved with only a constant overhead from a machine with any error bounded away from $1/2$. The amplified machine M' given by Theorem 10 interprets its random string r' of length $O(R)$ as an initial vertex in a Gabber–Galil graph followed by $O(R)$ edge labels (of constant size) specifying the edges on a walk in the graph. Formally, this graph is described as follows.

DEFINITION 24 (Gabber–Galil graph [10]). *The Gabber–Galil graph* $\mathrm{GG}(m)$ *is a graph with vertices* $\mathbb{Z}_m \times \mathbb{Z}_m$ *of degree* 5 *where the vertices adjacent to* $(x, y) \in$

[2] Excluding the noninvertible functions ($a = 0$) introduces a small bias. Although our family H_m is not perfectly two universal, it is close enough for our purposes.

$\mathbb{Z}_m \times \mathbb{Z}_m$ are the five pairs (x', y') obtained from the matrix multiplication $[x', y', 1]^T = A_i[x, y, 1]^T$ (over $\mathbb{Z}_m$) for $1 \le i \le 5$, where

$$A_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad A_3 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix},$$
$$A_4 = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad A_5 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

We now state a useful lemma which says that the vertex at the end of a path of length p in $\mathrm{GG}(2^k)$ can be found in time quasilinear in p and k, an improvement on the naive bound of $O(pk)$. We leave the proof for later.

LEMMA 25. *Given a vertex (x, y) of the graph $\mathrm{GG}(2^k)$ and a path π of length p indicated by edge labels $(e_1, e_2, \ldots, e_p)$, where $1 \le e_i \le 5$ for $1 \le i \le p$, the vertex connected to (x, y) by π can be determined in time $O(m \operatorname{polylog}(m))$ and space $O(m)$ where $m = k + p$.*

For the purposes of Theorem 10, the vertices of the Gabber–Galil graph must be described by R bits corresponding to the possible random strings for M. To this end, we choose $\mathrm{GG}(2^{R/2})$. Given input x and random string r', M' proceeds by deterministically carrying out the walk of length $p = O(R)$ on $\mathrm{GG}(2^{R/2})$ indicated by r'. Every β steps, where β is some constant, M' simulates M on input x and a random string corresponding to the label of the current vertex on the walk. M' accepts if a majority of these trials accept. As each edge relation can be computed by simple arithmetic in time $O(R)$ and space $O(R)$, and running M requires time T and space S, M' runs in time $O(R^2 + RT) = O(RT)$ and space $O(R + S)$. Cohen and Wigderson [5] and Impagliazzo and Zuckerman [13] show that when β is large enough and M has error smaller than some constant δ, the trials specified by a randomly chosen r' are close enough to uniform so that M' errs only with probability 2^{-R}. This establishes Theorem 10.

We now prove Lemma 13, giving a more time-efficient manner to determine if M' accepts on x and r' at the cost of using alternations.

Proof of Lemma 13. To arrive at Lemma 13, we show how to use alternations to verify if there is a majority of trials on the walk given by the random string r' where M accepts, in such a way that the final deterministic phase need only to simulate M once. Specifically, given input x and random string r', we can express the acceptance condition of M' as

$$(\exists Z \subseteq \{1, 2, \ldots, R'\}, |Z| = \lceil R'/2 \rceil)(\forall i \in Z) M(x, r_i) \text{ accepts}, \tag{47}$$

where R' is the number of trials of M specified by r', and r_i is the random string produced for the ith trial. Observe that this describes a Σ_2-computation which accepts if and only if M' accepts. The initial existential phase guesses the characteristic string of the set Z consisting of $\lceil R'/2 \rceil$ indices of the R' trials, for a total of $R' = O(R)$ bits. The universal phase guesses $\log R' = O(\log R)$ bits to determine the index of a trial to verify. The final deterministic stage must first determine r_i and then run M on input x and random string r_i. Once the former has been computed, the latter task takes time T and space S. Since r_i corresponds to the label of the (βi)th vertex on the walk in $\mathrm{GG}(2^{R/2})$ specified by r', Lemma 25 shows that r_i can be computed in time $O(R \operatorname{polylog}(R))$. Therefore, the final deterministic stage takes

time $O(T + R\,\mathrm{polylog}(R))$ and space $O(R + S)$. All told, we have shown that (47) is a computation in

$$\exists^R \forall^{\log R} \mathrm{DTISP}[T + R\,\mathrm{polylog}(R), R + S],$$

which accepts if and only if M' accepts. This completes the proof. □

All that remains is to establish Lemma 25, which follows from a divide-and-conquer strategy to efficiently evaluate a product of p matrices A_i.

Proof of Lemma 25. Throughout this proof, we use the fact that multiplication of b-bit integers can be done in time $O(b\,\mathrm{polylog}(b))$ and space $O(b)$. This follows from the FFT techniques of Schönhage and Strassen [11, Thm. 8.24, p. 240]. Let

$$A \doteq A_{e_p} A_{e_{p-1}} \cdots A_{e_1}.$$

Then the vertex (x', y') connected to (x, y) by π satisfies $[x', y', 1]^T = A[x, y, 1]^T \bmod 2^k$. Therefore, computing (x', y') reduces to computing A and multiplying by the vector $[x, y, 1]^T$ modulo 2^k.

We accomplish the latter with a divide-and-conquer strategy. Namely, we split the product approximately in half and recursively compute the subproducts $A_{e_p} A_{e_{p-1}} \cdots A_{e_{\lfloor p/2 \rfloor + 1}} \doteq B$ and $A_{e_{\lfloor p/2 \rfloor}} A_{e_{\lfloor p/2 \rfloor - 1}} \cdots A_{e_1} \doteq C$. It can be shown by induction that any product of p matrices A_i has entries bounded by 2^{p-1}, since each column of any matrix A_i has at most two nonzero entries, which are ones. This shows that once B and C are computed, A can be computed as $A = BC$ by $O(1)$ multiplications and additions of integers of bit length $p/2$, so we require time $O(p\ \mathrm{polylog}(p))$ in addition to the recursive calls. Thus, by following this strategy, we can see that at each recursive call to compute the product of q matrices, we solve two subproblems of size $q/2$ and do an additional amount of work which is quasilinear in q and uses space $O(q)$. Thus, A can be computed in total time $O(p\,\mathrm{polylog}(p))$ and space $O(p)$.

By our observation on the size of the product matrix entries, we also know that the entries of the matrix A are at most 2^{p-1}. Therefore, computing the product $A[x, y, 1]^T$ and reducing modulo 2^k can be done in time $O(m\,\mathrm{polylog}(m))$ and space $O(m)$ as desired. □

We point out that another natural way to arrive at Theorem 14 is to use expander walks to generate the shift vectors for Lautemann's simulation in lieu of amplifying the confidence of the simulated algorithm as above. While the effect of the latter is to reduce the *number* of shift vectors needed, the former allows a large number of "good" shifts to be described by *very few bits*. Briefly, the hitting property of expanders guarantees that the shifts satisfy the needed property (i.e., the shifts of the accepting set cover the entire set of random strings) with approximately the same probability when they are chosen by a random walk on a Gabber–Galil graph as when they are chosen independently. Thus, we can generate a set of $O(R)$ good shifts in the initial stage of the simulation with only $O(R)$ bits. Furthermore, each shift can be computed efficiently by Lemma 25, so we can avoid the $O(R)$ blowup in the running time of the final deterministic stage by using an additional alternation to verify that M accepts on some shift. This approach leads to a simulation that matches the parameters of Theorem 14.

Acknowledgments. We would like to thank Bess Berg as well as the anonymous referees of ICALP 2005 and SIAM for their helpful comments. We are grateful to Emanuele Viola for pushing us to optimize the space parameter in Theorems 1 and 2.

REFERENCES

[1] E. Allender, M. Koucky, D. Ronneburger, S. Roy, and V. Vinay, *Time-space tradeoffs in the counting hierarchy*, in Proceedings of the 16th IEEE Conference on Computational Complexity, IEEE, Los Alamitos, CA, 2001, pp. 295–302.

[2] J. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity* I, EATCS Monogr. Theoret. Comput. Sci. II, Springer-Verlag, Berlin, 1988.

[3] P. Beame, M. Saks, X. Sun, and E. Vee, *Time-space trade-off lower bounds for randomized computation of decision problems*, J. ACM, 50 (2003), pp. 154–195.

[4] L. Carter and M. Wegman, *Universal hash functions*, J. Comput. System Sci., 18 (1979), pp. 143–154.

[5] A. Cohen and A. Wigderson, *Dispersers, deterministic amplification, and weak random sources*, in Proceedings of the 30th IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 1989, pp. 14–19.

[6] S. Cook, *Short propositional formulas represent nondeterministic computations*, Inform. Process. Lett., 26 (1988), pp. 269–270.

[7] S. Diehl and D. van Melkebeek, *Time-space lower bounds for the polynomial-time hierarchy on randomized machines*, in Proceedings of the 32nd International Colloquium On Automata, Languages and Programming, Springer-Verlag, Berlin, 2005, pp. 982–993.

[8] L. Fortnow, *Time-space tradeoffs for satisfiability*, J. Comput. System Sci., 60 (2000), pp. 337–353.

[9] L. Fortnow and D. van Melkebeek, *Time-space tradeoffs for nondeterministic computation*, in Proceedings of the 15th IEEE Conference on Computational Complexity, IEEE, Los Alamitos, CA, 2000, pp. 2–13.

[10] O. Gabber and Z. Galil, *Explicit constructions of linear-sized superconcentrators*, J. Comput. System Sci., 22 (1981), pp. 407–420.

[11] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, Cambridge University Press, Cambridge, UK, 2003.

[12] F. Hennie and R. Stearns, *Two-tape simulation of multitape Turing machines*, J. Assoc. Comput. Mach., 13 (1966), pp. 533–546.

[13] R. Impagliazzo and D. Zuckerman, *How to recycle random bits*, in Proceedings of the 30th IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 1989, pp. 248–253.

[14] R. Kannan, *Towards separating nondeterminism from determinism*, Math. Systems Theory, 17 (1984), pp. 29–45.

[15] C. Lautemann, *BPP and the polynomial hierarchy*, Inform. Process. Lett., 17 (1983), pp. 215–217.

[16] J. H. van Lint, *Introduction to Coding Theory*, 3rd ed., Springer-Verlag, Berlin, 1999.

[17] R. Lipton and A. Viglas, *On the complexity of SAT*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 1999, pp. 459–464.

[18] G. A. Margulis, *Explicit construction of concentrators*, Problems Inform. Transmission, 9 (1973), pp. 325–332.

[19] D. van Melkebeek, *Time-space lower bounds for NP-complete problems*, in Current Trends in Theoretical Computer Science, G. Paun, G. Rozenberg, and A. Salomaa, eds., World Scientific, River Edge, NJ, 2004, pp. 265–291.

[20] N. Nisan, *On read-once vs. multiple access to randomness in logspace*, Theoret. Comput. Sci., 107 (1993), pp. 135–144.

[21] N. Nisan, $RL \subseteq SC$, Comput. Complex., 4 (1994), pp. 1–11.

[22] C. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, MA, 1994.

[23] E. Viola, *On Probabilistic Time Versus Alternating Time*, Tech. Report TR-05-137, Electronic Colloquium on Computational Complexity, 2005. Available online at http://eccc.hpi-web.de/eccc-reports/2005/TROS-137/index.html.

[24] R. Williams, *Better time-space lower bounds for SAT and related problems*, in Proceedings of the 20th IEEE Conference on Computational Complexity, IEEE, Los Alamitos, CA, 2005, pp. 40–49.

SIAM J. COMPUT.
Vol. 36, No. 3, pp. 595–608

INFINITELY-OFTEN AUTOREDUCIBLE SETS*

RICHARD BEIGEL†, LANCE FORTNOW‡, AND FRANK STEPHAN§

Abstract. A set A is autoreducible if one can compute, for all x, the value $A(x)$ by querying A only at places $y \neq x$. Furthermore, A is infinitely-often autoreducible if, for infinitely many x, the value $A(x)$ can be computed by querying A only at places $y \neq x$. For all other x, the computation outputs a special symbol to signal that the reduction is undefined. It is shown that for polynomial time Turing and truth-table autoreducibility there are A, B, C in the class EXP of all exponential-time computable sets such that A is not infinitely-often Turing autoreducible, B is Turing autoreducible but not infinitely-often truth-table autoreducible and C is truth-table autoreducible with $g(n)+1$ queries but not infinitely-often Turing autoreducible with $g(n)$ queries. Here n is the length of the input, g is nondecreasing, and there exists a polynomial p such that $p(n)$ bounds both the computation time and the value of g at input of length n. Furthermore, connections between notions of infinitely-often autoreducibility and notions of approximability are investigated. The Hausdorff-dimension of the class of sets which are not infinitely-often autoreducible is shown to be 1.

Key words. algorithmic randomness, autoreducible sets, infinitely-often autoreducible sets, computational complexity, exponential-time computable sets, Hausdorff-dimension

AMS subject classifications. 03D15, 03D30, 68Q15, 68Q30

DOI. 10.1137/S0097539704441630

1. Introduction.

Consider a set where every element is chosen independently at random. Intuitively the membership of x should not depend on the membership of the other elements. Indeed, random sets are not *autoreducible*; that is, it is impossible to compute for every x the membership of x from the membership of the $y \neq x$. Ebert [12, 13] gives the surprising result that one can nevertheless compute correctly the membership of an infinite number of elements of a random set from the membership of the other ones. The present work extends the study of this notion, called *infinitely-often autoreducible*.

Trakhtenbrot [24] introduced in the recursion theoretic context the notion of autoreducibility. There are many natural examples of autoreducible sets; for example, any index set B of partial recursive functions is autoreducible: By the Padding Lemma there is a recursive strictly increasing function p such that, for all x, $\varphi_x = \varphi_{p(x)}$. It follows that one can for given x compute the value $p(x)$ which is different from x and query whether $p(x) \in B$. As $\varphi_x = \varphi_{p(x)}$, one has that $x, p(x)$ are either both

*Received by the editors March 3, 2004; accepted for publication (in revised form) January 18, 2006; published electronically August 29, 2006.
http://www.siam.org/journals/sicomp/36-3/44163.html

†Department of Computer and Information Sciences, Temple University, Wachman Hall (038-24), 1805 North Broad St., Philadelphia, PA 19122-6094 (beigel@cis.temple.edu).

‡Department of Computer Science, University of Chicago, 1100 E. 58th St., Chicago, IL 60637 (fortnow@cs.uchicago.edu). This research was done while the second author was at the NEC Research Institute.

§Department of Mathematics and School of Computing, National University of Singapore, 2 Science Drive 2, Singapore 117543, Republic of Singapore (fstephan@comp.nus.edu.sg). This author's research was supported by the Deutsche Forschungsgemeinschaft (DFG), Heisenberg grant Ste 967/1-1, while he worked at the Universität Heidelberg until August 2003. From August 2003 until June 2004, he worked at the National ICT Australia LTD, which is funded by the Australian Government's Department of Communications, Information Technology and the Arts and by the Australian Research Council through Backing Australia's Ability and the ICT Centre of Excellence Program. Since July 2004, he has worked at the National University of Singapore and has been partially supported by NUS grant R252–000–212–112.

in B or both outside B and thus one knows whether $x \in B$. Other natural examples of autoreducible sets are retraceable sets, cylinders, creative sets like the halting problem, semirecursive sets, and recursive sets; see Odifreddi [20] for the definitions of these types of sets. On one hand, autoreducible sets are quite common and there are even nonrecursive Turing degrees containing only autoreducible sets [15]. On the other hand, Trakhtenbrot [24] constructed recursively enumerable sets which are not autoreducible.

The notion of autoreducibility can easily be carried over to resource-bounded reducibilities r such as polynomial time Turing reducibility.

DEFINITION 1.1. *A set is r-autoreducible iff there is an r-reduction that computes for every x the value $A(x)$ from the oracle A without querying A at x.*

For example, a many-one EXP-complete set A satisfies that A is many-one reducible to $\overline{A}$ via some function f; that is, $A(x) = \overline{A}(f(x)) = 1 - A(f(x))$. This guarantees that $f(x) \neq x$ for all x. So one has that A is autoreducible by a polynomial time Turing reduction which asks exactly one query: What is A at $f(x)$? Knowing this, the Turing reduction defines $A(x) = 1 - A(f(x))$.

In the present work, polynomial time truth-table and Turing reducibility are considered where the number of questions might also be bounded. Truth-table reducibility is different from Turing reducibility in the sense that the place of the nth query does not depend on the oracle's answers to previous queries and one can compute an explicit polynomially sized list of places queried. So the oracle is queried at many places in parallel, and afterwards its answers are used by the program of the truth-table reduction without any further interaction with the oracle.

Also in complexity theory there are many natural examples of autoreducible sets. The set SAT is truth-table autoreducible with two queries: If ϕ is any formula with the variable u built in, then the derived formulas $\phi[u \to 0]$ and $\phi[u \to 1]$, where u has been replaced by the logical constants 0 and 1, respectively, are different from ϕ, and one can compute with two queries to SAT whether these formulas are satisfiable. Then ϕ is satisfiable iff at least one of the formulas $\phi[u \to 0]$ and $\phi[u \to 1]$ is. By the way, Schnorr [23] studied this special case where the autoreduction goes to instances shorter than the original input and called a set self-reducible if it has such an autoreduction. Buhrman et al. [11] showed that the Turing complete sets for EXP are Turing autoreducible, while some of the Turing complete sets for the class EEXPSPACE of all sets which are, for some polynomial p, computable in space $2^{2^{p(n)}}$, fail to have this property. It is unknown whether all sets that are Turing complete for EEXP are autoreducible; settling this open question would separate some complexity classes which are not yet known to be different.

Random sets are not autoreducible, but Ebert [12, 13] showed that they satisfy the following surprising variant, called infinitely-often autoreducible.

DEFINITION 1.2. *A set A is infinitely-often r-autoreducible iff there is an r-reduction M which at input x queries A only at places $y \neq x$. For infinitely many x, M computes $A(x)$ correctly; for all other x, M is undefined and signals this by outputting a special symbol.*

The result that random sets are infinitely-often autoreducible has received a lot of attention. Not only because one would not expect that it is possible to make predictions about the membership of x in a random set by looking at which other y are in, but also because Ebert [12] introduced for his proof a mathematical puzzle which was easy to understand, became famous, and had some interest in its own right: the hat problem. It was the topic of several newspaper articles, for example, in the

German weekly newspaper *Die Zeit* [10] and in the *New York Times* [22].

It is already well known that there are sets which are not infinitely-often autoreducible [7], but these examples are outside EXP, the class of all exponential time computable sets. EXP is the first deterministic time class known to contain nondeterministic polynomial time NP and polynomial space PSPACE. Therefore, it is natural to study the structure of the sets inside EXP and the main result of the present work, given in section 2, says that there is a set A in EXP which is not infinitely-often Turing autoreducible. In section 3, the major autoreducibility notions are separated by showing that there are sets in EXP which are autoreducible for the first reduction but not infinitely-often autoreducible by the second reduction; this is done in particular for Turing versus truth-table and for truth-table with $g(n)+1$ queries versus Turing with $g(n)$ queries. This second result implies the separation of truth-table versus bounded truth-table. In section 4, the relations between notions of approximability and infinitely-often autoreducibility are investigated. Finally, in section 5, it is shown that there are a lot of sets in EXP which are not autoreducible: The class of these sets has Hausdorff-dimension 1 in exponential time.

NOTATION 1.3. *The notation follows standard textbooks in recursion theory such as that of Odifreddi* [20] *with some exceptions: The function* $x \mapsto \log(x)$ *denotes the logarithm of basis* 2 *with the exception that* $\log(q) = 0$ *for* $q < 1$ *in order to avoid to deal with too many exceptions in logarithmic expressions. The term* $\log^*(x)$ *denotes the number of iterations of* log *necessary to reach a number strictly below* 1. *So,* $\log^*(0) = 0$, $\log^*(1) = 1$ *and* $\log^*(2^x) = \log^*(x)+1$. *A set* D *is called* supersparse *iff, for all* x, $D \cap \{x, x+1, \ldots, 2^x\}$ *contains at most* $\log^*(x)$ *elements.* $A\Delta D$ *denotes the symmetric difference of* A *and* D*; the set* $A\Delta D$ *is called a* supersparse variant of A *if* D *is a supersparse set. Furthermore, the notation* $O(f)$ *is generalized to* $Poly(f)$ *which is the set of all* g *such that there is a polynomial* p *with (for all* n*)* $[g(n) \leq p(f(n))]$.

2. Some set in EXP is not infinitely-often autoreducible. Note that a computation is exponential in the length (= logarithm) of x iff it is quasi-polynomial in x itself. Therefore one considers in the case of functionals quasi-polynomial time bounds. More precisely, a partial functional f which assigns to inputs of the form $A(0)A(1)\ldots A(x)$ values $a_{x+1}a_{x+2}\ldots a_y$ is called a quasi-polynomial time extension functional iff there is a constant c permitting one to compute the extensions in time $x^{\log^c(x)}$. Following Lutz [17], one can introduce the following notion of resource-bounded genericity; see [1, 3, 5, 6, 7, 18, 19] for further background on notions of resource-bounded genericity, measure and dimension.

DEFINITION 2.1. *A set* A *is general generic iff, for every quasi-polynomial time computable functional* f*,*

- *either* $f(A(0)A(1)\ldots A(x))$ *is undefined for almost all* x
- *or there are* x, y *such that* $x < y$, $f(A(0)A(1)\ldots A(x)) = a_{x+1}a_{x+2}\ldots a_y$ *and* $A(z) = a_z$ *for* $z = x+1, x+2, \ldots, y$.

So, either "A *almost always avoids* f*" or "*A *meets* f*."*

General generic sets have to be distinguished from the weaker variant of generic sets as introduced by Ambos-Spies, Fleischhack and Huwig [2], which either meet or avoid every quasi-polynomial time extension functional which predicts only one bit whenever defined. Balcázar and Mayordomo [7] observed that these sets are not infinitely-often autoreducible.

FACT 2.2 (see [7]). *There are sets which are not infinitely-often autoreducible. In particular, general generic sets have this property.*

On the other hand, Ambos-Spies, Neis and Terwijn [5] showed that the notions

of generic sets and resource-bounded randomness are compatible. As random sets are infinitely-often autoreducible (even infinitely-often truth-table autoreducible) [12, 13], there are some generic sets which are infinitely-often autoreducible, and Fact 2.2 really needs the stronger version of general generic sets.

General generic sets cannot be in EXP due to the quasi-polynomial time bound. The following result shows that sets which are not infinitely-often autoreducible can be found in EXP. Note that many-one EXP-complete sets are (everywhere) autoreducible since they are many-one equivalent to their complement. Buhrman et al. [11] showed that every Turing EXP-complete set is autoreducible.

THEOREM 2.3. *There is a set in EXP which is not infinitely-often autoreducible with respect to Turing reducibility.*

Proof. Let $M_0, M_1, \ldots$ be an enumeration of all polynomial time autoreductions such that each M_e needs at most time $x + e$ at input x and queries the set A only at places $y \leq 2^x$ with $y \neq x$. Note that x is superpolynomial in $\log(x)$ and therefore, all polynomial time autoreductions are covered. The set A is constructed by a priority construction and satisfies at the end for every e one of the following two possibilities:

- There is a number x such that $M_e^A(x)$ outputs $b \in \{0, 1\}$ with $b \neq A(x)$.
- For almost every x, $M_e^A(x)$ is undefined.

The eth requirement is then the first of these two conditions. The construction will be such that it either satisfies the eth requirement explicitly or enforces the second condition implicitly.

The construction uses approximations A_x of A where $A_x(y) = A(y)$ for all x, y with $y < x$. So one has to simulate the construction only $x + 1$ stages to know the value of A at x. Together with a proof that every stage needs time exponential in the length of x, it follows that $A \in \text{EXP}$.

Construction of A. Let A_x and $r_{e,x}$ be the values of A and r_e before stage x, in particular, $A_0 = \emptyset$ and $r_{e,0} = 0$ for all x.

In stage x one searches the least e such that there is a finite set D satisfying the following requirements, where D is the set of the positions for which it is intended that A_{x+1} and A_x will differ:

Bound on e: $e \leq \log^*(x)$.

Respecting restraints: $r_{e',x} < x$ for all $e' \leq e$.

Requirement e not yet done: There is no number $x' < x$ such that $M_e^{A_x}(x')$ queries A_x only at places below x and computes a wrong prediction for $A(x')$.

Requirement e needs attention: $M_e^{A_x \Delta D}(x)$ computes a prediction different from $(A_x \Delta D)(x)$, where $A_x \Delta D$ denotes the symmetric difference of A_x and D.

Size-constraint on D met: $D \subseteq \{x, x + 1, x + 2, \ldots, 2^x\}$ and D has at most $2^{-e-2} \cdot \log^*(x) - 2e$ elements.

If the above procedure finds an e (which is the minimal one possible) and D is the corresponding set mentioned above, then

$$\begin{aligned} A_{x+1} &= A_x \Delta D; \\ r_{e,x+1} &= 2^x + 2; \\ r_{e',x+1} &= r_{e',x} \quad \text{for all } e' \neq e; \end{aligned}$$

else nothing changes; that is, $A_{x+1} = A_x$ and $r_{e',x+1} = r_{e',x}$ for all e'.

Verification. One first notes that in stage x the search considers only $\log^*(x)$ values of e and for each of these, the search runs over computations which make at most $x + e$ queries where at most $2^{-e-2} \cdot \log^*(x) - 2e$ of the answers can differ from the current values of A_x as these queries hit elements in D. Thus, for each e, there

are $O(x^{\log^*(x)})$ possible computation paths. As $\log^*(x) \leq \log(x+2)$, the running time of step x of the algorithm is quasi-polynomial in x, that is, exponential in $\log(x)$. As $A(x) = A_{x+1}(x)$, it follows that one can compute $A(x)$ by running the algorithm for the stages $0, 1, \ldots, x$ and thus the overall running time is quasi-polynomial in x; that is, $A \in \mathrm{EXP}$.

Note that for sufficiently large x the bound $2^{-e-2} \cdot \log^*(x) - 2e$ becomes positive as e is a constant and $\log^*(x)$ is unbounded. Therefore, cardinality requirements do not hinder Requirement e from being satisfied for sufficiently large x.

By usual priority arguments, one can show that every e is found only finitely often by the search algorithm and that $r_{e,x}$ converges from below to a final value $r_{e,\infty}$. More precisely, every $r_{e,x}$ changes only if the parameter e is selected in the search at stage x. One can show by induction that this happens only finitely often for all e. Assume that all $r_{e',\infty}$ with $e' < e$ exist and have maximum $\tilde{r}_e$. If e is not selected in the search at any $x > \tilde{r}_e$, then $r_{e,x}$ changes only finitely often and converges to some value $r_{e,\infty}$. If e is selected at some $x > \tilde{r}_e$, then the following happens. A_{x+1} is updated such that the autoreduction $M_e^{A_{x+1}}(x)$ returns a wrong value and queries A_{x+1} only at places smaller than $2^x + 1$. Furthermore, as at no $x' > x$ does the search select an $e' < e$, the restraint $r_{e,x+1}$ will be respected and the diagonalization of the autoreduction $M_e^{A_{x+1}}$ will not be undone by changing A at queried places; that is, $A_x(y) = A(y)$ for all values y queried by $M_e^{A_{x+1}}(x)$. Thus Requirement e will be counted as "done" at all stages $x' > x$ and therefore e will not be selected again by the search condition at those stages. Thus $r_{e,\infty} = 2^x + 2$.

Now it is shown that A is not infinitely-often autoreducible. Consider any autoreduction M_e which does not make any false prediction for A but might be undefined for some inputs. Let x be so large that $2^{-e-2} \cdot \log^*(x) - 2e > 2$ and $x > r_{e',\infty}$ for all $e' \leq e$. So the search does not return any $e' \leq e$ as otherwise the corresponding restraint would be increased again. It follows that $M_e^B(x)$ does not predict any value for input x on any set B of the form $A_x \Delta D$ with $|D| \leq 2^{-e-2} \cdot \log^*(x) - 2e - 1$ and $D \subseteq \{x, x+1, \ldots, 2^x\}$ (note that the search in the algorithm actually covers sets D with up to $2^{-e-2} \cdot \log^*(x) - 2e$ elements, but this one additional element might be needed to make the prediction be different from $(A_x \Delta D)(x)$). On the other hand, it might happen that up to $\log^*(x)$ requirements $e' > e$ act between stages x and 2^x. Due to the update rule of the restraints, each of them acts only once between these stages. Furthermore, each e' changes A at up to $2^{-e'-2} \cdot \log^*(2^x) - 2e' \leq 2^{-e'-2} \cdot \log^*(x) - 2e - 1$ positions between x and 2^x. The symmetric difference of A and A_x contains below x no element and between x and 2^x at most $2^{-e-2} \cdot \log^*(x) - 2e - 1$ elements. Thus, $M_e^A(x)$ also does not predict any value for $A(x)$ and M_e^A is undefined for almost all inputs. □

3. On truth-table autoreducibility. The topic of this section is the interrelation of autoreducibility notions with respect to truth-table reducibility, Turing reducibility and the variants of these reducibilities where the number of queries is bounded. It is shown that only the trivial implications hold and that all differences between the notions can be witnessed by sets in EXP. Theorem 3.2 in particular shows that there is a set in EXP which is $\mathrm{tt}(n+1)$-autoreducible but not infinitely-often $\mathrm{Turing}(n)$-autoreducible, where n is the logarithm (= length) of the input, $\mathrm{tt}(n+1)$ means that the truth-table reduction is permitted to make up to $n+1$ queries and $\mathrm{Turing}(n)$ means that the Turing reduction is permitted to make up to n queries. This implies that this set is tt-autoreducible but not btt-autoreducible.

The proof of the following theorem uses Kolmogorov complexity arguments to

construct a starting set A_0 which is Turing but not tt-complete for EXP. Watanabe [25] first constructed such a set; his ideas are also based on Kolmogorov complexity arguments.

THEOREM 3.1. *There is a set A in EXP which is Turing autoreducible but not infinitely-often tt-autoreducible.*

Proof. The construction is a modification of the one from Theorem 2.3 with the following two differences:

- The set A_0 is not empty but a set which is Turing complete (but not tt-complete) for EXP.
- Furthermore, $M_0, M_1, \ldots$ is a list of all polynomial time truth-table autoreductions which satisfy the same side conditions as in Theorem 2.3; that is, for any oracle X, $M_e^X(x)$ is computed in time $e+x$ and X is queried only at places $y \leq 2^x$ which are different from x.

The choice of A_0 is sensitive to the success of the construction because it must be guaranteed that one can compute all elements queried by M_e. This is done by placing the more difficult elements at positions which can be accessed by an adapted search but not by a nonadaptive truth-table reduction.

First take a sequence $b_0 b_1 \ldots$ of bits which is random for computations using computation time 2^{n^3} but can be computed in time 2^{n^6} and then let B be the set of all numbers of the form $2^n + \sum_{m<n} 2^m \cdot b_m$. B is in EXP. Furthermore, let C be a set which is many-one EXP-complete and contains 0. The set A_0 is then given by

$$A_0 = \{\langle b, c, d\rangle : b \in B, c \in C, c \leq b, d \in \mathbb{N}\}.$$

Now one uses that the eth truth-table autoreduction M_e queries at most x places and thus any query of it has at most Kolmogorov complexity $\log(e) + \log(x) + k$ for a constant k where the Kolmogorov complexity is measured with respect to time bound x^3. It follows that every $\langle b, c, d\rangle$ queried satisfies that b also has this bound (plus perhaps a constant) and that, whenever $\langle b, c, d\rangle$ is in A_0, then $b = 2^n + \sum_{m<n} 2^m \cdot b_m$ for some $n \leq 3(\log(x) + \log(e) + k')$, and $c \leq b$ where k' is a suitable constant independent of e, x. Thus the algorithm to compute all the values of A_0 at places queried by M_e at x with $e \leq \log^*(x)$ is exponential in x, and this gives that the set A is also in EXP.

The construction gives that A is not infinitely-often tt-autoreducible in the same way as the construction in Theorem 2.3 gives that the set constructed there is not infinitely-often Turing autoreducible. It remains to show that in this theorem the set A is Turing autoreducible.

Note that for fixed b, c and the two-thirds majority of the numbers $y = \langle b, c, d\rangle$ with $0 \leq d \leq 3\log^*(b+c)+6$ satisfies that $A(y) = A_0(y)$. Thus one can reduce every query whether $\langle b, c, 0\rangle \in A_0$ to polynomially many queries to A (omitting the query to x if it is among these queries) and thus it is sufficient to give in the construction below the queries to A_0.

- For input x, compute the c such that $A(x) = C(c)$. This can be done without querying an oracle as C is many-one EXP-complete.
- For every $b \in B$, there is an n such that $b = 2^n + \sum_{m<n} 2^m \cdot b_m$. Then one of the following numbers is the next element of B: $b + 2^n, b + 2^{n+1}$—the first one in case $b_n = 0$, the second one in case $b_n = 1$. Thus one can find for each member of B the next member of B by two queries to B and starting with 1, which is the minimum of B, one can find a $b \geq c$ such that $b \in B$ with $2\log(c) + 2$ queries to B. Since $b \in B$ iff $\langle b, 0, 0\rangle \in A_0$, this computation can also be done with the same number of queries to A_0.

- Having this $b \geq c$ such that $b \in B$, one can determine $A(x)$ with one query to A_0 as $x \in A$ iff $\langle b, c, 0\rangle \in A_0$.

The so constructed algorithm is a Turing autoreduction as the size $\log(c)$ of c is polynomially bounded in the size $\log(x)$ of x. Furthermore, as $3\log^*(b+c)+6 \leq 3\log(b+c)+12$, the number of queries to A is only by a factor polynomial in x greater than the number of queries to A_0 and thus the whole algorithm queries A only polynomially often. One can easily verify that the running time of the algorithm is also polynomial. □

THEOREM 3.2. *For every polynomial time computable and nondecreasing function $g \in Poly(n)$, there is a set A in EXP such that A is tt$(g(n)+1)$-autoreducible but not infinitely-often Turing$(g(n))$-autoreducible.*

Proof. Let $M_0, M_1, \ldots$ be a list of all polynomial time Turing$(g(n))$-autoreductions such that every value $M_e(x)$ can be computed in time $e+x$ and M_e queries only places below 2^x which are different from x. Furthermore, one can produce a polynomial time computable partition Π of almost all natural numbers with the following properties:

- There is an integer m such that $x \in I$ for some $I \in \Pi$ iff $x \geq 2^m$; that is, $\cup_{I\in\Pi} I = \{2^m, 2^m+1, \ldots\}$.
- For every $I \in \Pi$ there is an integer $n(I)$ such that all numbers $x \in I$ satisfy $n(I) \leq \log(x) < n(I)+1$.
- The cardinality of every $I \in \Pi$ is either $g(n(I))+2$ or $2g(n(I))+3$.

As $g \in Poly(n)$, one can find an m such that $2(g(n)+2)^2 < 2^n$ holds for all $n \geq m$. For each $n \geq m$, one can find a, b such that $2^n = a(g(n)+2)-b$ and $0 \leq b \leq g(n)+1$. Note that $a > 2b$ and thus one can partition the set $\{2^n, 2^n+1, \ldots, 2^{n+1}-1\}$ into b intervals of length $2g(n)+3$ followed by $a-2b$ intervals of length $g(n)+2$ and put these intervals into Π. The number $n(I)$ is for these intervals exactly the n with $I \subseteq \{2^n, 2^n+1, \ldots, 2^{n+1}-1\}$ and the numbers $0, 1, \ldots, 2^m-1$ do not belong to any interval.

Now given $I \in \Pi$, let $L(I)$ denote the $g(n(I))+2$ smallest and $H(I)$ the $g(n(I))+2$ largest elements in I. The overall construction of the set A will be such that for every interval $I \in \Pi$ the cardinalities of $L(I) \cap A$ and $H(I) \cap A$ are both even. Therefore one has the following tt$(g(n)+1)$-autoreduction for A.

- If $x < 2^m$, then output "$x \notin A$" and halt.
- Otherwise find the $I \in \Pi$ with $x \in I$.
- If $x \in L(I)$, then query the $g(n)+1$ elements in $L(I)-\{x\}$ and let a be the cardinality of $A \cap (L(I)-\{x\})$; else query the $g(n)+1$ elements in $H(I)-\{x\}$ and let a be the cardinality of $A \cap (H(I)-\{x\})$.
- If a is odd, then output "$x \in A$"; else output "$x \notin A$."

This reducibility is definitely a tt$(g(n)+1)$-autoreduction, can be done in time polynomial in $\log(x)$ and is correct since the cardinalities of $L(I) \cap A$ and $H(I) \cap A$ are even.

It remains to show that one can choose in exponential time the set A such that the resulting set is not infinitely-often Turing$(g(n))$-autoreducible. The construction is done in stages.

Construction. If $x < 2^m$, then let $A_{x+1} = \emptyset$ and $r_{e,x} = 0$ for all e. If $x \geq 2^m$, then determine the interval I with $x \in I$. Search for the least e such that there is a set D satisfying the following conditions:

Bound on e: $e \leq \log(x+1)$.

Respecting restraints: $r_{e',x} < \min(I)$ for all $e' \leq e$.

Requirement e not yet done: There is no $x' < x$ such that $M_e^{A_x}(x')$ is defined,

queries only places below $\min(I)$ and outputs something different from $A_x(x')$.

Requirement e needs attention: $M_e^{A_x \Delta D}(x)$ is defined and differs from $(A_x \Delta D)(x)$.

D not too large: D intersects only intervals J which contain either x or a $y > x$ queried by the $M_e^{A_x \Delta D}(x)$. Furthermore, $\min(I) \leq \min(D \cup \{x\})$, $\max(D \cup \{x\}) \leq 2^x$ and the sets $L(J) \cap D$ and $H(J) \cap D$ have an even number of elements for every interval $J \in \Pi$.

If the above procedure finds an e (which is the minimal one possible) and D is the corresponding set mentioned above, then

$$\begin{aligned} A_{x+1} &= A_x \Delta D; \\ r_{e,x+1} &= 2^x + 2; \\ r_{e',x+1} &= r_{e',x} \quad \text{for all } e' \neq e; \end{aligned}$$

else nothing changes; that is, $A_{x+1} = A_x$ and $r_{e',x+1} = r_{e',x}$ for all e'.

Verification. The set A is in EXP as the search in every stage goes through only exponentially many possibilities ($e \leq \log(x+1)$ and at most $g(\log(x)) + 2$ queries by M_e where g is polynomially bounded). Furthermore, every set D has on every interval of the form $L(I)$ and $H(I)$ an even number of elements so that the resulting set A has the same property: $A \cap L(I)$ and $A \cap H(I)$ have an even number of elements.

It follows from standard priority arguments that every e is found only finitely often by the above search conditions and that the restraints $r_{e,x}$ eventually all take a final value $r_{e,\infty}$.

Now assume that M_e^A never outputs an incorrect value. Consider any x in an interval $I \in \Pi$ such that $\min(I) > \max(\{r_{0,\infty}, r_{1,\infty}, \ldots, r_{e,\infty}\})$. Assume by way of contradiction that $M_e^A(x)$ is defined. Let E be the union of all intersections $J \cap (A_x \Delta A)$ where J contains either x or a $y > x$ which is queried by $M_e^A(x)$. Note that for all intervals J, $L(J) \cap E$ and $H(J) \cap E$ have an even number of elements. Furthermore, $A_x \Delta E$ and A coincide on all relevant elements; thus $M_e^{A_x \Delta E}(x) = A(x)$.

Let l and h be elements of I different from x and not queried by $M_e^{A_x \Delta E}(x)$ such that $l \in L(I)$, $h \in H(I)$, and both sets $\{l, h, x\} \cap L(I)$ and $\{l, h, x\} \cap H(I)$ have even cardinality; note that $L(I) = H(I) = I$ and $l = h$ in the case that I has $g(n) + 2$ elements. It follows from easy cardinality arguments that l, h exist. Now let $D = E \Delta \{l, h, x\}$. The autoreduction M_e behaves at x for the sets $A_x \Delta E$ and $A_x \Delta D$ exactly in the same way, but the output is wrong in the case that one considers $A_x \Delta D$. So, this e witnessed by this D or some $e' < e$ would qualify for the search in stage x, which contradicts the restraints $r_{0,\infty}, r_{1,\infty}, \ldots, r_{e,\infty}$ being below x in the limit. From this contradiction it follows that $M_e^A(x)$ is undefined for almost all x.

So, A is in EXP and A is $\mathrm{tt}(g(n) + 1)$-autoreducible but not infinitely-often $\mathrm{Turing}(g(n))$-autoreducible. □

An autoreduction is a bounded truth-table reduction iff there is a constant k such that the reduction makes for every input at most k queries. If a set A is $\mathrm{tt}(n+1)$-autoreducible but not infinitely-often $\mathrm{Turing}(n)$-autoreducible, then A is clearly tt-autoreducible but not infinitely-often btt-autoreducible. This gives the following corollary.

COROLLARY 3.3. *There is a set in EXP that is truth-table autoreducible but not infinitely-often bounded truth-table autoreducible.*

4. Notions of approximability. A set is called (a, b)-recursive iff there is a function f such that for all distinct $x_1, x_2, \ldots, x_b$ the function f computes a tuple $(y_1, y_2, \ldots, y_b)$ of bits such that at least a of the equations $A(x_k) = y_k$ are true. If

there are a, b such that $1 \leq a \leq b$ and if there is a polynomial time computable function f such that A is (a, b)-recursive via f, then A is called approximable [9] and if, in addition, $2a > b$, then A is called easily approximable.

In the recursion theoretic setting, every set which is $(1, b)$-recursive for some b is also autoreducible [16]. This does not carry over to complexity theory: Returning to the world of polynomial time computations, supersparse sets are $(1, 2)$-recursive but not Turing autoreducible. Supersparse sets are related to k-cheatable sets [8]. Nevertheless, approximable sets are still infinitely-often autoreducible.

PROPOSITION 4.1. *Every approximable set is infinitely-often btt-autoreducible.*

Proof. Assume that A is approximable via f and let l be the largest constant such that for infinitely many inputs of the form $x+1, x+2, \ldots, x+k$ at least l of the bits $y_1, y_2, \ldots, y_k$ are wrong. Let u be so large that it never happens for an $x > u$ that more than l of these bits are wrong. Now A is btt-autoreducible as follows.

Algorithm. Ignore inputs $x < u + k$. For $x \geq u + k$ query A at all numbers x' with $x - k < x' < x + k$ and $x' \neq x$. If there is an x' with $x - k \leq x' < x$ such that the answers $y_1, y_2, \ldots, y_k$ computed by f from input $x'+1, x'+2, \ldots, x'+k$ satisfy that $y_{k'} \neq A(x' + k')$ for l numbers $k' \in \{1, 2, \ldots, k\} - \{x - x'\}$, then predict $y_{x-x'}$ for $A(x)$; else do not make any prediction.

Verification. The correctness follows from the fact that there are at most l differences between $A(x' + k')$ and $y_{k'}$ and that these errors have been localized at places different from x. That this reduction works for infinitely many x is a consequence of the choice of l. □

Ogihara [21] also considered sets which are $(1, b(n))$-recursive, where b is a function in $Poly(n)$ and the parameter n is $\log(\max\{1, x_1, x_2, \ldots, x_b\})$, where $x_1, x_2, \ldots, x_b$ is the input to the function to compute the approximation. This generalized notion no longer enforces that A is infinitely-often autoreducible.

EXAMPLE 4.2. *The set from Theorem* 2.3 *satisfies Ogihara's generalized approximability condition but is not infinitely-often Turing autoreducible.*

Proof. It is sufficient to show that the set A constructed in Theorem 2.3 is $(1, b(n))$-recursive with $b(n) = 6\log^*(n) + 3$: On input $x_1, x_2, \ldots, x_{b(n)}$, let

$$y_k = \begin{cases} 0 & \text{if } x_k > \log\log(n), \\ A(x_k) & \text{if } x_k \leq \log\log(n), \end{cases}$$

and predict $(y_1, y_2, \ldots, y_{b(n)})$. If $x_k \leq \log\log(n)$, then $A(x_k) = y_k$ and this prediction is correct. Otherwise one knows that between $\log\log(n)$ and n the set A has at most $6\log^*(n) + 2$ elements. It follows that at least one of the predicted 0's is correct. Furthermore, the computations involved can all be done in polynomial time since $A(x_k)$ can be computed in time $2^{Poly(\log\log(n))}$ whenever $x_k \leq \log\log(n)$. □

An easily approximable set A has the property that every set B Turing reducible to A is also tt-reducible to A. This property is called *T-easy*. The next theorem shows that every T-easy set is either infinitely-often autoreducible or satisfies Ogihara's generalized approximability notion with $a = 1$ and $b = \log^2(n)$.

THEOREM 4.3. *Every set A satisfies at least one of the following properties.*

(a) *Not T-easy: There is a set B which is polynomial time Turing reducible but not polynomial time truth-table reducible to A.*
(b) *Infinitely-often truth-table autoreducible: For infinitely many z, $A(z)$ can be computed by queries to places different from z.*
(c) *Generalized approximable: A is $(1, 1 + \log^2(n))$-recursive via some function computable in polynomial time.*

Proof. Let A be any set. Furthermore, define a tuple-function which at input $x_1, x_2, \ldots, x_m$ computes the binary representations $u_0, u_1, \ldots, u_m$ of these numbers and outputs a number, denoted as $\langle x_1, x_2, \ldots, x_m \rangle$, which has the ternary representation $u_1 2 u_2 2 \ldots 2 u_m 2$. Note that $\langle x_1, x_2, \ldots, x_m \rangle > x_k$ for $k = 1, 2, \ldots, m$. Furthermore, $n = \log(\max\{1, x_1, x_2, \ldots, x_m\})$ satisfies $\langle x_1, x_2, \ldots, x_m \rangle < 3^{(n+3)m}$. Let $y(x_1, x_2, \ldots, x_m : A)$ be the number which is represented by the binary string $A(x_1)A(x_2)\ldots A(x_n)$. Depending on what type of reducibilities exists from

$$B = \{\langle x_1, x_2, \ldots, x_m \rangle : \langle x_1, x_2, \ldots, x_m, y(x_1, x_2, \ldots, x_m : A)\rangle \in A\}$$

to A, one of three properties holds:

(a) B is not truth-table reducible to A. Then A *is not T-easy*. This is verified by showing that B is Turing reducible to A as follows: One first queries whether $x_1, x_2, \ldots, x_m \in A$, then computes $y(x_1, x_2, \ldots, x_m : A)$ and at last queries whether $\langle x_1, x_2, \ldots, x_m, y(x_1, x_2, \ldots, x_m : A)\rangle$ is in A.

(b) B is truth-table reducible to A by a reduction which for infinitely many tuples $\langle x_1, x_2, \ldots, x_m \rangle$ computes $B(\langle x_1, x_2, \ldots, x_m \rangle)$ without querying A whether $\langle x_1, x_2, \ldots, x_m, y(x_1, x_2, \ldots, x_m : A)\rangle$ is in A. Then A *is infinitely-often truth-table autoreducible* by f defined as follows: Let $F(x)$ denote the set of queries which the truth-table reduction from B to A makes at input x. On input z, f checks whether there are a set E and numbers $m, x_1, x_2, \ldots, x_m$ such that

- $z = \langle x_1, x_2, \ldots, x_m, y(x_1, x_2, \ldots, x_m : E)\rangle$ and
- $z \notin F(E(x_1)E(x_2)\ldots E(x_m)) \cup \{x_1, x_2, \ldots, x_m\}$.

If so, then f queries A at the members of the set $F(\langle x_1, x_2, \ldots, x_m \rangle) \cup \{x_1, x_2, \ldots, x_m\}$ and outputs the result of the truth-table reduction from B to A computed for the value $B(\langle x_1, x_2, \ldots, x_m \rangle)$ in the case that $A(x_1) = E(x_1), \ldots, A(x_m) = E(x_m)$. If z is not of the above form or if the supposed values of E do not coincide with A or $z \in F(\langle x_1, x_2, \ldots, x_m \rangle)$, then $f(z)$ is undefined. Note that whenever $f(z)$ is defined, the value computed coincides with $A(z)$ and f does not query $A(z)$ as $x_1, x_2, \ldots, x_m < \langle x_1, x_2, \ldots, x_m, y(x_1, x_2, \ldots, x_m : E)\rangle = z$ and $z \notin F(E(x_1)E(x_2)\ldots E(x_m))$.

(c) The two previous cases do not hold. Then A *is* $(1, b(n))$*-recursive with* $b(n) = 1 + \lfloor \log^2(n) \rfloor$. As (a) does not hold, there is a tt-reduction from B to A. Let $F(x)$ be the set of queries made at input x. There is a constant c such that the cardinality of $F(x)$ is at most $\log^c(x)$ for almost all x. If n is sufficiently large and $x = \langle x_1, x_2, \ldots, x_m \rangle$ with $x_1 < x_2 < \ldots < x_m$ and $n = \lfloor \log(x_m) \rfloor$ and $m = b(n)$, then the following facts hold: $y(x_1, x_2, \ldots, x_m : A) \in F(\langle x_1, x_2, \ldots, x_m \rangle)$ as (b) does not hold, $n > m$, and $(\log(3) \cdot m \cdot (n+3))^c < n^{4c} < n^{\log(n)}$. As $2^m > n^{\log(n)}$, it follows that there is a number y with binary representation $d_1 d_2 \ldots d_m$ such that $\langle x_1, x_2, \ldots, x_m, y \rangle \notin F(\langle x_1, x_2, \ldots, x_m \rangle)$. Thus at least one of the conditions $A(x_1) \neq d_1, A(x_2) \neq d_2, \ldots, A(x_m) \neq d_m$ holds and A is $(1, b(n))$-recursive by outputting the vector $(1 - d(x_1), 1 - d(x_2), \ldots, 1 - d(x_m))$. The remaining case is where n is too small. But there are only finitely many $x_1, x_2, \ldots, x_m$ where this can happen and one can output the characteristic vector $(A(x_1), A(x_2), \ldots, A(x_m))$ in these finitely many cases. One can easily verify that the proposed algorithm runs in polynomial time and thus A is $(1, b(n))$-recursive. □

5. Hausdorff-dimension. Hausdorff [14] introduced a generalized notion of dimension for metric spaces; it also enables one to measure the size of fractal objects. Lutz [19] adapted the notion for complexity theory in order to measure the size of subclasses of the natural numbers. The following definition—one among several equivalent ones—defines the Hausdorff-dimension in terms of the growth rate of the capital

accumulated by a gambler who bets inductively on the bits of the characteristic functions of any set in the given class. Such growth rate functionals are called martingales.

DEFINITION 5.1. *A quasi-polynomial time computable functional f is called a* martingale, *iff it maps binary strings to positive numbers and the empty string to* 1 *and satisfies*

$$2f(a_0a_1\dots a_x) = f(a_0a_1\dots a_x0) + f(a_0a_1\dots a_x1)$$

for all binary strings $a_0a_1\dots a_x$. The Hausdorff-dimension *(with respect to EXP) of a class S is the infimum of all s such that there is a quasi-polynomial time computable martingale f succeeding on every set in $A \in S$ with growth rate 2^{1-s}; that is, f satisfies*

$$(\textit{for all } A \in S)\,(\exists^\infty x)\,[f(A(0)A(1)\dots A(x)) \geq 2^{(1-s)x}].$$

REMARK 5.2. *The class of all sets as well as the class EXP has Hausdorff-dimension* 1. *The upper bound is witnessed by the martingale mapping all strings to* 1, *and the lower bound is obtained by constructing for any given quasi-polynomial martingale f the set A in EXP which satisfies for $x = 0, 1, \dots$ that*

$$f(A(0)A(1)\dots A(x)A(x+1)) \leq f(A(0)A(1)\dots A(x));$$

that is, $A(x+1)$ takes just the value adversary to $A(x)$. Note that in EXP one cannot have one set doing this for all martingales due to the fact that every single set in EXP is predictable by a suitable quasi-polynomial time computable martingale. There are also sets A that are random for all quasi-polynomial time computable martingales and thus satisfy that the Hausdorff-dimension of $\{A\}$ is 1, *but no such A is in EXP.*

Ambos-Spies et al. [4, Corollary 16] considered the Hausdorff-dimension adapted to the class $E = \{A : (\exists c)\,(\exists M)\,(\text{for all } x)\,[M \text{ computes } A(x) \text{ in time } c + x^c]\}$. They showed that the class of many-one autoreducible sets in E has Hausdorff-dimension 1. This fact directly carries over to EXP.

FACT 5.3. *The class $\{A \text{ in EXP: } (\textit{for all } x)\,[A(x) = A(x^2)]\}$ consists only of many-one autoreducible sets and has Hausdorff-dimension* 1. *In particular, the classes of the many-one autoreducible, truth-table autoreducible and Turing autoreducible sets in EXP have Hausdorff-dimension* 1.

An interesting obvious question is to determine the Hausdorff-dimension of the class S of those sets in EXP which are not infinitely-often r-autoreducible. The next theorem shows that the Hausdorff-dimension is already 1 for the case of the polynomial time Turing reducibility; it then also follows for the other polynomial time reducibilities r. As every set which is x^3-random is already infinitely-often autoreducible [12, 13], the class S is a natural example of a class which has Hausdorff-dimension 1 and measure 0 with respect to quasi-polynomial time martingales.

THEOREM 5.4. *The class of all sets in EXP that are not infinitely-often autoreducible has Hausdorff-dimension* 1.

Proof. The basic idea of the proof is to use the same construction as in Theorem 2.3. But one has to make the following changes. One has to construct for every quasi-polynomial time martingale f such a set A. Furthermore, for given f, A_0 has to be chosen such that f does not have a fast growth rate either on A_0 or on any supersparse variant of A_0. In particular the supersparse variant A of A_0 will satisfy that there is no $s < 1$ with $(\exists^\infty x)\,[f(A(0)A(1)\dots A(x)) \geq 2^{(1-s)\cdot x}]$. Furthermore, A_0

has to be chosen such that not only A_0 itself but also the A constructed from A_0 is in EXP.

To meet these constraints, one chooses the polynomial time Turing reductions $M_0, M_1, \ldots$ in a bit more restrictive way than in Theorem 2.3; namely the M_e have to satisfy the following properties:

- $M_e^X(x)$ queries any given oracle X at up to $(\log(x))^{\log\log(x)}$ places; these places are below 2^x and different from x.
- $M_e^X(x)$ needs at most running time $e + (\log(x))^{\log\log(x)}$.

Note that the bound $(\log(x))^{\log\log(x)}$ is superpolynomial and thus one has, starting with a given reduction, only to modify it on finitely many inputs, where one can put the correct result into a table so that no query is necessary at all. Thus whenever A is infinitely-often autoreducible, then A is infinitely-often autoreducible by some M_e.

As f might behave very differently on A compared to on A_0, one has to define A_0 such that the growth rate of f does not only on A_0 but also on all supersparse variants of A_0 fail to reach 2^{1-s} for any $s < 1$.

So one considers the following functional f': Let r_D be the product of all $1 + 1/(1 + d \cdot d)$, where $d \in D$, $r_\emptyset = 1$. Now $f'(B(0)B(1)\ldots B(x))$ is the sum over all terms $r_D \cdot f(C(0)C(1)\ldots C(x))$, where $C \subseteq \{0, 1, \ldots, x\}$, $D = \{y \leq x : B(y) \neq C(y)\}$ and D is a supersparse set. The functional f' is no longer a martingale, but it satisfies the following condition:

$$f'(a_0a_1\ldots a_x0) + f'(a_0a_1\ldots a_x1) \leq 2 \cdot \tfrac{2+(x+1)\cdot(x+1)}{1+(x+1)\cdot(x+1)} \cdot f'(a_0a_1\ldots a_x).$$

Now one defines the set A_0 inductively as follows. $A_0(0) = 0$. $A_0(x+1) = 0$ iff one of the following two conditions holds and $A_0(x+1) = 1$ otherwise:

(a) there are a $y < x^{1/5}$, an $e \leq \log^*(x)$ and a supersparse set $D \subseteq \{0, 1, \ldots, 2^x\}$ such that $M_e^{(A_0 \cap \{0,1,\ldots,x\})\Delta D}(y)$ queries x.

(b) $f'(a_0a_1\ldots a_x0) \leq \frac{2+(x+1)\cdot(x+1)}{1+(x+1)\cdot(x+1)} \cdot f'(a_0a_1\ldots a_x)$.

Note that the search over the queried elements can be done in exponential time: Every M_e queries at most $(\log(x))^{\log\log(x)}$ elements, and at most $(\log^*(x)+1)^2$ of them can differ from the corresponding value of $A_0 \cap \{0, 1, \ldots, x\}$ as one searches over answers given by $A_0\Delta D$ and not just A_0. Thus one has to consider $((\log(x))^{\log\log(x)})^{(\log^*(x)+2)^2}$ computation paths for every $y \leq x^{1/5}$ and $e \leq \log^*(x)$; each path might need up to $\log^*(x) \cdot (\log(x))^{\log\log(x)}$ steps. As except $x^{1/5}$ all these factors grow slower than every function x^q, $q > 0$, there is a constant for c such that, for every x, the number of steps to be simulated is at most $x^{1/4} + c$. As every 0 caused by condition (a) in the construction of A_0 is due to a query of one of these simulated paths, there are at most $x^{1/4} + c$ places $y \leq x$ where $A_0(y+1) = 0$ due to condition (a).

If $A_0(x+1) = 0$ is caused by condition (a), then the value of f' can go up by the factor $2 \cdot \frac{2+(x+1)\cdot(x+1)}{1+(x+1)\cdot(x+1)}$; else $A(x+1)$ is chosen such that the value of f' goes up by only half of that, that is, by the factor $\frac{2+(x+1)\cdot(x+1)}{1+(x+1)\cdot(x+1)}$. The product over $f'(A(0))$ and all numbers $\frac{2+(x+1)\cdot(x+1)}{1+(x+1)\cdot(x+1)}$ is bounded by a constant d. So it follows that, for every x, $f'(A_0(x)A_1(x)\ldots A_x(x)) \leq 2^{x^{1/4}+c} \cdot d$. The construction of Theorem 2.3 gives that A is a supersparse variant of A_0. Let $B = A\Delta A_0$. For almost all x, $D = B \cap \{0, 1, \ldots, x\}$ has at most $(\log^*(x))^2$ elements, $1/r_D \leq x^{|D|} \leq 2^{x^{1/4}}$, and $f(A(0)A(1)\ldots A(x)) \leq f'(A(0)A(1)\ldots A(x))/r_D \leq 2^{x^{1/4}+c+1} \cdot d$. It follows that there is no $s < 1$ such that $f(A(0)A(1)\ldots A(x)) \geq 2^{(1-s)x}$ for infinitely many x.

It remains to show that A is in EXP. Note that it follows from the construction of Theorem 2.3 that at every stage x and every D considered in the search condition, the set $A_x \Delta D$ is a supersparse variant of A_0. It follows from the definition of A_0 that whenever $M_e^{A_x \Delta D}(x)$ queries an element $y > x^5$, then $A_0(y) = 0$. As A_0 is in EXP, there is a polynomial p such that one can compute in time $2^{p(\log(x))}$ the value of $A_0(y)$ for any given $y \leq x^5$. It follows that one can compute $A_0(y)$ in all queried places in time exponential in $\log(x)$. Every stage in the construction changes the approximation of A at fewer than x places, so one can bookkeep these changes and compute A in exponential time. This completes the proof. □

Acknowledgments. The authors would like to thank Klaus Ambos-Spies, Jack H. Lutz and Wolfgang Merkle for helpful discussions; furthermore, Jack H. Lutz proposed that we investigate the Hausdorff-dimension of the class of sets in EXP that are not infinitely-often autoreducible.

REFERENCES

[1] K. Ambos-Spies, *Resource-bounded genericity*, in Computability, Enumerability, Unsolvability, London Math. Soc. Lecture Note Ser. 224, S. B. Cooper et al., eds., Cambridge University Press, Cambridge, UK, 1996, pp. 1–59.

[2] K. Ambos-Spies, H. Fleischhack, and H. Huwig, *Diagonalizations over polynomial time computable sets*, Theoret. Comput. Sci., 51 (1997), pp. 177–204.

[3] K. Ambos-Spies and E. Mayordomo, *Resource-bounded measure and randomness*, in Complexity, Logic, and Recursion Theory, Lecture Notes in Pure and Appl. Math. 187, A. Sorbi, ed., Dekker, NY, 1997, pp. 1–47.

[4] K. Ambos-Spies, W. Merkle, J. Reimann, and F. Stephan, *Hausdorff dimension in exponential time*, in Proceedings of the Sixteenth Annual IEEE Conference on Computational Complexity, IEEE Computer Society, Los Alamitos, CA, 2001, pp. 210–217.

[5] K. Ambos-Spies, H.-C. Neis, and S. A. Terwijn, *Genericity and measure for exponential time*, Theoret. Comput. Sci., 168 (1996), pp. 3–19.

[6] K. Ambos-Spies, S. A. Terwijn, and X. Zheng, *Resource bounded randomness and weakly complete problems*, Theoret. Comput. Sci., 172 (1997), pp. 195–207.

[7] J. L. Balcázar and E. Mayordomo, *A note on genericity and bi-immunity*, in Proceedings of the Tenth Annual Structure in Complexity Theory Conference, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 193–196.

[8] R. Beigel, *Bi-immunity results for cheatable sets*, Theoret. Comput. Sci., 73 (1990), pp. 249–263.

[9] R. Beigel, M. Kummer, and F. Stephan, *Approximable sets*, Inform. and Comput., 120 (1995), pp. 304–314.

[10] W. Blum, *Denksport für Hutträger*, Die Zeit, Hamburg, Germany, 3 May 2001.

[11] H. Buhrman, L. Fortnow, D. van Melkebeek, and L. Torenvliet, *Separating complexity classes using autoreducibility*, SIAM J. Comput., 29 (2000), pp. 1497–1520.

[12] T. Ebert, *Applications of Recursive Operators to Randomness and Complexity*, Ph.D. thesis, University of California at Santa Barbara, Santa Barbara, CA, 1998.

[13] T. Ebert, W. Merkle, and H. Vollmer, *On the autoreducibility of random sequences*, SIAM J. Comput., 32 (2003), pp. 1542–1569.

[14] F. Hausdorff, *Dimension und äusseres Maß*, Math. Ann., 79 (1919), pp. 157–189.

[15] C. G. Jockusch and M. S. Paterson, *Completely autoreducible degrees*, Z. Math. Logik Grundlagen Math., 22 (1976), pp. 571–575.

[16] M. Kummer and F. Stephan, *Recursion theoretic properties of frequency computation and bounded queries*, Inform. and Comput., 120 (1995), pp. 59–77.

[17] J. H. Lutz, *Category and measure in complexity classes*, SIAM J. Comput., 19 (1990), pp. 1100–1131.

[18] J. H. Lutz, *Almost everywhere high nonuniform complexity*, J. Comput. System Sci., 44 (1992), pp. 220–258.

[19] J. H. Lutz, *Dimension in complexity classes*, SIAM J. Comput., 32 (2003), pp. 1236–1259.

[20] P. Odifreddi, *Classical Recursion Theory*, North-Holland, Amsterdam, 1989.

[21] M. Ogihara, *Polynomial-time membership comparable sets*, SIAM J. Comput., 24 (1995), pp. 1068–1081.

[22] S. Robinson, *Why mathematicians now care about their hat color*, The New York Times, New York, 10 April 2001.
[23] C.-P. Schnorr, *Optimal algorithms for self-reducible problems*, in Proceedings of the Third International Colloquium on Automata, Languages, and Programming (ICALP 1976), University of Edinburgh Press, Edinburgh, UK, 1976, pp. 322–337.
[24] B. A. Trakhtenbrot, *On autoreducibility*, Soviet Math. Dokl., 11 (1970), pp. 814–817.
[25] O. Watanabe, *A comparison of polynomial time completeness notions*, Theoret. Comput. Sci., 54 (1987), pp. 249–265.

SIAM J. COMPUT.
Vol. 36, No. 3, pp. 609–634

AN EXTENSION OF THE LOVÁSZ LOCAL LEMMA, AND ITS APPLICATIONS TO INTEGER PROGRAMMING*

ARAVIND SRINIVASAN†

Abstract. The Lovász local lemma due to Erdős and Lovász (*Infinite and Finite Sets*, Colloq. Math. Soc. J. Bolyai 11, 1975, pp. 609–627) is a powerful tool in proving the existence of rare events. We present an extension of this lemma, which works well when the event to be shown to exist is a conjunction of individual events, each of which asserts that a random variable does not deviate much from its mean. As applications, we consider two classes of NP-hard integer programs: minimax and covering integer programs. A key technique, *randomized rounding of linear relaxations*, was developed by Raghavan and Thompson (*Combinatorica*, 7 (1987), pp. 365–374) to derive good approximation algorithms for such problems. We use our extension of the local lemma to prove that randomized rounding produces, with nonzero probability, much better feasible solutions than known before, if the constraint matrices of these integer programs are *column-sparse* (e.g., routing using short paths, problems on hypergraphs with small dimension/degree). This complements certain well-known results from discrepancy theory. We also generalize the *method of pessimistic estimators* due to Raghavan (*J. Comput. System Sci.*, 37 (1988), pp. 130–143), to obtain constructive (algorithmic) versions of our results for covering integer programs.

Key words. Lovász local lemma, column-sparse integer programs, approximation algorithms, randomized rounding, discrepancy

AMS subject classifications. 60C05, 90C05, 90C10

DOI. 10.1137/S0097539703434620

1. Introduction.

The powerful Lovász local lemma (LLL) is often used to show the existence of *rare* combinatorial structures by showing that a random sample from a suitable sample space produces them with positive probability [14]; see Alon and Spencer [4] and Motwani and Raghavan [27] for several such applications. We present an extension of this lemma, and demonstrate applications to rounding fractional solutions for certain families of integer programs. A preliminary version of this work appeared in [35], with a sketch of the proof for minimax integer programs, and proofs omitted for our constructive results for covering integer programs. In this version, we provide all proofs, further generalize the main covering result of [35] to Theorem 5.9, and present applications thereof; in particular, the constructive approach to covering integer programs detailed in section 5.3, requires a fair amount of work.

Let e denote the base of natural logarithms. The symmetric case of the LLL shows that all of a set of "bad" events E_i can be avoided under some conditions.

LEMMA 1.1. (see [14]) *Let $E_1, E_2, \ldots, E_m$ be any events with $\Pr(E_i) \leq p\ \forall i$. If each E_i is mutually independent of all but at most d of the other events E_j and if $ep(d+1) \leq 1$, then $\Pr(\bigwedge_{i=1}^{m} \overline{E_i}) > 0$.*

*Received by the editors September 18, 2003; accepted for publication (in revised form) September 20, 2005; published electronically September 21, 2006. A preliminary version of this work appeared as a paper of the same title in the *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pp. 6–15, 1996. Work done in parts at: (i) the University of Maryland (supported in part by NSF Award CCR-0208005 and NSF ITR Award CNS-0426683), (ii) the National University of Singapore, (iii) DIMACS (supported in part by NSF-STC91-19999 and by support from the N. J. Commission on Science and Technology), (iv) the Institute for Advanced Study, Princeton, NJ (supported in part by grant 93-6-6 of the Alfred P. Sloan Foundation), and (v) while visiting the Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany.

http://www.siam.org/journals/sicomp/36-3/43462.html

†Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742 (srin@cs.umd.edu).

Though the LLL is powerful, one problem is that the "dependency" d is high in some cases, precluding the use of the LLL if p is not small enough. We present a partial solution to this via an extension of the LLL (Theorem 3.1), which shows how to essentially reduce d for a class of events E_i; this works well when each E_i denotes some random variable deviating "much" from its mean. In a nutshell, we show that such events E_i can often be decomposed suitably into subevents; although the subevents may have a large dependency among themselves, we show that it suffices to have a small "bipartite dependency" between the set of events E_i and the set of subevents. This, in combination with some other ideas, leads to the following applications in integer programming.

It is well known that a large number of NP-hard combinatorial optimization problems can be cast as integer linear programming problems (ILPs). Due to their NP-hardness, good approximation algorithms are of much interest for such problems. Recall that a ρ-approximation algorithm for a minimization problem is a polynomial-time algorithm that delivers a solution whose objective function value is at most ρ times optimal; ρ is usually called the *approximation guarantee*, *approximation ratio*, or *performance guarantee* of the algorithm. Algorithmic work in this area typically focuses on achieving the smallest possible ρ in polynomial time. One powerful paradigm here is to start with the linear programming (LP) relaxation of the given ILP wherein the variables are allowed to be *reals* within their integer ranges; once an optimal solution is found for the LP, the main issue is how to *round* it to a good feasible solution for the ILP. Rounding results in this context often have the following strong property: they present an integral solution of value at most $y^* \cdot \rho$, where y^* will throughout denote the optimal solution value of the LP relaxation. Since the optimal solution value OPT of the ILP is easily seen to be lower-bounded by y^*, such rounding algorithms are also ρ-approximation algorithms. Furthermore, they provide an upper bound of ρ on the ratio OPT/y^*, which is usually called the *integrality gap* or *integrality ratio* of the relaxation; the smaller this value, the better the relaxation.

This work presents improved upper bounds on the integrality gap of the natural LP relaxation for two families of ILPs: minimax integer programs (MIPs) and covering integer programs (CIPs). (The precise definitions and results are presented in section 2.) For the latter, we also provide the corresponding polynomial-time rounding algorithms. Our main improvements are in the case where the coefficient matrix of the given ILP is *column-sparse*: i.e., the number of nonzero entries in every column is bounded by a given parameter a. There are classical rounding theorems for such column-sparse problems (e.g., Beck and Fiala [6], Karp, Leighton, Rivest, Thompson, Vazirani, and Vazirani [18]). Our results complement, and are incomparable with, these results. Furthermore, the notion of column-sparsity, which denotes no variable occurring in "too many" constraints, occurs naturally in combinatorial optimization: e.g., routing using "short" paths, and problems on hypergraphs with "small" degree. These issues are discussed further in section 2.

A key technique, *randomized rounding of linear relaxations*, was developed by Raghavan and Thompson [32] to get approximation algorithms for such ILPs. We use Theorem 3.1 to prove that this technique produces, with nonzero probability, much better feasible solutions than known before, if the constraint matrix of the given MIP/CIP is column-sparse. (In the case of MIPs, our algorithm iterates randomized rounding several times with different choices of parameters, in order to achieve our result.) Such results cannot be got via Lemma 1.1, as the dependency d, in the sense of Lemma 1.1, can be as high as $\Theta(m)$ for these problems. Roughly speaking, Theorem 3.1 helps show that if no column in our given ILP has more than a nonzero

entries, then the dependency can essentially be brought down to a polynomial in a; this is the key driver behind our improvements.

Theorem 3.1 works well in combination with an idea that has blossomed in the areas of derandomization and pseudorandomness in the last two decades: (approximately) decomposing a function of several variables into a sum of terms, each of which depends on only a few of these variables. Concretely, suppose Z is a sum of random variables Z_i. Many tools have been developed to upper-bound $\Pr(Z - \mathbf{E}[Z] \geq z)$ and $\Pr(|Z - \mathbf{E}[Z]| \geq z)$ even if the Z_is are only (almost) k-wise independent for some "small" k, rather than completely independent. The idea is to bound the probabilities by considering $\mathbf{E}[(Z - \mathbf{E}[Z])^k]$ or similar expectations, which look at the Z_i, *k or fewer at a time* (via linearity of expectation). The main application of this has been that the Z_i can then be sampled using "few" random bits, yielding a derandomization/pseudorandomness result (e.g., [3, 23, 8, 26, 28, 33]). Our results show that such ideas can in fact be used to show that some structures exist! This is one of our main contributions.

What about polynomial-time algorithms for our existential results? Typical applications of Lemma 1.1 are "nonconstructive" (i.e., do not directly imply (randomized) polynomial-time algorithmic versions), since the positive probability guaranteed by Lemma 1.1 can be exponentially small in the size of the input. However, certain algorithmic versions of the LLL have been developed starting with the seminal work of Beck [5]. These ideas do not seem to apply to our extension of the LLL, and hence our MIP result is nonconstructive. Following the preliminary version of this work [35], two main algorithmic versions related to our work have been obtained: (i) for a subclass of the MIPs [20], and (ii) for a somewhat different notion of approximation than the one we study, for certain families of MIPs [11].

Our main algorithmic contribution is for CIPs and multicriteria versions thereof: we show, by a generalization of the *method of pessimistic estimators* [31], that we can efficiently construct the same structure as is guaranteed by our nonconstructive argument. We view this as interesting for two reasons. First, the generalized pessimistic estimator argument requires a quite delicate analysis, which we expect to be useful in other applications of developing constructive versions of existential arguments. Second, except for some of the algorithmic versions of the LLL developed in [24, 25], most current algorithmic versions minimally require something like "$pd^3 = O(1)$" (see, e.g., [5, 1]); all the LLL needs is that $pd = O(1)$. While this issue does not matter much in many applications, it crucially does, in some others. A good example of this is the existentially-optimal integrality gap for the edge-disjoint paths problem with "short" paths, shown using the LLL in [21]. The above-seen "$pd^3 = O(1)$" requirement of currently-known algorithmic approaches to the LLL leads to algorithms that will violate the edge-disjointness condition when applied in this context: specifically, they may route up to three paths on some edges of the graph; see [9] for a different—random-walk based—approach to low-congestion routing. An algorithmic version of this edge-disjoint paths result of [21] is still lacking. It is a very interesting open question whether there is an algorithmic version of the LLL that can construct the same structures as guaranteed to exist by the LLL. In particular, can one of the most successful derandomization tools, the method of conditional probabilities or its generalization, the pessimistic estimators method, be applied, fixing the underlying random choices of the probabilistic argument one-by-one? This intriguing question is open (and seems difficult) for now; it is elaborated upon further in section 6. As a step in this direction, we are able to show how such approaches can indeed be developed, in the context of CIPs.

Thus, our main contributions are as follows. (a) The LLL extension is of independent interest: it helps in certain settings where the "dependency" among the "bad" events is too high for the LLL to be directly applicable. We expect to see further applications/extensions of such ideas. (b) This work shows that certain classes of column-sparse ILPs have much better solutions than known before; such problems abound in practice (e.g., short paths are often desired/required in routing). (c) Our generalized method of pessimistic estimators should prove fruitful in other contexts also; it is a step toward complete algorithmic versions of the LLL.

A note on reading this paper. In order to get the main ideas of this work across, we summarize the main ideas of each section and some of the subsections, at their beginning. We also have paragraphs marked "Intuition" in some of the sections, in order to spell out the main ideas informally. Furthermore, we have moved simple-but-tedious calculations to the appendix; it may help the reader to read the body of the paper first, and to then read the material in the appendix if necessary. The rest of this paper is organized as follows. Our results are first presented in section 2, along with a discussion of related work. The extended LLL, and some large-deviation methods that will be seen to work well with it, are shown in section 3. Sections 4 and 5 are devoted to our rounding applications. Section 6 concludes, and some of the routine calculations are presented in the appendix.

2. Our results and related work. Let Z_+ denote the set of nonnegative integers; for any $k \in Z_+$, $[k] \doteq \{1, \ldots, k\}$. "Random variable" is abbreviated as "r.v.," and logarithms are to the base 2 unless specified otherwise.

DEFINITION 2.1 (minimax integer programs). *An MIP has the following variables: $\{x_{i,j} : i \in [n], j \in [\ell_i]\}$, for some integers $\{\ell_i\}$, and an extra variable W. Let $N = \sum_{i \in [n]} \ell_i$ and let x denote the N-dimensional vector of the variables $x_{i,j}$ (arranged in any fixed order). An MIP seeks to minimize W, an unconstrained real, subject to*

(i) *equality constraints: $\forall i \in [n]$ $\sum_{j \in [\ell_i]} x_{i,j} = 1$,*

(ii) *a system of linear inequalities $Ax \leq \vec{W}$, where $A \in [0,1]^{m \times N}$ and $\vec{W}$ is the m-dimensional vector with the variable W in each component, and*

(iii) *integrality constraints: $x_{i,j} \in \{0,1\}$ $\forall i, j$.*

We let g denote the maximum column sum in any column of A, and a be the maximum number of nonzero entries in any column of A.

To see what problems MIPs model, note from constraints (i) and (iii) of MIPs that for all i, any feasible solution will make the set $\{x_{i,j} : j \in [\ell_i]\}$ have precisely one 1, with all other elements being 0; MIPs thus model many "choice" scenarios. Consider, e.g., global routing in VLSI gate arrays [32]. Given are an undirected graph $G = (V, E)$, a function $\rho : V \to V$, and $\forall i \in V$, a set P_i of paths in G, each connecting i to $\rho(i)$; we must connect each i with $\rho(i)$ using exactly one path from P_i, so that the maximum number of times that any edge in G is used for, is minimized–an MIP formulation is obvious, with $x_{i,j}$ being the indicator variable for picking the jth path in P_i. This problem, the vector-selection problem of [32], and the discrepancy-type problems of section 4, are all modeled by MIPs; many MIP instances, e.g., global routing, are NP-hard.

We now introduce the next family of integer programs that we will work with.

DEFINITION 2.2 (covering integer programs). *Given $A \in [0,1]^{m \times n}$, $b \in [1, \infty)^m$ and $c \in [0,1]^n$ with $\max_j c_j = 1$, CIP seeks to minimize $c^T \cdot x$ subject to $x \in Z_+^n$ and $Ax \geq b$. If $A \in \{0,1\}^{m \times n}$, each entry of b is assumed integral. We define $B = \min_i b_i$, and let a be the maximum number of nonzero entries in any column of*

A. A CIP is called unweighted if $c_j = 1\ \forall j$, and weighted otherwise.

Note the parameters g, a, and B of Definitions 2.1 and 2.2. Though there are usually no restrictions on the entries of A, b, and c in CIPs aside of nonnegativity, it is well known and easy to check that the above restrictions are without loss of generality. CIPs again model many NP-hard problems in combinatorial optimization. Recall that a hypergraph $H = (V, E)$ is a family of subsets E (edges) of a set V (vertices). The classical *set cover* problem—covering V using the smallest number of edges in E (and its natural weighted version)—is a standard example of a CIP. The parameter a here is the maximum number of vertices in any edge.

Next, there is growing interest in *multicriteria* optimization, since different participating individuals and/or organizations may have different objective functions in a given problem instance; see, e.g., [29]. Motivated by this, we study multicriteria optimization in the setting of covering problems.

DEFINITION 2.3 (multicriteria CIPs; informal). *A multicriteria CIP has a system of constraints "$Ax \geq b$" as in CIPs, and has several given nonnegative vectors $c_1, c_2, \ldots, c_\ell$; the aim is to keep all the values $c_i^T \cdot x$ "low." (For instance, we may aim to minimize $\max_i c_i^T \cdot x$ subject to $Ax \geq b$.) As in Definition 2.2, we assume that $A \in [0,1]^{m \times n}$, $b \in [1, \infty)^m$ and $\forall\ i,\ c_i \in [0,1]^n$ with $max_j\ c_{i,j} = 1$.*

We now present a lemma to quantify our approximation results; its proof is given in section 3.

LEMMA 2.4. *Given independent r.v.s $X_1, \ldots, X_n \in [0,1]$, let $X = \sum_{i=1}^n X_i$ and $\mu = \boldsymbol{E}[X]$.*

a. *For any $\delta > 0$, $\Pr(X \geq \mu(1+\delta)) \leq G(\mu, \delta)$, where $G(\mu, \delta) = \left(e^\delta/(1+\delta)^{1+\delta}\right)^\mu$.*
b. *$\forall \mu > 0\ \forall p \in (0,1), \exists \delta = H(\mu, p) > 0$ such that $\lceil \mu\delta \rceil \cdot G(\mu, \delta) \leq p$ and such that*

$$H(\mu, p) = \Theta\left(\frac{\log(p^{-1})}{\mu \log(\log(p^{-1})/\mu)}\right) \text{ if } \mu \leq \log(p^{-1})/2;$$

$$H(\mu, p) = \Theta\left(\sqrt{\frac{\log(\mu + p^{-1})}{\mu}}\right) \text{ otherwise.}$$

Given an ILP, we can find an optimal solution x^* to its LP relaxation efficiently, but need to round fractional entries in x^* to integers. The idea of randomized rounding is: given a real $v > 0$, round v to $\lfloor v \rfloor + 1$ with probability $v - \lfloor v \rfloor$, and round v to $\lfloor v \rfloor$ with probability $1 - v + \lfloor v \rfloor$. This has the nice property that the mean outcome is v. Starting with this idea, the analysis of [32] produces an integral solution of value at most $y^* + O(\min\{y^*, m\} \cdot H(\min\{y^*, m\}, 1/m))$ for MIPs (though phrased a bit differently); this is derandomized in [31]. But this does not exploit the sparsity of A; the previously-mentioned result of [18] produces an integral solution of value at most $y^* + g + 1$.

For CIPs, the idea is to solve the LP relaxation, scale *up* the components of x^* suitably, and then perform randomized rounding; see section 5 for the details. Starting with this idea, the work of [32] leads to certain approximation bounds; similar bounds are achieved through different means by Plotkin, Shmoys, and Tardos [30]. Work of this author [36] improved upon these results by observing a "correlation" property of CIPs, getting an approximation ratio of $1 + O(\max\{\ln(mB/y^*)/B, \sqrt{\ln(mB/y^*)/B}\})$. Thus, while the work of [32] gives a general approximation bound for MIPs, the result of [18] gives good results for sparse MIPs. For CIPs, the current-best results are those of [36]; however, no better results were known for sparse CIPs.

2.1. Improvements achieved. For MIPs, we use the extended LLL and an idea of Éva Tardos that leads to a bootstrapping of the LLL extension.

THEOREM 2.5. *For any given MIP, there exists an integral solution of value at most* $y^* + O(1) + O(\min\{y^*, m\} \cdot H(\min\{y^*, m\}, 1/a))$. Since $a \leq m$, this is always as good as the $y^* + O(\min\{y^*, m\} \cdot H(\min\{y^*, m\}, 1/m))$ bound of [32] and is a good improvement, if $a \ll m$. It also is an improvement over the additive g factor of [18] in cases where g is not small compared to y^*.

Consider, e.g., the global routing problem and its MIP formulation, sketched above; m here is the number of edges in G, and $g = a$ is the maximum length of any path in $\bigcup_i P_i$. To focus on a specific interesting case, suppose y^*, the fractional congestion, is at most one. Then while the previous results ([32] and [18], resp.) give bounds of $O(\log m / \log\log m)$ and $O(a)$ on an integral solution, we get the improved bound of $O(\log a / \log\log a)$. Similar improvements are easily seen for other ranges of y^* also; e.g., if $y^* = O(\log a)$, an integral solution of value $O(\log a)$ exists, improving on the previously known bounds of $O(\log m / \log(2 \log m / \log a))$ and $O(a)$. Thus, routing along *short* paths (this is the notion of sparsity for the global routing problem) is very beneficial in keeping the congestion low. Section 4 presents a scenario where we get such improvements, for discrepancy-type problems [34, 4]. In particular, we generalize a hypergraph-partitioning result of Füredi and Kahn [16].

Recall the bounds of [36] for CIPs mentioned in the paragraph preceding this subsection; our bounds for CIPs depend only on the set of constraints $Ax \geq b$, i.e., they hold for any nonnegative objective-function vector c. Our improvements over [36] get better as y^* decreases.

THEOREM 2.6. *For any given CIP, there exists a feasible solution of value at most* $y^* \cdot (1 + O(max\{\ln(a+1)/B, \sqrt{\ln(a+B)/B}\}))$.

This CIP bound is better than that of [36] if y^* is small enough. In particular, we generalize the result of Chvátal [10] on weighted set cover. Consider, e.g., a facility location problem on a directed graph $G = (V, A)$: given a cost $c_i \in [0, 1]$ for each $i \in V$, we want a min-cost assignment of facilities to the nodes such that each node sees at least B facilities in its out-neighborhood; multiple facilities at a node are allowed. If Δ_{in} is the maximum in-degree of G, Theorem 2.6 guarantees an integrality gap of $1 + O(\max\{\ln(\Delta_{in}+1)/B, \sqrt{\ln(B(\Delta_{in}+1))/B}\})$. This improves on [36] if $y^* \leq |V|B / \max\{\Delta_{in}, B\}$; it shows an $O(1)$ (resp., $1 + o(1)$) integrality gap if B grows as fast as (resp., strictly faster than) $\log \Delta_{in}$.

A key corollary of Theorem 2.6 is that for families of instances of CIPs, we get a good ($O(1)$ or $1 + o(1)$) integrality gap if B grows at least as fast as $\log a$; bounds on the result of a greedy algorithm for CIPs relative to the optimal *integral* solution are known [12, 13]. Our bound of Theorem 2.6 improves that of [12] and is incomparable with [13]; for any given A, c, and the unit vector $b/||b||_2$, our bound improves on [13] if B is more than a certain threshold. As it stands, randomized rounding produces such improved solutions for several CIPs only with a very low, sometimes exponentially small, probability. Thus, it does not imply a randomized algorithm, often. To this end, we generalize Raghavan's method of pessimistic estimators to derive an algorithmic (polynomial-time) version of Theorem 2.6, in section 5.3.

We also show via Theorem 5.9 and Corollary 5.10 that multicriteria CIPs can be approximated well. In particular, Corollary 5.10 shows some interesting cases where the approximation guarantee for multicriteria CIPs grows in a very much sublinear fashion with the number ℓ of given vectors c_i: the approximation ratio is at most $O(\log\log \ell)$ times what we show for CIPs (which correspond to the case where $\ell = 1$).

We are not aware of any such earlier work on multicriteria CIPs. The constructive version of Corollary 5.10 that we present in section 5.3 requires poly$(n^{\log \ell}, m)$ time, though. It would be interesting to be able to improve this to a polynomial-time algorithm.

2.2. Preliminary version of this work, and followup. A preliminary version of this work appeared in [35], with a sketch of the proof for minimax integer programs, and proofs omitted for our constructive results on covering integer programs. In this version, we provide all proofs, further generalize the main covering result of [35] to Theorem 5.9, and present a sample application of Theorem 5.9 in Corollary 5.10. As mentioned in section 1, two main algorithmic versions related to our work have been obtained following [35]. First, for a subclass of the MIPs where the nonzero entries of the matrix A are "reasonably large," constructive versions of our results have been obtained in [20]. Second, for a notion of approximation that is different from the one we study, algorithmic results have been developed for certain families of MIPs in [11]. Furthermore, our Theorem 2.6 for CIPs has been used in [19] to develop approximation algorithms for CIPs that have given upper bounds on the variables x_j.

3. The extended LLL and an approach to large deviations. We now present our LLL extension, Theorem 3.1. For any event E, define $\chi(E)$ to be its indicator r.v.: 1 if E holds and 0 otherwise. Suppose we have "bad" events $E_1, \ldots, E_m$ with a "dependency" d' (in the sense of Lemma 1.1) that is "large." Theorem 3.1 shows how to essentially replace d' by a possibly much-smaller d, if we are able to define appropriate nonnegative valued random variables $\{C_{i,j}\}$ for each E_i. It generalizes Lemma 1.1 (define one r.v., $C_{i,1} = \chi(E_i)$, for each i, to get Lemma 1.1), and its proof is very similar to the classical proof of Lemma 1.1. The motivation for Theorem 3.1 will be clarified by the applications; in particular, given some additional preparations, Theorems 4.2 and 2.6 follow, respectively, and with relatively less work, from Theorem 3.1 and its proof approach.

THEOREM 3.1. *Given events $E_1, \ldots, E_m$ and any $I \subseteq [m]$, let $Z(I) \doteq \bigwedge_{i \in I} \overline{E_i}$. Suppose that for some positive integer d, we can define, for each $i \in [m]$, a finite number of r.v.s $C_{i,1}, C_{i,2}, \ldots$ each taking on only nonnegative values such that*

(i) *any $C_{i,j}$ is mutually independent of all, but at most d of the events E_k, $k \neq i$, and*

(ii) $\forall I \subseteq ([m] - \{i\})$, $\Pr(E_i \mid Z(I)) \leq \sum_j \boldsymbol{E}[C_{i,j} \mid Z(I)]$.

Let p_i denote $\sum_j \boldsymbol{E}[C_{i,j}]$. Suppose that $\forall i \in [m]$ we have $ep_i(d+1) \leq 1$. Then $\Pr(\bigwedge_i \overline{E_i}) \geq (d/(d+1))^m > 0$.

Remark 3.2. Note, by setting $I = \phi$ in (ii), that $\Pr(E_i) \leq p_i \ \forall\ i$. Also, $C_{i,j}$ and $C_{i,j'}$ can "depend" on *different* subsets of $\{E_k | k \neq i\}$; the only restriction is that these subsets be of size at most d. Note that we have essentially reduced the dependency among the E_is to just d: $ep_i(d+1) \leq 1$ suffices. Another important point is that the dependency *among the r.v.s $C_{i,j}$* could be much higher than d: all we count is the number of E_k that any $C_{i,j}$ depends on.

Proof of Theorem 3.1. We prove by induction on $|I|$ that if $i \notin I$, then $\Pr(E_i \mid Z(I)) \leq ep_i$, which suffices to prove the theorem since $\Pr(\bigwedge_i \overline{E_i}) = \prod_{i \in [m]}(1 - \Pr(E_i \mid Z([i-1])))$. For the base case where $I = \emptyset$, $\Pr(E_i \mid Z(I)) = \Pr(E_i) \leq p_i$. For the inductive step, let $S_{i,j,I} \doteq \{k \in I \mid C_{i,j} \text{ depends on } E_k\}$, and $S'_{i,j,I} = I - S_{i,j,I}$; note that $|S_{i,j,I}| \leq d$. If $S_{i,j,I} = \emptyset$, then $\mathbf{E}[C_{i,j} \mid Z(I)] = \mathbf{E}[C_{i,j}]$. Otherwise, letting

$S_{i,j,I} = \{\ell_1, \ldots, \ell_r\}$, we have

$$\mathbf{E}[C_{i,j} \mid Z(I)] = \frac{\mathbf{E}[C_{i,j} \cdot \chi(Z(S_{i,j,I})) \mid Z(S'_{i,j,I})]}{\Pr(Z(S_{i,j,I}) \mid Z(S'_{i,j,I}))} \leq \frac{\mathbf{E}[C_{i,j} \mid Z(S'_{i,j,I})]}{\Pr(Z(S_{i,j,I}) \mid Z(S'_{i,j,I}))},$$

since $C_{i,j}$ is nonnegative. The numerator of the last term is $\mathbf{E}[C_{i,j}]$, by assumption. The denominator can be lower-bounded as follows:

$$\prod_{s\in[r]} (1 - \Pr(E_{\ell_s} \mid Z(\{\ell_1, \ell_2, \ldots, \ell_{s-1}\} \cup S'_{i,j,I}))) \geq \prod_{s\in[r]} (1 - ep_{\ell_s})$$
$$\geq (1 - 1/(d+1))^r \geq (d/(d+1))^d > 1/e;$$

the first inequality follows from the induction hypothesis. Hence, $\mathbf{E}[C_{i,j} \mid Z(I)] \leq e\mathbf{E}[C_{i,j}]$ and thus, $\Pr(E_i \mid Z(I)) \leq \sum_j \mathbf{E}[C_{i,j} \mid Z(I)] \leq ep_i \leq 1/(d+1)$.

The crucial point is that the events E_i could have a large dependency d', in the sense of the classical Lemma 1.1. The main utility of Theorem 3.1 is that if we can "decompose" each E_i into the r.v.s $C_{i,j}$ that satisfy the conditions of the theorem, then there is the possibility of effectively reducing the dependency noticeably (d' can be replaced by the value d). Concrete instances of this will be studied in later sections.

The tools behind our MIP application are our new LLL, and a result of [33]. Define, for $z = (z_1, \ldots, z_n) \in \Re^n$, a family of polynomials $S_j(z), j = 0, 1, \ldots, n$, where $S_0(z) \equiv 1$, and for $j \in [n]$,

$$S_j(z) \doteq \sum_{1\leq i_1<i_2\cdots<i_j\leq n} z_{i_1} z_{i_2} \cdots z_{i_j}. \tag{3.1}$$

Remark 3.3. For real x and nonnegative integral r, we define $\binom{x}{r} \doteq x(x-1)\cdots(x-r+1)/r!$ as usual; this is the sense meant in Theorem 3.4 below.

We define a nonempty event to be any event with a nonzero probability of occurrence. The relevant theorem of [33] is the following.

THEOREM 3.4 (see [33]). *Given r.v.s $X_1, \ldots, X_n \in [0,1]$, let $X = \sum_{i=1}^n X_i$ and $\mu = \boldsymbol{E}[X]$. Then,*

(a) *For any $q > 0$, any nonempty event Z, and any nonnegative integer $k \leq q$,*

$$\Pr(X \geq q \mid Z) \leq \boldsymbol{E}[Y_{k,q} \mid Z],$$

where $Y_{k,q} = S_k(X_1, \ldots, X_n)/\binom{q}{k}$.

(b) *If the X_is are independent, $\delta > 0$, and $k = \lceil \mu\delta \rceil$, then $\Pr(X \geq \mu(1+\delta)) \leq \boldsymbol{E}[Y_{k,\mu(1+\delta)}] \leq G(\mu, \delta)$, where $G(\cdot,\cdot)$ is as in Lemma 2.4.*

(c) *If the X_is are independent, then $\boldsymbol{E}[S_k(X_1, \ldots, X_n)] \leq \binom{n}{k} \cdot (\mu/n)^k \leq \mu^k/k!$.*

Proof. Suppose $r_1, r_2, \ldots r_n \in [0,1]$ satisfy $\sum_{i=1}^n r_i \geq q$. Then, a simple proof is given in [33], for the fact that for any nonnegative integer $k \leq q$, $S_k(r_1, r_2, \ldots, r_n) \geq \binom{q}{k}$. This clearly holds even given the occurrence of any nonempty event Z. Thus we get $\Pr(X \geq q \mid Z) \leq \Pr(Y_{k,q} \geq 1 \mid Z) \leq \mathbf{E}[Y_{k,q} \mid Z]$, where the second inequality follows from Markov's inequality. The proofs of (b) and (c) are given in [33]. □

We next present the proof of Lemma 2.4.

Proof of Lemma 2.4. Part (a) is the Chernoff–Hoeffding bound (see, e.g., Appendix A of [4], or [27]). For (b), we proceed as follows. For any $\mu > 0$, it is easy to check that

$$G(\mu, \delta) = e^{-\Theta(\mu\delta^2)} \text{ if } \delta \in (0,1), \tag{3.2}$$
$$G(\mu, \delta) = e^{-\Theta(\mu(1+\delta)\ln(1+\delta))} \text{ if } \delta \geq 1. \tag{3.3}$$

Now if $\mu \leq \log(p^{-1})/2$, choose

$$\delta = C \cdot \frac{\log(p^{-1})}{\mu \log(\log(p^{-1})/\mu)}$$

for a suitably large constant C. Since $\mu \leq \log(p^{-1})/2$, we can make $\delta \geq 1$ and have (3.3) hold, by choosing, e.g., $C \geq 1$. Simple algebraic manipulation now shows that if C is large enough, then $\lceil \mu\delta \rceil \cdot G(\mu, \delta) \leq p$ holds. Similarly, if $\mu > \log(p^{-1})/2$, we set $\delta = C \cdot \sqrt{\frac{\log(\mu + p^{-1})}{\mu}}$ for a large enough constant C, and use (3.2).

4. Approximating minimax integer programs. We now employ Theorem 3.1 to develop improved results on integral solutions for MIPs. We broadly proceed in two steps. First, we define a useful parameter t; a relatively direct application of Theorem 3.1 leads to Theorem 4.2. Second, we "bootstrap" Theorem 4.2 to develop the stronger Theorem 2.5. We present the intuitions behind these two theorems in the paragraph preceding Theorem 4.2 and in the paragraph preceding the proof of Theorem 2.5.

Let us first define a parameter t. This is useful as a first step in developing Theorem 4.2, but we will dispense with it when we work up to the proof of Theorem 2.5. Suppose we are given an MIP conforming to Definition 2.1. Define t to be $\max_{i \in [m]} NZ_i$, where NZ_i is the number of rows of A which have a nonzero coefficient corresponding to at least one variable among $\{x_{i,j} : j \in [\ell_i]\}$. Note that

$$g \leq a \leq t \leq \min\{m, a \cdot \max_{i \in [n]} \ell_i\}. \tag{4.1}$$

Theorem 4.2 now shows how Theorem 3.1 can help for sparse MIPs, those where $t \ll m$. We will then bootstrap Theorem 4.2 to get the further-improved Theorem 2.5. We start with a proposition, whose proof is shown in the appendix.

PROPOSITION 4.1. *If $0 < \mu_1 \leq \mu_2$, then for any $\delta > 0$, $G(\mu_1, \mu_2\delta/\mu_1) \leq G(\mu_2, \delta)$.*

Intuition for Theorem 4.2. Given an MIP, let us conduct randomized rounding in a natural way: independently for each i, randomly round exactly one $x_{i,j}$ to 1, with $x_{i,j}$ getting rounded to 1 with probability $x^*_{i,j}$. Now, we have one bad event E_i for each row i of A, corresponding to its row-sum going noticeably above its expectation. We would like to define random variables $C_{i,j}$ corresponding to each E_i, in order to employ Theorem 3.1. If we can choose a suitable integer k, then Theorem 3.4(a) suggests a natural choice for the $C_{i,j}$. Since E_i is an upper-tail event of the type captured by Theorem 3.4(a), we can try and bijectively map the $C_{i,j}$ to the $\binom{n}{k}$ r.v.s "$X_{i_1} X_{i_2} \cdots X_{i_k} / \binom{q}{k}$" that constitute the r.v. "$Y_{k,q}$" of Theorem 3.4(a). Condition (ii) of Theorem 3.1 is satisfied by Theorem 3.4(a); we will bound the parameter d in condition (i) of Theorem 3.1 by using the definition of t. These, in conjunction with some simple observations, lead to Theorem 4.2.

THEOREM 4.2. *Given an MIP conforming to Definition* 2.1*, randomized rounding produces a feasible solution of value at most $y^* + \lceil \min\{y^*, m\} \cdot H(\min\{y^*, m\}, 1/(et)) \rceil$, with nonzero probability.*

Proof. Conduct randomized rounding: independently for each i, randomly round exactly one $x_{i,j}$ to 1, guided by the "probabilities" $\{x^*_{i,j}\}$. Recall again that $x^* = \{x^*_{i,j}\}$ is the vector obtained by solving the LP relaxation. We may assume that $\{x^*_{i,j}\}$ is a *basic* feasible solution to this relaxation. Hence, at most m of the $\{x^*_{i,j}\}$ will be neither zero nor one, and only these variables will participate in the rounding. Since

all the entries of A are in $[0,1]$, we can assume without loss of generality from now on that $y^* \leq m$ (and that $\max_{i\in[n]}\ell_i \leq m$); this explains the "$\min\{y^*, m\}$" term in our stated bounds. Let $b_i = (Ax^*)_i$ denote the r.h.s. value that we get for the ith row in the optimal solution for the LP relaxation; so we have $b_i \leq y^*$. If $z \in \{0,1\}^N$ denotes the randomly rounded vector, then $\mathbf{E}[(Az)_i] = b_i \leq y^*$ by the linearity of expectation. Defining

$$k = \lceil y^* H(y^*, 1/(et)) \rceil \tag{4.2}$$

and events $E_1, E_2, \ldots, E_m$ by $E_i \equiv$ "$(Az)_i \geq b_i + k$," we now show that $\Pr(\bigwedge_{i\in[m]} \overline{E_i}) > 0$ using Theorem 3.1. Rewrite the ith constraint of the MIP as

$$\sum_{r\in[n]} X_{i,r} \leq W, \text{ where } X_{i,r} = \sum_{s\in[\ell_r]} A_{i,(r,s)} x_{r,s};$$

the notation $A_{i,(r,s)}$ assumes that the pairs $\{(r,s) : r \in [n], s \in [\ell_r]\}$ have been mapped bijectively to $[N]$, in some fixed way. Defining the r.v.

$$Z_{i,r} = \sum_{s\in[\ell_r]} A_{i,(r,s)} z_{r,s},$$

we note that for each i, the r.v.s $\{Z_{i,r} : r \in [n]\}$ lie in $[0,1]$ and are *independent*. Also, $E_i \equiv$ "$\sum_{r\in[n]} Z_{i,r} \geq b_i + k$."

Theorem 3.4 suggests a suitable choice for the crucial r.v.s $C_{i,j}$ (to apply Theorem 3.1). Let $u = \binom{n}{k}$; we now define the r.v.s $\{C_{i,j} : i \in [m], j \in [u]\}$ as follows. Fix any $i \in [m]$. Identify each $j \in [u]$ with some distinct k-element subset $S(j)$ of $[n]$, and let

$$C_{i,j} \doteq \frac{\prod_{v\in S(j)} Z_{i,v}}{\binom{b_i+k}{k}}. \tag{4.3}$$

We now need to show that the r.v.s $C_{i,j}$ satisfy the conditions of Theorem 3.1. For any $i \in [m]$, let $\delta_i = k/b_i$. Since $b_i \leq y^*$, we have, for each $i \in [m]$,

$$\begin{aligned} G(b_i, \delta_i) &\leq G(y^*, k/y^*) \text{ (by Proposition 4.1)} \\ &\leq G(y^*, H(y^*, 1/(et))) \text{ (by (4.2))} \\ &\leq 1/(ekt) \text{ (by the definition of } H). \end{aligned}$$

Now by Theorem 3.4, we get the following fact.

FACT 4.3. *For all $i \in [m]$ and for all nonempty events Z, $\Pr(E_i \mid Z) \leq \sum_{j\in[u]} \boldsymbol{E}[C_{i,j} \mid Z]$. Also, $p_i \doteq \sum_{j\in[u]} \boldsymbol{E}[C_{i,j}] < G(b_i, \delta_i) \leq 1/(ekt)$.*

Next since any $C_{i,j}$ involves (a product of) k terms, each of which "depends" on at most $(t-1)$ of the events $\{E_v : v \in ([m] - \{i\})\}$ by definition of t, we see the important

FACT 4.4. *$\forall i \in [m]$ $\forall j \in [u]$, $C_{i,j} \in [0,1]$ and $C_{i,j}$ "depends" on at most $d = k(t-1)$ of the set of events $\{E_v : v \in ([m] - \{i\})\}$.*

From Facts 4.3 and 4.4 and by noting that $ep_i(d+1) \leq e(kt - k + 1)/(ekt) \leq 1$, we invoke Theorem 3.1, to see that $\Pr(\bigwedge_{i\in[m]} \overline{E_i}) > 0$, concluding the proof of Theorem 4.2. □

We are now ready to improve Theorem 4.2, to obtain Theorem 2.5.

Intuition for Theorem 2.5. Theorem 4.2 gives good results if $t \ll m$, but can we improve it further, say by replacing t by a ($\leq t$) in it? As seen from (4.1), the key reason for $t \gg a^{\Theta(1)}$ is that $\max_{i\in[n]} \ell_i \gg a^{\Theta(1)}$. If we can essentially "bring down" $\max_{i\in[n]} \ell_i$ by forcing many $x^*_{i,j}$ to be zero for *each* i, then we effectively reduce t ($t \leq a \cdot max_i\ \ell_i$, see (4.1)); this is so since only those $x^*_{i,j}$ that are neither zero nor one take part in the rounding. A way of bootstrapping Theorem 4.2 to achieve this using a "slow rounding" technique that proceeds in $O(\log\log t)$ iterations, is shown by Theorem 2.5.

Proof of Theorem 2.5. Let $K_0 > 0$ be a sufficiently large absolute constant. Now if

$$(y^* \geq t^{1/7}) \text{ or } (t \leq \max\{K_0, 2\}) \text{ or } (t \leq a^4) \tag{4.4}$$

holds, then we will be done by Theorem 4.2. So we may assume that (4.4) is false. Also, if $y^* \leq t^{-1/7}$, Theorem 4.2 guarantees an integral solution of value $O(1)$; thus, we also suppose that $y^* > t^{-1/7}$. The basic idea now is, as sketched above, to set many $x^*_{i,j}$ to zero for each i (without losing too much on y^*), so that $\max_i \ell_i$ and hence, t, will essentially get reduced. Such an approach, whose performance will be validated by arguments similar to those of Theorem 4.2, is repeatedly applied until (4.4) holds, owing to the (continually reduced) t becoming small enough to satisfy (4.4). There are two cases.

Case I. $y^* \geq 1$. Solve the LP relaxation, and set $x'_{i,j} := (y^*)^2(\log^5 t)x^*_{i,j}$. Conduct randomized rounding on the $x'_{i,j}$ now, rounding *each* $x'_{i,j}$ independently to $z_{i,j} \in \{\lfloor x'_{i,j}\rfloor, \lceil x'_{i,j}\rceil\}$. (Note the key difference from Theorem 4.2, where for each i, we round exactly one $x^*_{i,j}$ to 1.)

Let $K_1 > 0$ be a sufficiently large absolute constant. We now use ideas similar to those used in our proof of Theorem 4.2 to show that with nonzero probability, we have both of the following:

$$\forall i \in [m], (Az)_i \leq (y^*)^3 \log^5 t \cdot (1 + K_1/((y^*)^{1.5}\log^2 t)), \quad \text{and} \tag{4.5}$$

$$\forall i \in [n], \left|\sum_j z_{i,j} - (y^*)^2 \log^5 t\right| \leq K_1 y^* \log^3 t. \tag{4.6}$$

To show this, we proceed as follows. Let $E_1, E_2, \ldots, E_m$ be the "bad" events, one for each event in (4.5) not holding; similarly, let $E_{m+1}, E_{m+2}, \ldots, E_{m+n}$ be the "bad" events, one for each event in (4.6) not holding. We want to use our extended LLL to show that with positive probability, all these bad events can be avoided; specifically, we need a way of associating each E_i with some finite number of nonnegative r.v.s $C_{i,j}$. (Loosely speaking, we will "decompose" each E_i into a finite number of such $C_{i,j}$.) We do this as follows. For each event $E_{m+\ell}$ where $\ell \geq 1$, we define just one r.v. $C_{m+\ell,1}$: this is the indicator variable for the occurrence of $E_{m+\ell}$. For the events E_i where $i \leq m$, we decompose E_i into r.v.s $C_{i,j}$ just as in (4.3): each such $C_{i,j}$ is now a scalar multiple of at most

$$O((y^*)^3 \log^5 t/((y^*)^{1.5}\log^2 t)) = O((y^*)^{1.5}\log^3 t) = O(t^{3/14}\log^3 t)$$

independent binary r.v.s that underlie our randomized rounding; the second equality (big-Oh bound) here follows since (4.4) has been assumed to not hold. Thus, it is easy to see that $\forall\ i$, $1 \leq i \leq m+n$, and for any j, the r.v. $C_{i,j}$ depends on at most

$$O(t \cdot t^{3/14}\log^3 t) \tag{4.7}$$

events E_k, where $k \neq i$. Also, as in our proof of Theorem 4.2, Theorem 3.4 gives a direct proof of requirement (ii) of Theorem 3.1; part (b) of Theorem 3.4 shows that for any desired constant K, we can choose the constant K_1 large enough so that $\forall\, i,\ \sum_j \mathbf{E}[C_{i,j}] \leq t^{-K}$. Thus, in view of (4.7), we see by Theorem 3.1 that $\Pr(\bigwedge_{i=1}^{m+n} \overline{E_i}) > 0$.

Fix a rounding z satisfying (4.5) and (4.6). For each $i \in [n]$ and $j \in [\ell_i]$, we renormalize as follows: $x''_{i,j} := z_{i,j}/\sum_u z_{i,u}$. Thus we have $\sum_u x''_{i,u} = 1$ for all i; we now see that we have two very useful properties. First, since $\sum_j z_{i,j} \geq (y^*)^2 \log^5 t \cdot \left(1 - O(\frac{1}{y^* \log^2 t})\right)$ for all i from (4.6), we have, $\forall i \in [m]$,

$$(Ax'')_i \leq \frac{y^*(1 + O(1/((y^*)^{1.5} \log^2 t)))}{1 - O(1/(y^* \log^2 t))} \leq y^*(1 + O(1/(y^* \log^2 t))). \tag{4.8}$$

Second, since the $z_{i,j}$ are nonnegative integers summing to at most $(y^*)^2 \log^5 t(1 + O(1/(y^* \log^2 t)))$, *at most* $O((y^*)^2 \log^5 t)$ *values* $x''_{i,j}$ *are nonzero*, for each $i \in [n]$. Thus, by losing a little in y^* (see (4.8)), our "scaling up-rounding-scaling down" method has given a fractional solution x'' with a much-reduced ℓ_i for each i; ℓ_i is now $O((y^*)^2 \log^5 t)$, essentially. Thus, t has been reduced to $O(a(y^*)^2 \log^5 t)$; i.e., t has been reduced to at most

$$K_2 t^{1/4+2/7} \log^5 t \tag{4.9}$$

for some constant $K_2 > 0$ that is independent of K_0, since (4.4) was assumed false. Repeating this scheme $O(\log \log t)$ times makes t small enough to satisfy (4.4). More formally, define $t_0 = t$ and $t_{i+1} = K_2 t_i^{1/4+2/7} \log^5 t_i$ for $i \geq 0$. Stop this sequence at the first point where either $t = t_i$ satisfies (4.4) or $t_{i+1} \geq t_i$ holds. Thus, we finally have t small enough to satisfy (4.4) or to be bounded by some absolute constant. How much has $\max_{i \in [m]} (Ax)_i$ increased in the process? By (4.8), we see that at the end,

$$\max_{i \in [m]} (Ax)_i \leq y^* \cdot \prod_{j \geq 0} (1 + O(1/(y^* \log^2 t_j))) \leq y^* \cdot e^{O(\sum_{j \geq 0} 1/(y^* \log^2 t_j))} \leq y^* + O(1), \tag{4.10}$$

since the values $\log t_j$ decrease geometrically and are lower-bounded by some absolute positive constant. We may now apply Theorem 4.2.

Case II. $t^{-1/7} < y^* < 1$. The idea is the same here, with the scaling up of $x^*_{i,j}$ being by $(\log^5 t)/y^*$; the same "scaling up-rounding-scaling down" method works out. Since the ideas are very similar to Case I, we only give a proof sketch here. We now scale up all the $x^*_{i,j}$ first by $(\log^5 t)/y^*$ and do a randomized rounding. The analogs of (4.5) and (4.6) that we now want are

$$\forall i \in [m], (Az)_i \leq \log^5 t \cdot (1 + K'_1/\log^2 t), \quad \text{and} \tag{4.11}$$

$$\forall i \in [n], \left| \sum_j z_{i,j} - \log^5 t/y^* \right| \leq K'_1 \log^3 t/\sqrt{y^*}. \tag{4.12}$$

Proceeding identically as in Case I, we can show that with positive probability, (4.11) and (4.12) hold simultaneously. Fix a rounding where these two properties hold and renormalize as before: $x''_{i,j} := z_{i,j}/\sum_u z_{i,u}$. Since (4.11) and (4.12) hold, we get that

the following analogs of (4.8) and (4.9) hold:

$$(Ax'')_i \le \frac{y^*(1+O(1/\log^2 t))}{1-O(\sqrt{y^*}/\log^2 t)} \le y^*(1+O(1/\log^2 t)); \quad \text{and}$$

t has been reduced to $O(a\log^5 t/y^*)$, i.e., to $O(t^{1/4+1/7}\log^5 t)$.

We thus only need $O(\log\log t)$ iterations, again. Also, the analog of (4.10) now is that

$$\max_{i\in[m]}(Ax)_i \le y^* \cdot \prod_{j\ge 0}(1+O(1/\log^2 t_j)) \le y^* \cdot e^{O(\sum_{j\ge 0} 1/\log^2 t_j)} \le y^* + O(1).$$

This completes the proof of Theorem 2.5. □

We now study our improvements for discrepancy-type problems, which are an important class of MIPs that, among other things, are useful in devising divide-and-conquer algorithms. Given is a set-system (X, F), where $X = [n]$ and $F = \{D_1, D_2, \ldots, D_M\} \subseteq 2^X$. Given a positive integer ℓ, the problem is to partition X into ℓ parts, so that *each* D_j is "split well"; we want a function $f : X \to [\ell]$ which minimizes $\max_{j\in[M],k\in[\ell]} |\{i \in D_j : f(i) = k\}|$. (The case $\ell = 2$ is the standard set-discrepancy problem.) To motivate this problem, suppose we have a (di)graph (V, A); we want a partition of V into $V_1, \ldots, V_\ell$ such that $\forall v \in V$, $\{|\{j \in N(v)\cap V_k\}| : k \in [\ell]\}$ are "roughly the same," where $N(v)$ is the (out-)neighborhood of v; see, e.g., [2, 17] for how this helps construct divide-and-conquer approaches. This problem is naturally modeled by the above set-system problem.

Let Δ be the degree of (X, F), i.e., $\max_{i\in[n]}|\{j : i \in D_j\}|$, and let $\Delta' \doteq \max_{D_j\in F} |D_j|$. Our problem is naturally written as an MIP with $m = M\ell$, $\ell_i = \ell$ for each i, and $g = a = \Delta$, in the notation of Definition 2.1; $y^* = \Delta'/\ell$ here. The analysis of [32] gives an integral solution of value at most $y^*(1+O(H(y^*, 1/(M\ell))))$, while [18] presents a solution of value at most $y^* + \Delta$. Also, since any $D_j \in F$ intersects at most $(\Delta-1)\Delta'$ other elements of F, Lemma 1.1 shows that randomized rounding produces, with positive probability, a solution of value at most $y^*(1 + O(H(y^*, 1/(e\Delta'\Delta\ell))))$. This is the approach taken by [16] for their case of interest: $\Delta = \Delta'$, $\ell = \Delta/\log\Delta$.

Theorem 2.5 shows the existence of an integral solution of value a constant plus $y^*(1 + O(H(y^*, 1/\delta)))$; i.e., *removes the dependence of the approximation factor on* Δ'. This is an improvement on all three results above. As a specific interesting case, suppose ℓ grows at most as fast as $\Delta'/\log\Delta$. Then we see that good integral solutions—those that grow at the rate of $O(y^*)$ or better—exist, which was not known before. (The approach of [16] shows such a result for $\ell = O(\Delta'/\log(\max\{\Delta, \Delta'\}))$. Our bound of $O(\Delta'/\log\Delta)$ is always better than this, and especially so if $\Delta' \gg \Delta$.)

5. Approximating covering integer programs. One of the main ideas behind Theorem 3.1 was to extend the basic inductive proof behind the LLL by decomposing the "bad" events E_i appropriately into the r.v.s $C_{i,j}$. We now use this general idea in a different context, that of (multicriteria) covering integer programs, with an additional crucial ingredient being a useful correlation inequality, the FKG inequality [15]. The reader is asked to recall the discussion of (multicriteria) CIPs from section 2. We start with a discussion of randomized rounding for CIPs, the Chernoff lower-tail bound, and the FKG inequality in section 5.1. These lead to our improved, but non-constructive, approximation bound for column-sparse (multicriteria) CIPs, in section 5.2. This is then made constructive in section 5.3; we also discuss there what we view

as novel about this constructive approach. The two paragraphs marked "Intuition" in section 5.2, as well as the first two paragraphs in section 5.3, describe some of our main ideas here at an intuitive level.

5.1. Preliminaries. Let us start with a simple and well-known approach to tail bounds. Suppose Y is a random variable and y is some value. Then, for any $0 \leq \delta < 1$, we have

$$\Pr(Y \leq y) \leq \Pr((1-\delta)^Y \geq (1-\delta)^y) \leq \frac{E[(1-\delta)^Y]}{(1-\delta)^y}, \tag{5.1}$$

where the inequality is a consequence of Markov's inequality.

We next setup some basic notions related to approximation algorithms for (multicriteria) CIPs. Recall that in such problems, we have ℓ given nonnegative vectors $c_1, c_2, \ldots, c_\ell$ such that $\forall\ i,\ c_i \in [0,1]^n$ with $\max_j c_{i,j} = 1$; $\ell = 1$ in the case of CIPs. Let $x = (x_1^*, x_2^*, \ldots, x_n^*)$ denote a given fractional solution that satisfies the system of constraints $Ax \geq b$. We are not concerned here with how x^* was found: typically, x^* would be an optimal solution to the LP relaxation of the problem. (The LP relaxation is obvious if, e.g., $\ell = 1$, or, say, if the given multicriteria aims to minimize $\max_i c_i^T \cdot x^*$, or to keep each $c_i^T \cdot x^*$ bounded by some target value v_i.) We now consider how to round x^* to some integral z so that

(P1) the constraints $Az \geq b$ hold, and

(P2) for all i, $c_i^T \cdot z$ is "not much bigger" than $c_i^T \cdot x^*$: our approximation bound will be a measure of how small a "not much bigger value" we can achieve in this sense.

Let us now discuss the "standard" randomized rounding scheme for (multicriteria) CIPs. We assume a fixed instance as well as x^*, from now on. For an $\alpha > 1$ to be chosen suitably, set $x'_j = \alpha x_j^*$, for each $j \in [n]$. We then construct a random integral solution z by setting, independently for each $j \in [n]$, $z_j = \lfloor x'_j \rfloor + 1$ with probability $x'_j - \lfloor x'_j \rfloor$, and $z_j = \lfloor x'_j \rfloor$ with probability $1 - (x'_j - \lfloor x'_j \rfloor)$. The aim then is to show that with positive (hopefully high) probability, (P1) and (P2) happen simultaneously. We now introduce some useful notation. For every $j \in [n]$, let $s_j = \lfloor x'_j \rfloor$. Let A_i denote the ith row of A, and let $X_1, X_2, \ldots, X_n \in \{0,1\}$ be *independent* r.v.s with $\Pr(X_j = 1) = x'_j - s_j\ \forall j$. The bad event E_i that the ith constraint is violated by our randomized rounding is given by $E_i \equiv$ "$A_i \cdot X < \mu_i(1-\delta_i)$," where $\mu_i = E[A_i \cdot X]$, s is the vector with entries s_j, and $\delta_i = 1 - (b_i - A_i \cdot s)/\mu_i$. We aim to bound $\Pr(E_i)$ for all i, when the standard randomized rounding is used. We assume without loss of generality that $A_i \cdot s < b_i$ for each i; otherwise, the bad event E_i cannot happen. So, we have $\delta_i \in (0,1)\ \forall i$.

We now bound $\Pr(E_i)$; the proof involves routine Chernoff bounds and calculations, and is shown in the appendix.

LEMMA 5.1. *Define* $g(B,\alpha) \doteq (\alpha \cdot e^{-(\alpha-1)})^B$. *Then* $\forall i$,

$$\Pr(E_i) \leq \frac{\mathrm{E}[(1-\delta_i)^{A_i \cdot X}]}{(1-\delta_i)^{(1-\delta_i)\mu_i}} \leq g(B,\alpha) \leq e^{-B(\alpha-1)^2/(2\alpha)}$$

under standard randomized rounding.

Next, we state a special case of the FKG inequality [15]. Given binary vectors $\vec{a} = (a_1, a_2, \ldots, a_\ell) \in \{0,1\}^\ell$ and $\vec{b} = (b_1, b_2, \ldots, b_\ell) \in \{0,1\}^\ell$, let us partially order them by coordinate-wise domination: $\vec{a} \preceq \vec{b}$ iff $a_i \leq b_i\ \forall\ i$. Now suppose $Y_1, Y_2, \ldots, Y_\ell$ are *independent* r.v.s, each taking values in $\{0,1\}$. Let $\vec{Y}$ denote the

vector $(Y_1, Y_2, \ldots, Y_\ell)$. Suppose an event $\mathcal{A}$ is completely defined by the value of $\vec{Y}$. Define $\mathcal{A}$ to be *increasing* iff: $\forall\ \vec{a} \in \{0,1\}^\ell$ such that $\mathcal{A}$ holds when $\vec{Y} = \vec{a}$, $\mathcal{A}$ also holds when $\vec{Y} = \vec{b}$, for any $\vec{b}$ such that $\vec{a} \preceq \vec{b}$. Analogously, event $\mathcal{A}$ is *decreasing* iff: for all $\vec{a} \in \{0,1\}^\ell$ such that $\mathcal{A}$ holds when $\vec{Y} = \vec{a}$, $\mathcal{A}$ also holds when $\vec{Y} = \vec{b}$, for any $\vec{b} \preceq \vec{a}$, the FKG inequality proves certain intuitively appealing bounds.

LEMMA 5.2 (FKG inequality). *Let $I_1, I_2, \ldots, I_t$ be any sequence of increasing events and $D_1, D_2, \ldots, D_t$ be any sequence of decreasing events (each I_i and D_i completely determined by $\vec{Y}$). Then for any $i \in [t]$ and any $S \subseteq [t]$,*

(i) $\Pr(I_i | \bigwedge_{j \in S} I_j) \geq \Pr(I_i)$ *and* $\Pr(D_i | \bigwedge_{j \in S} D_j) \geq \Pr(D_i)$;

(ii) $\Pr(I_i | \bigwedge_{j \in S} D_j) \leq \Pr(I_i)$ *and* $\Pr(D_i | \bigwedge_{j \in S} I_j) \leq \Pr(D_i)$.

Returning to our random variables X_j and events E_i, we get the following lemma as an easy consequence of the FKG inequality, since each event of the form "$\overline{E_i}$" or "$X_j = 1$" is an increasing event as a function of the vector $(X_1, X_2, \ldots, X_n)$.

LEMMA 5.3. *For all $B_1, B_2 \subseteq [m]$ such that $B_1 \cap B_2 = \emptyset$ and for any $B_3 \subseteq [n]$,* $\Pr(\bigwedge_{i \in B_1} \overline{E_i} \mid ((\bigwedge_{j \in B_2} \overline{E_j}) \wedge (\bigwedge_{k \in B_3} (X_k = 1)))) \geq \prod_{i \in B_1} \Pr(\overline{E_i})$.

5.2. Nonconstructive approximation bounds for (multicriteria) CIPs. We now work up to our main approximation bound for multicriteria CIPs in Theorem 5.9; this theorem is presented in an abstract manner, and one concrete corollary is shown by Corollary 5.10. We develop Theorem 5.9 by starting with the special case of CIPs (which have just one objective function) and proving Theorem 2.6; the basic ideas are then generalized to obtain Theorem 5.9. The randomized rounding approach underlying Theorem 5.9, as it stands, may only construct the solution guaranteed with very low (much less than inverse-polynomial) probability; algorithmic versions of Theorem 5.9, which in some cases involve superpolynomial time, are then developed in section 5.3.

DEFINITION 5.4 (the function $\mathcal{R}$). *For any s and any $j_1 < j_2 < \cdots < j_s$, let $\mathcal{R}(j_1, j_2, \ldots, j_s)$ be the set of indices i such that row i of the constraint system "$Ax \geq b$" has at least one of the variables j_k, $1 \leq k \leq s$, appearing with a nonzero coefficient. (Note from the definition of a in Definition 2.2 that $|\mathcal{R}(j_1, j_2, \ldots, j_s)| \leq a \cdot s$.)*

Let the vector $x^* = (x_1^*, x_2^*, \ldots, x_n^*)$, the parameter $\alpha > 1$, and the "standard" randomized rounding scheme, be as defined in section 5.1. The standard rounding scheme is sufficient for our (nonconstructive) purposes now; we generalize this scheme as follows for later use in section 5.3.

DEFINITION 5.5 (general randomized rounding). *Given a vector $p = (p_1, p_2, \ldots, p_n) \in [0,1]^n$, the general randomized rounding with parameter p generates independent random variables $X_1, X_2, \ldots, X_n \in \{0,1\}$ with $\Pr(X_j = 1) = p_j$; the rounded vector z is defined by $z_j = \lfloor \alpha x_j^* \rfloor + X_j\ \forall\ j$. (As in the standard rounding, we set each z_j to be either $\lfloor \alpha x_j^* \rfloor$ or $\lceil \alpha x_j^* \rceil$; the standard rounding is the special case in which $\boldsymbol{E}[z_j] = \alpha x_j^*\ \forall\ j$.)*

We now present an important lemma, Lemma 5.6, to get correlation inequalities which "point" in the "direction" opposite to FKG. Some ideas from the proof of Lemma 1.1 will play a crucial role in our proof this lemma.

LEMMA 5.6. *Suppose we employ general randomized rounding with some parameter p, and that $\Pr(\bigwedge_{i=1}^m \overline{E_i})$ is nonzero under this rounding. The following hold for*

any q and any $1 \leq j_1 < j_2 < \cdots < j_q \leq n$:

(i)

$$\Pr\left(X_{j_1} = X_{j_2} = \cdots = X_{j_q} = 1 \;\Big|\; \bigwedge_{i=1}^{m} \overline{E_i}\right) \leq \frac{\prod_{t=1}^{q} p_{j_t}}{\prod_{i \in \mathcal{R}(j_1, j_2, \ldots, j_q)} (1 - \Pr(E_i))}; \tag{5.2}$$

the events $E_i \equiv ((Az)_i < b_i)$ are defined here w.r.t. the general randomized rounding.

(ii) *In the special case of standard randomized rounding,*

$$\prod_{i \in \mathcal{R}(j_1, j_2, \ldots, j_q)} (1 - \Pr(E_i)) \geq (1 - g(B, \alpha))^{aq}; \tag{5.3}$$

the function g is as defined in Lemma 5.1.

Proof. (i) Note first that if we wanted a *lower* bound on the l.h.s., the FKG inequality would immediately imply that the l.h.s. is at least $p_{j_1} p_{j_2} \cdots p_{j_q}$. We get around this "correlation problem" as follows. Let $Q = \mathcal{R}(j_1, j_2, \ldots, j_q)$; let $Q' = [m] - Q$. Let

$$Z_1 \equiv \left(\bigwedge_{i \in Q} \overline{E_i}\right), \text{ and } Z_2 \equiv \left(\bigwedge_{i \in Q'} \overline{E_i}\right).$$

Letting $Y = \prod_{t=1}^{q} X_{j_t}$, note that

$$|Q| \leq aq \quad \text{and} \tag{5.4}$$

$$Y \text{ is independent of } Z_2. \tag{5.5}$$

Now,

$$\begin{aligned}
\Pr(Y = 1 \mid (Z_1 \wedge Z_2)) &= \frac{\Pr(((Y = 1) \wedge Z_1) \mid Z_2)}{\Pr(Z_1 \mid Z_2)} \\
&\leq \frac{\Pr((Y = 1) \mid Z_2)}{\Pr(Z_1 \mid Z_2)} \\
&= \frac{\Pr(Y = 1)}{\Pr(Z_1 \mid Z_2)} \quad \text{(by (5.5))} \\
&= \frac{\prod_{t=1}^{q} \Pr(X_{j_t} = 1)}{\Pr(Z_1 \mid Z_2)} \qquad (5.6) \\
&\leq \frac{\prod_{t=1}^{q} \Pr(X_{j_t} = 1)}{\prod_{i \in \mathcal{R}(j_1, j_2, \ldots, j_q)} (1 - \Pr(E_i))} \quad \text{(by Lemma 5.3).}
\end{aligned}$$

(ii) We get (5.3) from Lemma 5.1 and (5.4). ∎

Intuition. Note that in the proof of part (i) of Lemma 5.6, we broadly proceed as in the proofs of Lemma 1.1 and Theorem 3.1 up to (5.6). The key difference in the next (and final) step is that instead of applying induction to lower-bound the denominator as in the proofs of Lemma 1.1 and Theorem 3.1, we are directly able to obtain a lower bound via Lemma 5.3. In addition to giving a better lower bound, this type of explicitly-available denominator will be of considerable value in section 5.3.

We will use Lemmas 5.3 and 5.6 to prove Theorem 5.9. As a warmup, we start with Theorem 2.6, which deals with the special case of CIPs; recall that y^* denotes $c^T \cdot x^*$. We present a simple proposition, whose proof is in the appendix:

PROPOSITION 5.7. *Suppose α and β are chosen as follows, for some sufficiently large absolute constant $K > 0$:*

$$\alpha = K \cdot \ln(a+1)/B \text{ and } \beta = 2, \text{ if } \ln(a+1) \geq B, \text{ and} \tag{5.7}$$

$$\alpha = \beta = 1 + K \cdot \sqrt{\ln(a+B)/B}, \text{ if } \ln(a+1) < B. \tag{5.8}$$

Then we have $\beta(1 - g(B,\alpha))^a > 1$.

We now prove Theorem 2.6, our result for CIPs.

Proof of Theorem 2.6. Let α and β be as in (5.7) and (5.8). Conduct standard randomized rounding, and let $\mathcal{E}$ be the event that $c^T \cdot z > y^* \alpha\beta$. Setting $Z \equiv \bigwedge_{i \in [m]} \overline{E_i}$ and $\mu \doteq \mathbf{E}[c^T \cdot z] = y^*\alpha$, we see by Markov's inequality that $\Pr(\mathcal{E} \mid Z)$ is at most $R = (\sum_{j=1}^n c_j \Pr(X_j = 1 \mid Z))/(\mu\beta)$. Note that $\Pr(Z) > 0$ since $\alpha > 1$; so, we need only prove that $R < 1$. Lemma 5.6 shows that

$$R \leq \frac{\sum_j c_j p_j}{\mu\beta \cdot (1 - g(B,\alpha))^a} = \frac{1}{\beta(1 - g(B,\alpha))^a};$$

thus, the condition $\beta(1 - g(B,\alpha))^a > 1$ suffices. Proposition 5.7 completes the proof. □

Intuition. The basic approach of our proof of Theorem 2.6 is to follow the main idea of Theorem 3.1, and to decompose the event "$\mathcal{E} \mid Z$" into a nonnegative linear combination of events of the form "$X_j = 1 \mid Z$"; we then exploited the fact that each X_j depends on at most a of the events comprising Z. We now extend Theorem 2.6 and also generalize to multicriteria CIPs. Instead of employing just a "first moment method" (Markov's inequality) as in the proof of Theorem 2.6, we will work with higher moments: the functions S_k defined in (3.1) and used in Theorem 3.4.

Suppose some parameters $\lambda_i > 0$ are given, and that our goal is to round x^* to z so that the event

$$\mathcal{A} \equiv \left[(Az \geq b) \ \wedge \ (\forall i, \ c_i^T \cdot z \leq \lambda_i)\right] \tag{5.9}$$

holds. We first give a sufficient condition for this to hold, in Theorem 5.9; we then derive some concrete consequences in Corollary 5.10. We need one further definition before presenting Theorem 5.9. Recall that A_i and b_i, respectively, denote the ith row of A and the ith component of b. Also, the vector s and values δ_i will throughout be as in the definition of standard randomized rounding.

DEFINITION 5.8 (the functions ch and ch′). *Suppose we conduct general randomized rounding with some parameter p; i.e., let $X_1, X_2, \ldots, X_n$ be independent binary random variables such that $\Pr(X_j = 1) = p_j$. For each $i \in [m]$, define*

$$\mathrm{ch}_i(p) \doteq \frac{\boldsymbol{E}[(1-\delta_i)^{A_i \cdot X}]}{(1-\delta_i)^{b_i - A_i \cdot s}} = \frac{\prod_{j \in [n]} \boldsymbol{E}[(1-\delta_i)^{A_{i,j} X_j}]}{(1-\delta_i)^{b_i - A_i \cdot s}} \quad \text{and} \quad \mathrm{ch}'_i(p) \doteq \min\{\mathrm{ch}_i(p), 1\}.$$

(Note from (5.1) *that if we conduct general randomized rounding with parameter p, then $\Pr((Az)_i < b_i) \leq \mathrm{ch}'_i(p)$; also, "*ch*" stands for "Chernoff–Hoeffding.")*

THEOREM 5.9. *Suppose we are given a multicriteria CIP, as well as some parameters $\lambda_1, \lambda_2, \ldots, \lambda_\ell > 0$. Let $\mathcal{A}$ be as in* (5.9). *Then, for any sequence of positive*

integers $(k_1, k_2, \ldots, k_\ell)$ *such that* $k_i \le \lambda_i$, *the following hold:*

(i) *Suppose we employ general randomized rounding with parameter* $p = (p_1, p_2, \ldots, p_n)$. *Then,* $\Pr(\mathcal{A})$ *is at least*

$$\Phi(p) \doteq \left(\prod_{r\in[m]} (1 - \mathrm{ch}'_r(p))\right) - \sum_{i=1}^{\ell} \frac{1}{\binom{\lambda_i}{k_i}} \cdot \sum_{j_1<\cdots<j_{k_i}} \left(\prod_{t=1}^{k_i} c_{i,j_t} \cdot p_{j_t}\right) \cdot \prod_{r\notin\mathcal{R}(j_1,\ldots,j_{k_i})} (1 - \mathrm{ch}'_r(p)); \tag{5.10}$$

as in Definition 2.3, $c_{i,j} \in [0,1]$ *is the* j*th coordinate of the vector* c_i.

(ii) *Suppose we employ the standard randomized rounding to get a rounded vector* z. *Let* $\lambda_i = \nu_i(1+\gamma_i)$ *for each* $i \in [\ell]$, *where* $\nu_i = \boldsymbol{E}[c_i^T \cdot z] = \alpha \cdot (c_i^T \cdot x^*)$ *and* $\gamma_i > 0$ *is some parameter. Then,*

$$\Phi(p) \ge (1 - g(B,\alpha))^m \cdot \left(1 - \sum_{i=1}^{\ell} \frac{\binom{n}{k_i} \cdot (\nu_i/n)^{k_i}}{\binom{\nu_i(1+\gamma_i)}{k_i}} \cdot (1 - g(B,\alpha))^{-a\cdot k_i}\right). \tag{5.11}$$

In particular, if the r.h.s. of (5.11) *is positive, then* $\Pr(\mathcal{A}) > 0$ *for standard randomized rounding.*

The proof of Theorem 5.9 is a simple generalization of that of Theorem 2.6—basically, we use higher moments (the moment S_{k_i} for objective function c_i) and employ Theorem 3.4, instead of using the first moment and Markov's inequality. This proof is deferred to the appendix. Theorem 2.6 is the special case of Theorem 5.9 corresponding to $\ell = k_1 = 1$. To make the general result of Theorem 5.9 more concrete, we now present an additional special case, Corollary 5.10. We provide this as one possible "proof of concept," rather than as an optimized one; e.g., the constant "3" in the bound "$c_i^T \cdot z \le 3\nu_i$" of Corollary 5.10 can be improved. The proof of Corollary 5.10 requires routine calculations after setting $k_i = \lceil \ln(2\ell) \rceil$ and $\gamma_i = 2$ for all i in part (ii) of Theorem 5.9; its proof is given in the appendix.

COROLLARY 5.10. *There is an absolute constant* $K' > 0$ *such that the following holds. Suppose we are given a multicriteria CIP with notation as in part* (ii) *of Theorem* 5.9. *Define* $\alpha = K' \cdot \max\{\frac{\ln(a)+\ln\ln(2\ell)}{B},\ 1\}$. *Now if* $\nu_i \ge \log^2(2\ell)\ \forall\ i \in [\ell]$, *then standard randomized rounding produces a feasible solution* z *such that* $c_i^T \cdot z \le 3\nu_i\ \forall\ i$, *with positive probability.*

In particular, this can be shown by setting $k_i = \lceil \ln(2\ell) \rceil$ *and* $\gamma_i = 2\ \forall\ i$, *in part* (ii) *of Theorem* 5.9.

5.3. Constructive version. It can be shown that for many problems,randomized rounding produces the solutions shown to exist by Theorems 2.6 and 5.9, with very low probability: e.g., probability almost exponentially small in the input size. Thus we need to obtain constructive versions of these theorems. Our method will be a deterministic procedure that makes $O(n)$ calls to the function $\Phi(\cdot)$, in addition to $\mathrm{poly}(n,m)$ work. Now, if k' denotes the maximum of all the k_i, we see that Φ can be evaluated in $\mathrm{poly}(n^{k'}, m)$ time. Thus, our overall procedure runs in time $\mathrm{poly}(n^{k'}, m)$. In particular, we get constructive versions of Theorem 2.6 and Corollary 5.10 that run in time $\mathrm{poly}(n,m)$ and $\mathrm{poly}(n^{\log \ell}, m)$, respectively.

Our approach is as follows. We start with a vector p that corresponds to standard randomized rounding, for which we know (say, as argued in Corollary 5.10) that $\Phi(p) > 0$. In general, we have a vector of probabilities $p = (p_1, p_2, \ldots, p_n)$ such that

$\Phi(p) > 0$. If $p \in \{0,1\}^n$, we are done. Otherwise suppose some p_j lies in $(0,1)$; by renaming the variables, we will assume without loss of generality that $j = n$. Define $p' = (p_1, p_2, \ldots, p_{n-1}, 0)$ and $p'' = (p_1, p_2, \ldots, p_{n-1}, 1)$. The main fact we wish to show is that $\Phi(p') > 0$ or $\Phi(p'') > 0$: we can then set p_n to 0 or 1 appropriately and continue. (As mentioned in the previous paragraph, we thus have $O(n)$ calls to the function $\Phi(\cdot)$ in total.) Note that although some of the p_j will lie in $\{0,1\}$, we can crucially continue to view the X_j as *independent* random variables with $\Pr(X_j = 1) = p_j$.

So, our main goal is: assuming that $p_n \in (0,1)$ and that

$$\Phi(p) > 0, \tag{5.12}$$

to show that $\Phi(p') > 0$ or $\Phi(p'') > 0$. In order to do so, we make some observations and introduce some simplifying notation. Define, for each $i \in [m]$: $q_i = \mathrm{ch}'_i(p)$, $q'_i = \mathrm{ch}'_i(p')$, and $q''_i = \mathrm{ch}'_i(p'')$. Also define the vectors $q \doteq (q_1, q_2, \ldots, q_m)$, $q' \doteq (q'_1, q'_2, \ldots, q'_m)$, and $q'' \doteq (q''_1, q''_2, \ldots, q''_m)$. We now present a useful lemma about these vectors.

LEMMA 5.11. *For all $i \in [m]$, we have*

$$0 \le q''_i \le q'_i \le 1; \tag{5.13}$$

$$q_i \ge p_n q''_i + (1 - p_n) q'_i; \text{ and} \tag{5.14}$$

$$q'_i = q''_i = q_i \text{ if } i \notin \mathcal{R}(n). \tag{5.15}$$

Proof. It is directly seen from the definition of "ch_i" that $\mathrm{ch}_i(p'') \le \mathrm{ch}_i(p')$. Since $q''_i = \min\{\mathrm{ch}_i(p''), 1\}$ and $q'_i = \min\{\mathrm{ch}_i(p'), 1\}$, we get $q''_i \le q'_i$. The remaining inequalities of (5.13), as well as the equalities in (5.15), are straightforward. As for (5.14), we proceed as in [36]. First of all, if $q_i = 1$, then we are done, since $q''_i,\ q'_i \le 1$. So suppose $q_i < 1$; in this case, $q_i = \mathrm{ch}_i(p)$. Now, Definition 5.8 shows that

$$\mathrm{ch}_i(p) = p_n \mathrm{ch}_i(p'') + (1 - p_n)\mathrm{ch}_i(p').$$

Therefore, $q_i = \mathrm{ch}_i(p) = p_n \mathrm{ch}_i(p'') + (1-p_n)\mathrm{ch}_i(p') \ge p_n \mathrm{ch}'_i(p'') + (1-p_n)\mathrm{ch}'_i(p')$. ∎

Since we are mainly concerned with the vectors p, p', and p'' now, we will view the values $p_1, p_2, \ldots, p_{n-1}$ as arbitrary but *fixed*, subject to (5.12). The function $\Phi(\cdot)$ now has a simple form; to see this, we first define, for a vector $r = (r_1, r_2, \ldots, r_m)$ and a set $U \subseteq [m]$,

$$f(U, r) = \prod_{i \in U} (1 - r_i).$$

Recall that $p_1, p_2, \ldots, p_{n-1}$ are considered as constants now. Then, it is evident from (5.10) that there exist constants $u_1, u_2, \ldots, u_t$ and $v_1, v_2, \ldots, v_{t'}$, as well as subsets

$U_1, U_2, \ldots, U_t$ and $V_1, V_2, \ldots, V_{t'}$ of $[m]$, such that

$$\Phi(p) = f([m], q) - \left(\sum_i u_i \cdot f(U_i, q)\right) - \left(p_n \cdot \sum_j v_j \cdot f(V_j, q)\right); \tag{5.16}$$

$$\Phi(p') = f([m], q') - \left(\sum_i u_i \cdot f(U_i, q')\right) - \left(0 \cdot \sum_j v_j \cdot f(V_j, q')\right)$$

$$= f([m], q') - \sum_i u_i \cdot f(U_i, q'); \tag{5.17}$$

$$\Phi(p'') = f([m], q'') - \left(\sum_i u_i \cdot f(U_i, q'')\right) - \left(1 \cdot \sum_j v_j \cdot f(V_j, q'')\right)$$

$$= f([m], q'') - \left(\sum_i u_i \cdot f(U_i, q'')\right) - \left(\sum_j v_j \cdot f(V_j, q'')\right). \tag{5.18}$$

Importantly, we also have the following:

$$\text{the constants } u_i, v_j \text{ are nonnegative; } \quad \forall j,\ V_j \cap \mathcal{R}(n) = \emptyset. \tag{5.19}$$

Recall that our goal is to show that $\Phi(p') > 0$ or $\Phi(p'') > 0$. We will do so by proving that

$$\Phi(p) \leq p_n \Phi(p'') + (1 - p_n)\Phi(p'). \tag{5.20}$$

Let us use the equalities (5.16), (5.17), and (5.18). In view of (5.15) and (5.19), the term "$-p_n \cdot \sum_j v_j \cdot f(V_j, q)$" on both sides of the inequality (5.20) cancels; defining $\Delta(U) \doteq (1 - p_n) \cdot f(U, q') + p_n \cdot f(U, q'') - f(U, q)$, inequality (5.20) reduces to

$$\Delta([m]) - \sum_i u_i \cdot \Delta(U_i) \geq 0. \tag{5.21}$$

Before proving this, we pause to note a challenge we face. Suppose we only had to show that, say, $\Delta([m])$ is nonnegative; this is exactly the issue faced in [36]. Then, we will immediately be done by part (i) of Lemma 5.12, which states that $\Delta(U) \geq 0$ for any set U. However, (5.21) also has terms such as "$u_i \cdot \Delta(U_i)$" with a *negative* sign in front. To deal with this, we need something more than just that $\Delta(U) \geq 0$ for all U; we handle this by part (ii) of Lemma 5.12. We view this as the main novelty in our constructive version here.

LEMMA 5.12. *Suppose* $U \subseteq V \subseteq [m]$. *Then,* (i) $\Delta(U) \geq 0$, *and* (ii) $\Delta(U)/f(U, q) \leq \Delta(V)/f(V, q)$. *(Since* $\Phi(p) > 0$ *by* (5.12)*, we have that* $q_i < 1$ *for each* i. *So,* $1/f(U, q)$ *and* $1/f(V, q)$ *are well defined.)*

Assuming that Lemma 5.12 is true, we will now show (5.21); the proof of Lemma

5.12 is given below. We have

$$\begin{aligned}\Delta([m]) - \sum_i u_i \cdot \Delta(U_i) &= (\Delta([m])/f([m],q)) \cdot f([m],q) \\ &\quad - \sum_i (\Delta(U_i)/f(U_i,q)) \cdot u_i \cdot f(U_i,q) \\ &\geq (\Delta([m])/f([m],q)) \cdot \left[f([m],q) - \sum_i u_i \cdot f(U_i,q) \right] \\ &\quad \text{(by Lemma 5.12)} \\ &\geq 0 \quad \text{(by (5.12) and (5.16))}.\end{aligned}$$

Thus we have (5.21).

Proof of Lemma 5.12. It suffices to show the following. Assume $U \neq [m]$; suppose $u \in ([m] - U)$ and that $U' = U \cup \{u\}$. Assuming by induction on $|U|$ that $\Delta(U) \geq 0$, we show that $\Delta(U') \geq 0$ and that $\Delta(U)/f(U,q) \leq \Delta(U')/f(U',q)$. It is easy to check that this way we will prove both claims of the lemma.

The base case of the induction is that $|U| \in \{0,1\}$, where $\Delta(U) \geq 0$ is directly seen by using (5.14). Suppose inductively that $\Delta(U) \geq 0$. Using the definition of $\Delta(U)$ and the fact that $f(U',q) = (1-q_u)f(U,q)$, we have

$$\begin{aligned}f(U',q) &= (1-q_u) \cdot [(1-p_n)f(U,q') + p_n f(U,q'') - \Delta(U)] \\ &\leq (1 - (1-p_n)q'_u - p_n q''_u) \cdot [(1-p_n)f(U,q') + p_n f(U,q'')] - (1-q_u) \cdot \Delta(U),\end{aligned}$$

where this last inequality is a consequence of (5.14). Therefore, using the definition of $\Delta(U')$ and the facts $f(U',q') = (1-q'_u)f(U,q')$ and $f(U',q'') = (1-q''_u)f(U,q'')$,

$$\begin{aligned}\Delta(U') &= (1-p_n)(1-q'_u)f(U,q') + p_n(1-q''_u)f(U,q'') - f(U',q) \\ &\geq (1-p_n)(1-q'_u)f(U,q') + p_n(1-q''_u)f(U,q'') \\ &\quad + (1-q_u) \cdot \Delta(U) - (1 - (1-p_n)q'_u - p_n q''_u) \cdot [(1-p_n)f(U,q') + p_n f(U,q'')] \\ &= (1-q_u) \cdot \Delta(U) + p_n(1-p_n) \cdot (f(U,q'') - f(U,q')) \cdot (q'_u - q''_u) \\ &\geq (1-q_u) \cdot \Delta(U) \quad \text{(by (5.13))}.\end{aligned}$$

So, since we assumed that $\Delta(U) \geq 0$, we get $\Delta(U') \geq 0$; furthermore, we get that $\Delta(U') \geq (1-q_u) \cdot \Delta(U)$, which implies that $\Delta(U')/f(U',q) \geq \Delta(U)/f(U,q)$. □

6. Conclusion. We have presented an extension of the LLL that basically helps reduce the "dependency" much in some settings; we have seen applications to two families of integer programming problems. It would be interesting to see how far these ideas can be pushed further. Two other open problems suggested by this work are: (i) developing a constructive version of our result for MIPs, and (ii) developing a $\text{poly}(n,m)$-time constructive version of Theorem 5.9, as opposed to the $\text{poly}(n^{k'},m)$-time constructive version that we presented in section 5.3.

Finally, a very interesting question is to develop a theory of applications of the LLL (Lemma 1.1) that can be made constructive with (essentially) no loss. Suppose we have bad events $E_1, E_2, \ldots, E_m$ in the setting of the LLL, which are functions of n independent binary random variables $X_1, X_2, \ldots, X_n$, where $\Pr(X_i = 1) = p_i$. In an attempt to mimic the approach described in the second paragraph of section 5.3, suppose we proceed as follows. The standard proof of the LLL [14] presents a function Φ such that: (i) $\Pr(\bigwedge_{i=1}^m \overline{E_i}) > 0$ if $\Phi(p) > 0$, and (ii) if the conditions of the LLL

hold, then $\Phi(p) > 0$. Can we now try to set values for the X_i one-by-one, as in section 5.3? Unfortunately, there are two problems we face in this regard. First, the function Φ given by the proof of the LLL does not appear to be polynomial-time computable, basically due to the type of induction it uses; indeed, as briefly mentioned in the paragraph following the proof of Lemma 5.6, the type of "explicitly-available denominator" that we obtain in the proof of Lemma 5.6 is one crucial driver in obtaining an efficiently-computable Φ in section 5.3. Second, again appealing to the notation of the second paragraph of section 5.3, it is unclear if we can prove here that "$\Phi(p') > 0$ or $\Phi(p'') > 0$." (In fact, Joel Spencer has suggested to the author the possibility that such a disjunction may not hold for all applications of the LLL.) It would be of much interest to guarantee these for some interesting class of applications of the LLL, or to develop fresh approaches to obtain constructive versions of the LLL with (essentially) no loss.

Appendix. Proofs and calculation-details.

Proof of Proposition 4.1. Taking logarithms on both sides, we aim to show that

$$\mu_1 \cdot [\mu_2\delta/\mu_1 - (1+\mu_2\delta/\mu_1)\ln(1+\mu_2\delta/\mu_1)] \leq \mu_2 \cdot [\delta - (1+\delta)\ln(1+\delta)]\,.$$

This simplifies to showing

$$(1+\psi\delta)\ln(1+\psi\delta) \geq \psi \cdot (1+\delta)\ln(1+\delta),$$

where $\psi = \mu_2/\mu_1 \geq 1$. This inequality, in turn, follows from the fact that the function $t \mapsto (1+t)\ln(1+t)$ is convex for $t \geq 0$ (since its second derivative, $1/(1+t)$, is positive), and since this function equals 0 when $t = 0$. □

Proof of Lemma 5.1. The first inequality follows from (5.1). Next, the Chernoff–Hoeffding lower-tail approach [4, 27] shows that

$$\frac{E[(1-\delta_i)^{A_i\cdot X}]}{(1-\delta_i)^{(1-\delta_i)\mu_i}} \leq \left(\frac{e^{-\delta_i}}{(1-\delta_i)^{1-\delta_i}}\right)^{\mu_i} = \frac{{\mu_i}^{b_i - A_i\cdot s}\cdot e^{b_i-\mu_i-A_i\cdot s}}{(b_i - A_i\cdot s)^{b_i - A_i\cdot s}}. \tag{A.1}$$

Recall that

$$A_i \cdot s < b_i; \quad A_i\cdot s + \mu_i \geq \alpha b_i; \quad \alpha > 1. \tag{A.2}$$

Let us now bound the last (third) term in (A.1), denoted ψ, say.

Fix all parameters other than μ_i; subject to (A.2), let us first observe that ψ is maximized when the second inequality in (A.2) is an equality. This is easily seen by recalling the definition of ψ (from (A.1)) and noting by differentiation that ψ is a decreasing function of μ_i when $\mu_i > b_i - A_i\cdot s$. Next, since $\mu_i = \alpha b_i - A_i\cdot s$,

$$\psi = \left(\frac{\alpha b_i - A_i\cdot s}{b_i - A_i\cdot s}\right)^{b_i - A_i\cdot s}\cdot e^{(1-\alpha)b_i} = \left(1 + \frac{(\alpha-1)b_i}{b_i - A_i\cdot s}\right)^{b_i - A_i\cdot s}\cdot e^{(1-\alpha)b_i}.$$

Let $x = (\alpha-1)b_i/(b_i - A_i\cdot s)$, and note that $x \geq \alpha - 1$. We have

$$\ln\psi = (1-\alpha)b_i + (\alpha-1)b_i\ln(1+x)/x. \tag{A.3}$$

Keeping b_i fixed,

$$\begin{aligned}\frac{d(\ln\psi)}{dx} &= (\alpha-1)b_i\cdot\frac{x/(1+x)-\ln(1+x)}{x^2}\\ &= (\alpha-1)b_i\cdot\frac{x/(1+x)+\ln(1-x/(1+x))}{x^2}\\ &\le (\alpha-1)b_i\cdot\frac{x/(1+x)-x/(1+x)}{x^2}\\ &= 0.\end{aligned}$$

So, ψ is maximized when $x=\alpha-1$ (i.e., when $A_i\cdot s=0$). From (A.3), $\ln\psi = b_i\cdot(1-\alpha+\ln\alpha)$; the term multiplying b_i here is nonpositive, since $\ln\alpha=\ln(1+\alpha-1)\le\alpha-1$. Therefore, ψ is maximized when b_i equals its minimum value of B, and so, $\psi\le g(B,\alpha)$.

Finally, let us check that $g(B,\alpha)\le e^{-B(\alpha-1)^2/(2\alpha)}$. Taking logarithms on both sides, we want

$$1-\alpha+\ln\alpha\le-(\alpha-1)^2/(2\alpha) \tag{A.4}$$

for $\alpha\ge1$. The l.h.s. and r.h.s. of (A.4) are equal when $\alpha=1$; their respective derivatives are $-1+1/\alpha$ and $-1/2+1/(2\alpha^2)$, and so it suffices to prove that $1-1/\alpha\ge1/2-1/(2\alpha^2)$ for $\alpha\ge1$. That is, we need $2\alpha(\alpha-1)\ge(\alpha-1)\cdot(\alpha+1)$, which is true since $\alpha\ge1$. □

Proof of Proposition 5.7. Suppose first that $\ln(a+1)\ge B$. Then, $\alpha=K\cdot\ln(a+1)/B$ and so if K is large enough, then $\alpha/e^{\alpha-1}\le e^{-\alpha/2}$. Therefore,

$$g(B,\alpha)=(\alpha\cdot e^{-(\alpha-1)})^B\le e^{-\alpha B/2}=(a+1)^{-K/2}.$$

So, since $\beta=2$, our goal of proving that $\beta(1-g(B,\alpha))^a>1$ reduces to proving that $2\cdot(1-(a+1)^{-K/2})^a>1$, which is true since $a\ge1$ and K is chosen sufficiently large.

We next consider the case where $\ln(a+1)<B$. Recall from the statement of Lemma 5.1 that $g(B,\alpha)\le e^{-B(\alpha-1)^2/(2\alpha)}$. Since $\alpha<K+1$ in our case, we now have

$$g(B,\alpha)<e^{-B(\alpha-1)^2/(2(K+1))}=e^{-K^2\ln(a+B)/(2(K+1))}=(a+B)^{-K^2/(2(K+1))}.$$

So, our goal of showing that $\beta(1-g(B,\alpha))^a>1$ reduces to proving that

$$(1+K\cdot\sqrt{\ln(a+B)/B})\cdot(1-(a+B)^{-K^2/(2(K+1))})^a\ge1,$$

which in turn holds if $(1+K\cdot\sqrt{\ln(a+B)/B})\cdot(1-a\cdot(a+B)^{-K^2/(2(K+1))})\ge1$; this final inequality is true if we choose K to be a sufficiently large constant, since $a\cdot(a+B)^{-K^2/(2(K+1))}\ll K\cdot\sqrt{\ln(a+B)/B}$ under such a choice. □

Proof of Theorem 5.9. (i) Let $E_r\equiv((Az)_r<b_r)$ be defined w.r.t. general randomized rounding with parameter p; as observed in Definition 5.8, $\Pr(E_r)\le\mathrm{ch}'_r(p)$. Now if $\mathrm{ch}'_r(p)=1$ for some r, then part (i) is trivially true; so we assume that $\Pr(E_r)\le\mathrm{ch}'_r(p)<1\ \forall\ r\in[m]$. Defining $Z\equiv(Az\ge b)\equiv\bigwedge_{r\in[m]}\overline{E_r}$, we get by the FKG inequality that

$$\Pr(Z)\ge\prod_{r\in[m]}(1-\Pr(E_r)).$$

Define, for $i = 1, 2, \ldots, \ell$, the "bad" event $\mathcal{E}_i \equiv (c_i^T \cdot z > \lambda_i)$. Fix any i. Our plan is to show that

(A.5)
$$\Pr(\mathcal{E}_i \mid Z) \leq \frac{1}{\binom{\lambda_i}{k_i}} \cdot \sum_{j_1 < j_2 < \cdots < j_{k_i}} \left(\prod_{t=1}^{k_i} c_{i,j_t} \cdot p_{j_t} \right) \cdot \left(\prod_{r \in \mathcal{R}(j_1, j_2, \ldots, j_{k_i})} (1 - \Pr(E_r))^{-1} \right).$$

If we prove (A.5), then we will be done as follows. We have

(A.6)
$$\Pr(\mathcal{A}) \geq \Pr(Z) \cdot \left(1 - \sum_i \Pr(\mathcal{E}_i \mid Z) \right) \geq \left(\prod_{r \in [m]} (1 - \Pr(E_r)) \right) \cdot \left(1 - \sum_i \Pr(\mathcal{E}_i \mid Z) \right).$$

Now, the term "$(\prod_{r \in [m]} (1 - \Pr(E_r)))$" is a decreasing function of each of the values $\Pr(E_r)$; so is the lower bound on "$-\Pr(\mathcal{E}_i \mid Z)$" obtained from (A.5). Hence, bounds (A.5) and (A.6), along with the bound $\Pr(E_r) \leq \mathrm{ch}'_r(p)$, will complete the proof of part (i).

We now prove (A.5) using Theorem 3.4(a) and Lemma 5.6. Recall the symmetric polynomials S_k from (3.1). Define $Y = S_{k_i}(c_{i,1} X_1, c_{i,2} X_2, \ldots, c_{i,n} X_n) / \binom{\lambda_i}{k_i}$. By Theorem 3.4(a), $\Pr(\mathcal{E}_i \mid Z) \leq \mathbf{E}[Y \mid Z]$. Next, the typical term in $\mathbf{E}[Y \mid Z]$ can be upper bounded using Lemma 5.6

$$\mathbf{E}\left[\left(\prod_{t=1}^{k_i} c_{i,j_t} \cdot X_{j_t} \right) \mid \bigwedge_{i=1}^{m} \overline{E_i} \right] \leq \frac{\prod_{t=1}^{k_i} c_{i,j_t} \cdot p_{j_t}}{\prod_{r \in \mathcal{R}(j_1, j_2, \ldots, j_{k_i})} (1 - \Pr(E_r))}.$$

Thus we have (A.5), and the proof of part (i) is complete.

(ii) We have

$$\Phi(p) = \left[\prod_{r \in [m]} (1 - \mathrm{ch}'_r(p)) \right] \cdot \left(1 - \sum_{i=1}^{\ell} \frac{1}{\binom{\lambda_i}{k_i}} \cdot \sum_{j_1 < \cdots < j_{k_i}} \left[\prod_{t=1}^{k_i} c_{i,j_t} \cdot p_{j_t} \right] \cdot \left(\prod_{r \in \mathcal{R}(j_1, \ldots, j_{k_i})} \frac{1}{1 - \mathrm{ch}'_r(p)} \right) \right). \tag{A.7}$$

Lemma 5.1 shows that under standard randomized rounding, $\mathrm{ch}'_r(p) \leq g(B, \alpha) < 1 \; \forall \; r$. So, the r.h.s. κ of (A.7) gets lower-bounded as follows:

$$\begin{aligned}
\kappa &\geq (1 - g(B,\alpha))^m \cdot \left(1 - \sum_{i=1}^{\ell} \frac{1}{\binom{\nu_i(1+\gamma_i)}{k_i}} \cdot \sum_{j_1 < \cdots < j_{k_i}} \left(\prod_{t=1}^{k_i} c_{i,j_t} \cdot p_{j_t} \right) \right. \\
&\quad \left. \cdot \left[\prod_{r \in \mathcal{R}(j_1, \ldots, j_{k_i})} (1 - g(B,\alpha)) \right]^{-1} \right) \\
&\geq (1 - g(B,\alpha))^m \cdot \left(1 - \sum_{i=1}^{\ell} \frac{1}{\binom{\nu_i(1+\gamma_i)}{k_i}} \cdot \sum_{j_1 < \cdots < j_{k_i}} \left(\prod_{t=1}^{k_i} c_{i,j_t} \cdot p_{j_t} \right) \cdot (1 - g(B,\alpha))^{-a k_i} \right) \\
&\geq (1 - g(B,\alpha))^m \cdot \left(1 - \sum_{i=1}^{\ell} \frac{\binom{n}{k_i} \cdot (\nu_i / n)^{k_i}}{\binom{\nu_i(1+\gamma_i)}{k_i}} \cdot (1 - g(B,\alpha))^{-a k_i} \right),
\end{aligned}$$

where the last line follows from Theorem 3.4(c). □

Proof of Corollary 5.10. Let us employ Theorem 5.9(ii) with $k_i = \lceil \ln(2\ell) \rceil$ and $\gamma_i = 2 \; \forall \; i$. We just need to establish that the r.h.s. of (5.11) is positive. We need to show that

$$\sum_{i=1}^{\ell} \frac{\binom{n}{k_i} \cdot (\nu_i/n)^{k_i}}{\binom{3\nu_i}{k_i}} \cdot (1 - g(B,\alpha))^{-a \cdot k_i} < 1;$$

it is sufficient to prove that for all i,

$$\frac{\nu_i^{k_i}/k_i!}{\binom{3\nu_i}{k_i}} \cdot (1 - g(B,\alpha))^{-a \cdot k_i} < 1/\ell. \tag{A.8}$$

We make two observations now.

- Since $k_i \sim \ln \ell$ and $\nu_i \geq \log^2(2\ell)$,

$$\binom{3\nu_i}{k_i} = (1/k_i!) \cdot \prod_{j=0}^{k_i - 1} (3\nu_i - j) = (1/k_i!) \cdot (3\nu_i)^{k_i} \cdot e^{-\Theta(\sum_{j=0}^{k_i-1} j/\nu_i)} = \Theta((1/k_i!) \cdot (3\nu_i)^{k_i}).$$

- $(1 - g(B,\alpha))^{-a \cdot k_i}$ can be made arbitrarily close to 1 by choosing the constant K' large enough.

These two observations establish (A.8). □

Acknowledgments. This work started while visiting the Sandia National Laboratories in the summer of 1994; I thank Leslie Goldberg and Z. Sweedyk who were involved in the early stages of this work. I would like to thank Éva Tardos for suggesting the key idea that helped bootstrap Theorem 4.2 to get Theorem 2.5. I also thank Noga Alon, Alan Frieze, Tom Leighton, Chi-Jen Lu, Alessandro Panconesi, Prabhakar Raghavan, Satish Rao, Joel Spencer, the SODA 1996 referees, and journal referees for their very helpful comments and suggestions.

REFERENCES

[1] N. Alon, *A parallel algorithmic version of the local lemma*, Random Structures Algorithms, 2 (1991), pp. 367–378.
[2] N. Alon, *The strong chromatic number of a graph*, Random Structures Algorithms, 3 (1992), pp. 1–7.
[3] N. Alon, L. Babai, and A. Itai, *A fast and simple randomized parallel algorithm for the maximal independent set problem*, J. Algorithms, 7 (1986), pp. 567–583.
[4] N. Alon and J. H. Spencer, *The Probabilistic Method*, 2nd ed., John Wiley and Sons, New York, 2000.
[5] J. Beck, *An algorithmic approach to the Lovász Local Lemma*, Random Structures Algorithms, 2 (1991), pp. 343–365.
[6] J. Beck and T. Fiala, *"Integer-making" theorems*, Discrete Appl. Math., 3 (1981), pp. 1–8.
[7] J. Beck and J. H. Spencer, *Integral approximation sequences*, Math. Program., 30 (1984), pp. 88–98.
[8] B. Berger and J. Rompel, *Simulating* $(\log^c n)$*-wise independence in NC*, J. Assoc. Comput. Math., 38 (1991), pp. 1026–1046.
[9] A. Z. Broder, A. M. Frieze, and E. Upfal, *Static and dynamic path selection on expander graphs: A random walk approach*, Random Structures Algorithms, 14 (1999), pp. 87–109.
[10] V. Chvátal, *A greedy heuristic for the set covering problem*, Math. Oper. Res., 4 (1979), pp. 233–235.

[11] A. Czumaj and C. Scheideler, *An algorithmic approach to the general Lovász Local Lemma with applications to scheduling and satisfiability problems*, in Proceedings of the ACM Symposium on Theory of Computing, Portland, OR, 2000, pp. 38–47.
[12] G. Dobson, *Worst-case analysis of greedy heuristics for integer programming with nonnegative data*, Math. Oper. Res., 7 (1982), pp. 515–531.
[13] M. L. Fisher and L. A. Wolsey, *On the greedy heuristic for continuous covering and packing problems*, SIAM J. Algebraic Discrete Methods, 3 (1982), pp. 584–591.
[14] P. Erdős and L. Lovász, *Problems and results on 3-chromatic hypergraphs and some related questions*, Infinite and Finite Sets, A. Hajnal et al., eds., Colloq. Math. Soc. J. Bolyai 11, North-Holland, Amsterdam, 1975, pp. 609–627.
[15] C. M. Fortuin, J. Ginibre, and P. W. Kasteleyn, *Correlational inequalities for partially ordered sets*, Comm. of Math. Phys., 22 (1971), pp. 89–103.
[16] Z. Füredi and J. Kahn, *On the dimensions of ordered sets of bounded degree*, Order, 3 (1986), pp. 15–20.
[17] H. J. Karloff and D. B. Shmoys, *Efficient parallel algorithms for edge coloring problems*, J. Algorithms, 8 (1987), pp. 39–52.
[18] R. M. Karp, F. T. Leighton, R. L. Rivest, C. D. Thompson, U. V. Vazirani, and V. V. Vazirani, *Global wire routing in two-dimensional arrays*, Algorithmica, 2 (1987), pp. 113–129.
[19] S. G. Kolliopoulos and N. E. Young, *Tight approximation results for general covering integer programs*, in Proceedings of the IEEE Symposium on Foundations of Computer Science, Las Vegas, 2001, pp. 522–528.
[20] F. T. Leighton, C. J. Lu, S. B. Rao, and A. Srinivasan, *New algorithmic aspects of the local lemma with applications to routing and partitioning*, SIAM J. Comput., 31 (2001), pp. 626–641.
[21] F. T. Leighton, S. B. Rao, and A. Srinivasan, *Multicommodity flow and circuit switching*, in Proceedings of the Hawaii International Conference on System Sciences, Kohala Coast, HI, 1998, pp. 459–465.
[22] L. Lovász, *On the ratio of optimal integral and fractional covers*, Discrete Math., 13 (1975), pp. 383–390.
[23] M. Luby, *A simple parallel algorithm for the maximal independent set problem*, SIAM J. Comput., 15 (1986), pp. 1036–1053.
[24] M. Molloy and B. Reed, *Further algorithmic aspects of the local lemma*, in Proceedings of the ACM Symposium on Theory of Computing, Dallas, 1998, pp. 524–529.
[25] M. Molloy and B. Reed, *Graph Coloring and the Probabilistic Method*, Springer-Verlag, Berlin, 2002.
[26] R. Motwani, J. Naor, and M. Naor, *The probabilistic method yields deterministic parallel algorithms*, J. Comput. System Sci., 49 (1994), pp. 478–516.
[27] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.
[28] N. Nisan, *Pseudorandom generators for space–bounded computation*, Combinatorica, 12 (1992), pp. 449–461.
[29] C. H. Papadimitriou and M. Yannakakis, *On the approximability of trade-offs and optimal access of web sources*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, Redondo Beach, 2000, pp. 86–92.
[30] S. A. Plotkin, D. B. Shmoys, and É. Tardos, *Fast approximation algorithms for fractional packing and covering problems*, Math. Oper. Res., 20 (1995), pp. 257–301.
[31] P. Raghavan, *Probabilistic construction of deterministic algorithms: Approximating packing integer programs*, J. Comput. System Sci., 37 (1988), pp. 130–143.
[32] P. Raghavan and C. D. Thompson, *Randomized rounding: A technique for provably good algorithms and algorithmic proofs*, Combinatorica, 7 (1987), pp. 365–374.
[33] J. P. Schmidt, A. Siegel, and A. Srinivasan, *Chernoff–Hoeffding bounds for applications with limited independence*, SIAM J. Discrete Math., 8 (1995), pp. 223–250.
[34] J. H. Spencer, *Ten Lectures on the Probabilistic Method*, SIAM, Philadelphia, 1987.
[35] A. Srinivasan, *An extension of the Lovász Local Lemma, and its applications to integer programming*, in Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, Atlanta, 1996, pp. 6–15.
[36] A. Srinivasan, *Improved approximation guarantees for packing and covering integer programs*, SIAM J. Comput., 29 (1999), pp. 648–670.

SIAM J. COMPUT.
Vol. 36, No. 3, pp. 635–656

APPROXIMATING LONGEST CYCLES IN GRAPHS WITH BOUNDED DEGREES*

GUANTAO CHEN†, ZHICHENG GAO‡, XINGXING YU§, AND WENAN ZANG¶

Abstract. Jackson and Wormald conjecture that if G is a 3-connected n-vertex graph with maximum degree $d \geq 4$, then G has a cycle of length $\Omega(n^{\log_{d-1} 2})$. We show that this conjecture holds when $d-1$ is replaced by $\max\{64, 4d+1\}$. Our proof implies a cubic algorithm for finding such a cycle.

Key words. bounded degree, 3-connected components, long cycles, algorithm

AMS subject classifications. 05C38, 35C45, 05C85

DOI. 10.1137/050633263

1. Introduction. From the point of view of approximation algorithms, finding a longest cycle in a graph is one of the "harder" NP-hard problems. There is no known polynomial time algorithm which guarantees an approximation ratio better than $n/\text{polylog}(n)$. For graphs with a cycle of length k, it is shown in [1] that one can find in polynomial time a cycle of length $\Omega((\log k)^2/\log\log k)$. Gabow [6] showed how to find in polynomial time a cycle of length $\exp(\Omega(\sqrt{\log k/\log\log k}))$ through a given vertex v in a graph that contains a cycle of length k through v. Recently, Feder and Motwani [5] obtained a cubic algorithm which, given a graph with maximum degree d and containing a k-vertex 3-cyclable minor, finds a cycle of length $k^{1/(2c\log d)}$ for some $c \geq 2$. A consequence of their result improves Gabow's result in certain situations.

Karger, Motwani, and Ramkumar [10] showed that unless $\mathcal{P} = \mathcal{NP}$ it is impossible to find, in polynomial time, a path of length $n - n^{\epsilon}$ in an n-vertex Hamiltonian graph for any $\epsilon < 1$. They conjecture that it is as hard even for graphs with bounded degrees. On the other hand, Feder, Motwani, and Subi [4] showed that there is a polynomial time algorithm for finding a cycle of length at least $n^{(\log_3 2)/2}$ in any 3-connected cubic n-vertex graph. They also proposed the problem for 3-connected graphs with bounded degrees. For a graph G, let $\Delta(G)$ denote its maximum degree. Jackson and Wormald [9] proved that every 3-connected n-vertex graph G with $\Delta(G) \leq d$ has a cycle of length at least $\frac{1}{2}n^{\log_b 2} + 1$, where $b = 6d^2$. Recently, Chen, Xu, and Yu [3] gave a cubic algorithm that, given a 3-connected n-vertex graph G with $\Delta(G) \leq d$, finds a cycle of length at least $n^{\log_b 2}$, where $b = 2(d-1)^2 + 1$. It was conjectured in 1993 by Jackson and Wormald [9] that for $d \geq 4$ the right value for b should be $d-1$.

*Received by the editors June 8, 2005; accepted for publication (in revised form) April 17, 2006; published electronically October 3, 2006.

http://www.siam.org/journals/sicomp/36-3/63326.html

†Department of Math and Statistics, Georgia State University, Atlanta, GA 30303 and Faculty of Math and Statistics, Huazhong Normal University, Wuhan, China (gchen@gsu.edu). The work of this author was partially supported by NSA grant H98230-04-1-0300 and NSF grant DMS-0500951.

‡School of Math and Statistics, Carleton University, Ottawa, Ontario, K1S 5B6, and Center for Combinatorics, LPMC, Nankai University, Tianjin, 300071, China (zgao@math.carleton.ca). The work of this author was partially supported by NSERC and RGC grant HKU7056/04P.

§School of Mathematics, Georgia Institute of Technology, Atlanta, GA 30332 and Center for Combinatorics, LPMC, Nankai University, Tianjin, 300071, China (yu@math.gatech.edu). The work of this author was partially supported by NSF grant DMS-0245530, NSA grant MDA904-03-1-0052, and RGC grant HKU7056/04P.

¶Department of Mathematics, University of Hong Kong, Hong Kong, China (wzang@maths.hku.hk). The work of this author was partially supported by RGC grant HKU7056/04P.

The main result of this paper shows that this conjecture holds for a linear function b of d. (This result appears in the extended abstract [2].)

THEOREM 1.1. *Let $n \geq 4$ and $d \geq 4$ be integers. Let G be a 3-connected graph with n vertices and $\Delta(G) \leq d$. Then G contains a cycle of length at least $\frac{1}{2}n^{\log_b 2}+3$, where $b = \max\{64, 4d+1\}$.*

For 3-connected graphs, this improves the above-mentioned result of Feder and Motwani [5]. Our proof of Theorem 1.1 implies a cubic algorithm for finding a cycle of length at least $\frac{1}{2}n^{\log_b 2} + 3$. The multiplicative constant $1/2$ and the additive constant 3 are for induction purpose. As in [3], we prove the following three statements simultaneously.

THEOREM 1.2. *Let $n \geq 5$ and $d \geq 4$ be integers, let $b = \max\{64, 4d+1\}$ and $r = \log_b 2$, and let G be a 3-connected graph with n vertices. Then the following statements hold.*

(a) *Let $xy \in E(G)$ and $z \in V(G)-\{x,y\}$, and let t denote the number of neighbors of z distinct from x and y. Assume $\Delta(G) \leq d+1$, and that every vertex of degree $d+1$ (if any) is incident with edge zx or zy. Then there is a cycle C through xy in $G-z$ such that $|C| \geq \frac{1}{2}(\frac{(d-1)n}{dt})^r + 2$.*

(b) *Suppose $\Delta(G) \leq d$. Then for any distinct $e, f \in E(G)$, there is a cycle C through e and f in G such that $|C| \geq \frac{1}{2}(\frac{n}{d})^r + 3$.*

(c) *Suppose $\Delta(G) \leq d$. Then for any $e \in E(G)$, there is a cycle C through e in G such that $|C| \geq \frac{1}{2}n^r + 3$.*

Note the degree condition in (a): zx and zy need not be edges of G, but if x (respectively, y) has degree $d+1$, then zx (respectively, zy) must be an edge of G, and if z has degree $d+1$, then zx or zy must be an edge of G. This condition is due to the addition of edges in order to maintain 3-connectivity.

When $n \geq 5$, Theorem 1.2(c) clearly implies Theorem 1.1. When $n = 4$, Theorem 1.1 is obvious. The next result says that Theorem 1.2 holds for graphs with bounded size, which will enable us to avoid dealing with small graphs in inductive proofs. We omit its proof, since it is rather straightforward.

LEMMA 1.3. *Let G, n, d, b, r be the same as in Theorem* 1.2. *If $n \leq 4d+1$, then Theorem* 1.2(a) *and* (b) *hold, and if $n \leq (4d+1)^2$, then Theorem* 1.2(c) *holds.*

To prove Theorem 1.2, we need to deal with graphs obtained from 3-connected graphs by deleting a vertex (such as $G-z$ in (a)); such graphs need not be 3-connected. By using a result of Tutte [11] and an algorithm of Hopcroft and Tarjan [7], we can decompose such graphs into "3-connected components." We then find long paths through certain 3-connected components and use properties of the function $x^{\log_b 2}$ to account for the unused 3-connected components. (For a brief outline of our approach, the reader is referred to the algorithm in section 6.) Our approach is similar to that in [3], but here we prove stronger properties of the function $x^{\log_b 2}$ and analyze the 3-connected components in a more sophisticated way.

We organize this paper as follows. In section 2, we recall notation of Hopcroft and Tarjan [7] concerning the decomposition result of Tutte [11] of 2-connected graphs into 3-connected components. We then define cycle chains of 3-connected components, and prove several results on paths in cycle chains. We prove in section 3 several useful properties of the function $f(x) = x^{\log_b 2}$. We also define block chains of 3-connected components, and prove lemmas concerning paths in block chains. Theorem 1.2 will be shown inductively. So in sections 4 and 5, we show how to reduce Theorem 1.2 to smaller graphs. In section 6, we complete the proof of our main result, and outline a cubic algorithm for finding a long cycle in a 3-connected graph with bounded degree.

For graphs G and H, we use $G \cong H$ (respectively, $G \ncong H$) to mean that G is isomorphic to (respectively, not isomorphic to) H. Let G be a graph, H a subgraph of G, and $S := \{v_1, \ldots, v_k, x_1y_1, \ldots, x_py_p\}$, where v_i, x_j, y_j are vertices of G and $\{x_1, y_1, \ldots, x_p, y_p\} \subseteq \{v_1, \ldots, v_k\} \cup V(H)$. Then $H + S$ denotes the simple graph with $V(H + S) := V(H) \cup \{v_1, \ldots, v_k\}$ and $E(H + S) = E(H) \cup \{x_1y_1, \ldots, x_py_p\}$.

2. Paths in cycle chains. For convenience, we recall the decomposition of a 2-connected graph into 3-connected components. A detailed description can be found in [3] and [7].

Let G be a 2-connected graph. We allow multiple edges for the description of this decomposition. Then, $E(G)$ in this section is treated as a multi-set. We say that $\{a, b\} \subseteq V(G)$ is a *separation pair* in G if there are subgraphs G_1, G_2 of G such that $G_1 \cup G_2 = G$, $V(G_1 \cap G_2) = \{a, b\}$, $E(G_1 \cap G_2) = \emptyset$, and $|E(G_i)| \geq 2$ for $i = 1, 2$. Let $G_i' := (V(G_i), E(G_i) \cup \{ab\})$ for $i = 1, 2$. Then G_1' and G_2' are called *split graphs* of G with respect to the separation pair $\{a, b\}$, and the new edge ab added to G_i is called a *virtual* edge. It is easy to see that, since G is 2-connected, G_i' is 2-connected or G_i' consists of two vertices and at least three multiple edges between them.

Suppose a multigraph is split, and the split graphs are split, and so on, until no more splits are possible. Then each remaining graph is called a *split component*. No split component contains a separation pair and, therefore, each split component must be one of the following: a triangle, a *triple bond* (two vertices and three multiple edges between them), or a 3-connected graph.

It is not hard to see that if a split component of a 2-connected graph is 3-connected, then it is uniquely determined. It is also easy to see that, for any two split components G_1, G_2 of a 2-connected graph, we have $|V(G_1 \cap G_2)| \leq 2$, and if $|V(G_1 \cap G_2)| = 2$, then either G_1 and G_2 share a virtual edge between the vertices in $V(G_1 \cap G_2)$ or there is a sequence of triple bonds such that the first shares a virtual edge with G_1, any two consecutive triple bonds in the sequence share a virtual edge, and the last triple bond shares a virtual edge with G_2.

In order to make such decomposition unique, some triple bonds and triangles need to be merged. Let $G_i' = (V_i', E_i')$, $i = 1, 2$, be two split components, both containing a virtual edge ab. Let $G' = (V_1' \cup V_2', (E_1' - \{ab\}) \cup (E_2' - \{ab\}))$. The graph G' is called the *merge graph* of G_1' and G_2'. Clearly, a merge of triple bonds gives a graph consisting of two vertices and multiple edges, which is called a *bond*. Also a merge of triangles gives a cycle, and a merge of cycles gives a cycle as well.

Let $\mathcal{D}$ denote the set of those 3-connected split components of a 2-connected graph G. We merge the split components of G not in $\mathcal{D}$ as follows: the bonds are merged as much as possible to give a set of bonds $\mathcal{B}$, and the cycles are merged as much as possible to give a set of cycles $\mathcal{C}$. Then $\mathcal{B} \cup \mathcal{C} \cup \mathcal{D}$ is the set of the 3-*connected components* of G. Note that any two 3-connected components either are edge disjoint or share exactly one virtual edge. The following theorem is a combination of a result of Tutte [11] and an algorithm of Hopcroft and Tarjan [7].

THEOREM 2.1. *The* 3*-connected components of any* 2*-connected graph are unique and can be found in* $O(E)$ *time.*

If we define a graph whose vertices are the 3-connected components of G and, where two vertices are adjacent whenever the corresponding 3-connected components share a virtual edge, then this graph is a tree, which we call the *block-bond tree* of G. For convenience, 3-connected components that are not bonds are called 3-*blocks*. An *extreme* 3-block is a 3-block that contains at most one virtual edge. That is, either it is the only 3-connected component (in which case G is 3-connected), or it corresponds

to a degree one vertex in the block-bond tree.

A *cycle chain* in a 2-connected graph G is a sequence $C_1C_2\dots C_k$ of 3-blocks of G such that each C_i is a cycle and there exist bonds (possibly empty) $B_1, B_2, \dots, B_{k-1}$ in G such that $C_1B_1C_2B_2\dots B_{k-1}C_k$ is a path in the block-bond tree of G. For convenience, we sometimes write $H := C_1\dots C_k$ for a cycle chain, and view H as the simple graph obtained from the union of C_i $(1 \le i \le k)$ by identifying virtual edges between the vertices of $C_i \cap C_{i+1}$ $(1 \le i \le k-1)$. The following is a direct consequence of the definition of a cycle chain.

PROPOSITION 2.2. *Let G be a 2-connected graph and $H := C_1\dots C_k$ be a cycle chain in G. Then deleting all edges of H with both ends in $V(C_i \cap C_{i+1})$, $1 \le i \le k-1$, results in a cycle.*

The next result finds a path linking two edges in a cycle chain.

PROPOSITION 2.3. *Let G be a 2-connected graph, let $H := C_1\dots C_k$ be a cycle chain in G, let $uv \in E(C_1)$ with $\{u,v\} \ne V(C_1 \cap C_2)$ when $k \ne 1$, and let $ab \in E(C_k)$ with $\{a,b\} \ne V(C_{k-1} \cap C_k)$ when $k \ne 1$. Then there is a path in $H - \{v, ab\}$ from u to $\{a,b\}$ and containing $V(\bigcup_{i=1}^{k-1}(C_i \cap C_{i+1})) - (\{a,b\} \cup \{u,v\})$.*

Proof. We apply induction on k. The result holds trivially for $k = 1$. So assume $k \ge 2$. Let $H' := C_2\dots C_k$ and $V(C_1 \cap C_2) = \{u_1, v_1\}$. Without loss of generality, we may assume that $C_1 - \{v, v_1\}$ contains a path P from u to u_1. Suppose $v_1 = v$. By induction, we find a path Q in $H' - \{v_1, ab\}$ from u_1 to $\{a,b\}$ and containing $V(\bigcup_{i=2}^{k-1}(C_i \cap C_{i+1})) - (\{a,b\} \cup \{u_1, v_1\})$. Then $P \cup Q$ gives the desired path. Now assume $v_1 \ne v$. By induction, we find a path Q' in $H' - \{u_1, ab\}$ from v_1 to $\{a,b\}$ and containing $V(\bigcup_{i=2}^{k-1}(C_i \cap C_{i+1})) - (\{a,b\} \cup \{u_1, v_1\})$. Now $(P \cup Q') + u_1v_1$ gives the desired path. □

Remark. The path, say R, found in Proposition 2.3 may use edges between the vertices of $C_i \cap C_{i+1}$ $(1 \le i \le k-1)$. However, either G has an edge between the vertices of $C_i \cap C_{i+1}$, or $C_i \cap C_{i+1}$ is contained in a 3-block of G not in $\mathcal{H}$. Hence, from R we can produce a path in G by replacing virtual edges in R with appropriate paths in G, and this new path is at least as long as R. This observation applies to the next three results as well, and will be frequently used.

A similar argument establishes the following result, which finds a path in a cycle chain between two vertices and avoiding a specific vertex.

PROPOSITION 2.4. *Let G be a 2-connected graph, let $H := C_1\dots C_k$ be a cycle chain in G, let $uv \in E(C_1)$ with $\{u,v\} \ne V(C_1 \cap C_2)$ when $k \ne 1$, and let $x \in V(C_k)$ with $x \ne v$ when $k = 1$ and $x \notin V(C_{k-1} \cap C_k)$ when $k \ne 1$. Then there is a path in $H - v$ from u to x and containing $V(\bigcup_{i=1}^{k-1}(C_i \cap C_{i+1})) - \{v\}$.*

It is clear that the paths and cycle in the above three propositions can be found in $O(V)$ time. The following two results are Propositions 2.7 and 2.8 in [3], which find in $O(V)$ time paths through a given edge in a cycle chain.

PROPOSITION 2.5. *Let G be a 2-connected graph, let $H := C_1\dots C_k$ be a cycle chain in G, let $uv \in E(C_1)$ with $\{u,v\} \ne V(C_1 \cap C_2)$ when $k \ne 1$, $ab \in E(C_k)$ with $\{a,b\} \ne V(C_{k-1} \cap C_k)$ when $k \ne 1$, and $cd \in E(\bigcup_{i=1}^{k} C_i) - \{ab\}$. Suppose $ab \ne uv$ when $k = 1$. Then there is a path P in $H - ab$ from $\{a,b\}$ to $\{c,d\}$ such that $uv \in E(P)$, $cd \notin E(P)$ unless $cd = uv$, and $V(\bigcup_{i=1}^{k-1}(C_i \cap C_{i+1})) \subseteq V(P)$.*

PROPOSITION 2.6. *Let G be a 2-connected graph, let $H := C_1\dots C_k$ be a cycle chain in G, let $uv \in E(C_1)$ with $\{u,v\} \ne V(C_1) \cap V(C_2)$ when $k \ne 1$, $x \in V(C_k)$ with $x \notin V(C_{k-1} \cap C_k)$ when $k \ne 1$, and $cd \in E(\bigcup_{i=1}^{k} C_i)$. Then there is a path P in H from x to $\{c,d\}$ such that $uv \in E(P)$, $cd \notin E(P)$ unless $cd = uv$, and*

$V(\bigcup_{i=1}^{k-1}(C_i \cap C_{i+1})) \subseteq V(P)$.

We conclude this section by recalling from [3] two graph operations and three lemmas. Let G be a graph and let e, f be distinct edges of G. An *H-transform* of G *at* $\{e, f\}$ is an operation that subdivides e and f by vertices x and y, respectively, and then adds the edge xy. Let $x \in V(G)$ such that x is not incident with e. A *T-transform* of G *at* $\{x, e\}$ is an operation that subdivides e with a vertex y and then adds the edge xy. If there is no need to specify e, f, x, we simply speak of an H-transform or a T-transform. The following result is Lemma 3.3 in [3].

LEMMA 2.7. *Let $d \geq 3$ be an integer and let G be a 3-connected graph with $\Delta(G) \leq d$. Let G' be a graph obtained from G by an H-transform or a T-transform. Then G' is a 3-connected graph, the vertex of G involved in the T-transform has degree at most $d+1$, and all other vertices of G' have degree at most d.*

The next two results are Lemmas 3.6 and 3.7 in [3], where it is shown that the path P can be found in $O(V)$ time.

LEMMA 2.8. *Let G be a 3-connected graph, let $f \in E(G)$, let $ab, cd, vw \in E(G) - \{f\}$, and assume that $\{c, d\} \neq \{v, w\}$. Then there exists a path P in G from $\{a, b\}$ to some $z \in \{c, d\} \cup \{v, w\}$ such that* (i) *$f \in E(P)$,* (ii) *$cd \in E(P)$ or $vw \in E(P)$,* (iii) *if $cd \in E(P)$, then $z \in \{v, w\}$ and $vw \notin E(P)$, and* (iv) *if $vw \in E(P)$, then $z \in \{c, d\}$ and $cd \notin E(P)$.*

LEMMA 2.9. *Let G be a 3-connected graph, let $f \in E(G)$, let $x \in V(G)$ such that x is not incident with f, let $cd, vw \in E(G) - \{f\}$, and assume that $\{c, d\} \neq \{v, w\}$. Then there exists a path P in G from x to some $z \in \{c, d\} \cup \{v, w\}$ such that* (i) *$f \in E(P)$,* (ii) *$cd \in E(P)$ or $vw \in E(P)$,* (iii) *if $cd \in E(P)$, then $z \in \{v, w\}$ and $vw \notin E(P)$, and* (iv) *if $vw \in E(P)$, then $z \in \{c, d\}$ and $cd \notin E(P)$.*

3. Paths in block chains. We first prove four lemmas concerning the function $x^{\log_b 2}$. These lemmas will then be used to find long paths in block chains. First, we recall Lemma 3.1 in [3].

LEMMA 3.1. *Let $b \geq 4$ be an integer, and let $m \geq n$ be positive integers. Then $m^{\log_b 2} + n^{\log_b 2} \geq (m + (b-1)n)^{\log_b 2}$.*

When m is sufficiently larger than n, we have the following result.

LEMMA 3.2. *Let $b \geq 9$ be an integer, let m and n be positive integers, and assume $m \geq \frac{b(b-1)}{4}n$. Then $m^{\log_b 2} + n^{\log_b 2} \geq (m + \frac{b(b-1)}{4}n)^{\log_b 2}$.*

Proof. By dividing $m^{\log_b 2}$ to the above inequality, we see what we need to prove is equivalent to the statement: for any $0 \leq s \leq \frac{4}{b(b-1)}$, $1 + s^{\log_b 2} \geq (1 + \frac{b(b-1)}{4}s)^{\log_b 2}$.

Let $f(s) = 1 + s^{\log_b 2} - (1 + \frac{b(b-1)}{4}s)^{\log_b 2}$. Clearly, $f(0) = 0$. Note that $b(b-1) > 4(b-1)$ when $b \geq 5$. Hence $f(1) = 2 - (1 + \frac{b(b-1)}{4})^{\log_b 2} < 2 - b^{\log_b 2} = 0$. Taking derivative about s, we have $f'(s) = (\log_b 2)(s^{(\log_b 2)-1} - \frac{b(b-1)}{4}(1 + \frac{b(b-1)}{4}s)^{(\log_b 2)-1})$. A simple calculation shows that $f'(s) = 0$ has a unique solution. Therefore, if $f(c) > 0$ for some $0 < c < 1$, then $f(s) \geq 0$ for all $0 \leq s \leq c$.

Note that $0 < \frac{4}{b(b-1)} < 1$ and $f(\frac{4}{b(b-1)}) > 1 + (\frac{1}{b^2})^{\log_b 2} - 2^{\log_b 2} \geq 1.25 - 2^{\log_9 2} = 0.005587\ldots > 0$. Therefore, we have $f(s) \geq 0$ for all $s \in [0, \frac{4}{b(b-1)}]$. ∎

When m is not sufficiently larger than n, we have the following complementary result.

LEMMA 3.3. *Let $b \geq 64$ be an integer, let $m \geq n$ be positive integers, and assume $m \leq \frac{b(b-1)}{4}n$. Then $m^{\log_b 2} + n^{\log_b 2} \geq (4m)^{\log_b 2}$.*

Proof. The statement of Lemma 3.3 is equivalent to $1 + s^{\log_b 2} \geq 4^{\log_b 2}$ for all $\frac{4}{b(b-1)} \leq s \leq 1$. Therefore, it suffices to show $1 + (\frac{4}{b(b-1)})^{\log_b 2} \geq 4^{\log_b 2}$. This is true

because $1+(\frac{4}{b(b-1)})^{\log_b 2} \geq 1+(\frac{4}{b^2})^{\log_b 2} = 1+\frac{4^{\log_b 2}}{4} > 4^{\log_b 2}$ (since $b \geq 64$). □

We shall also use the following observations in the proof of Theorem 1.2.

LEMMA 3.4. *Let m be an integer, $d \geq 3$, and $b \geq d+1$. If $m \geq 4$, then $m \geq \frac{1}{2}m^{\log_b 2}+3$. If $m \geq 3$, then $m > \frac{1}{2}(\frac{m}{d})^{\log_b 2}+2$. If $m \geq 2$, then $m > \frac{1}{2}(\frac{m}{d})^{\log_b 2}+1$.*

Proof. Let $f(x) = x - \frac{1}{2}x^{\log_b 2}$. We can show that $f'(x) > 0$ for $x \geq 1$. Hence $f(x)$ is an increasing function when $x \geq 1$. Thus, when $x \geq 4$, we have $f(x) \geq f(4) = 4 - \frac{1}{2}4^{\log_b 2} \geq 3$ (since $b \geq 4$). The first inequality holds.

Let $f(x) = x - \frac{1}{2}(\frac{x}{d})^{\log_b 2}$; then $f(x)$ is increasing when $x \geq 1$. The second inequality follows from $f(3) > 2$, and the third inequality follows from $f(2) > 1$. □

We now turn to paths in block chains. Let G be a 2-connected graph. A *block chain* in G is a sequence $H_1 \dots H_h$ for which (1) each H_i is either a cycle chain in G or a 3-connected 3-block of G, (2) for any $1 \leq s \leq h-1$, H_s or H_{s+1} is 3-connected, and (3) there exist bonds (possibly empty) $B_1, \dots, B_{h-1}$ such that $H_1B_1H_2B_2 \dots B_{h-1}H_h$ form a path in the block-bond tree of G (by also including the tree paths corresponding to H_i when H_i is a cycle chain). A detailed description with examples can be found in [3]. For convenience, we sometimes write $\mathcal{H} := H_1 \dots H_h$ for a block chain and view $\mathcal{H}$ as the simple graph obtained from $\bigcup_{i=1}^{n} H_i$ by identifying edges between the vertices in $H_i \cap H_{i+1}$ $(1 \leq i \leq n-1)$. The edges of $\mathcal{H}$ between the vertices of $H_i \cap H_{i+1}$ are called *separating edges* of $\mathcal{H}$. Such edges are to be avoided when we find paths in block chains.

Let $H_1 \dots H_h$ be a block chain and let $V(H_s \cap H_{s+1}) = \{x_s, y_s\}$, $1 \leq s \leq h-1$. For each $1 \leq s \leq h$, we define $A(H_s)$ as follows. If H_s is 3-connected, then $A(H_s) := V(H_s)$. If $H_s = C_1 \dots C_k$ is a cycle chain, then let

- $A(H_s) := V(\bigcup_{i=1}^{k-1}(C_i \cap C_{i+1})) - (\{x_{s-1}, y_{s-1}\} \cup \{x_s, y_s\})$ when $1 < s < h$,
- $A(H_s) := V(\bigcup_{i=1}^{k-1} C_i \cap C_{i+1})$ when $s = 1 = h$, $A(H_s) := V(\bigcup_{i=1}^{k-1}(C_i \cap C_{i+1})) - \{x_s, y_s\}$ when $s = 1 < h$, and
- $A(H_s) := V(\bigcup_{i=1}^{k-1}(C_i \cap C_{i+1})) - \{x_{s-1}, y_{s-1}\}$ when $1 < s = h$.

We write $\sigma(\mathcal{H}) := \sum_{s=1}^{h} |A(H_s)|$ and $|\mathcal{H}| := |V(\bigcup_{i=1}^{h} H_i)|$. For convenience, we define $B_1(\mathcal{H}) = \{H_i : H_i \text{ is 3-connected or } |A(H_i)| \leq 1\}$ and $B_2(\mathcal{H}) = \{H_i : H_i \text{ is a cycle chain and } |A(H_i)| \geq 2\}$.

In the remainder of this section, we show how to find long paths in block chains (in terms of $\sigma(\mathcal{H})$). All proofs imply $O(V)$ algorithms that reduce the problem of finding a path to Theorem 1.2 for smaller graphs.

LEMMA 3.5. *Let $n \geq 6$ be an integer and assume Theorem 1.2 holds for graphs with at most $n-1$ vertices. Let $\mathcal{H} := H_1H_2 \dots H_h$ be a block chain in a 2-connected graph such that $|\mathcal{H}| < n$ and $\Delta(H_i) \leq d$ for $1 \leq i \leq h$. Let $uv \in E(H_1)$ such that $\{u, v\}$ is not a cut of H_1, and if $h \geq 2$, then $\{u, v\} \neq V(H_1 \cap H_2)$. Then there is a path P in $\mathcal{H}$ from u to v such that $|E(P)| \geq \frac{1}{2}(\frac{(d-1)\sigma(\mathcal{H})}{d})^r + 2$ and P contains no separating edge of $\mathcal{H}$.*

Proof. When $h \geq 2$, we use a, b to denote the vertices in $V(H_1 \cap H_2)$. Suppose $|A(H_1)| \geq \frac{(d-1)\sigma(\mathcal{H})}{d}$. First assume H_1 is a cycle chain or $H_1 \cong K_4$. Then there is a Hamilton path P_1 in H_1 from u to v (by Proposition 2.2 when H_1 is a cycle chain). If $|H_1| = 3$, then $|A(H_1)| = 0$, and hence, $|E(P_1)| \geq \frac{1}{2}|A(H_1)|^r + 2$. If $|H_1| \geq 4$, then $|E(P_1)| \geq 3$, and by Lemma 3.4, $|E(P_1)| \geq \frac{1}{2}|H_1|^r + 2 \geq \frac{1}{2}|A(H_1)|^r + 2$. Now assume H_1 is 3-connected and $H_1 \ncong K_4$. Then by Theorem 1.2(c), H_1 has a cycle C_1 through uv such that $|E(C_1)| \geq \frac{1}{2}|H_1|^r + 3 = \frac{1}{2}|A(H_1)|^r + 3$. Let $P_1 := C_1 - uv$. If $h = 1$ or $ab \notin E(P_1)$, then $P := P_1$ gives the desired path. If $h \geq 2$ and $ab \in E(P_1)$, then by replacing ab with a path in $H_2 \dots H_h$ between a and b and not containing

any separating edge of $\mathcal{H}$, we obtain the desired path P.

So we may assume $|A(H_1)| < \frac{(d-1)\sigma(\mathcal{H})}{d}$. In particular, $h \geq 2$. If H_1 is a cycle chain or $H_1 \cong K_4$, then, as in the above paragraph, we find a Hamilton path P_1 from u to v in H_1 through ab such that $|E(P_1)| \geq \frac{1}{2}|A(H_1)|^r + 2$. Now assume H_1 is 3-connected and $H_1 \ncong K_4$. Then by Theorem 1.2(b), H_1 has a cycle C_1 through uv and ab such that $|E(C_1)| \geq \frac{1}{2}(\frac{|A(H_1)|}{d})^r + 3$; let $P_1 := C_1 - uv$.

By induction, we find a path P' in $\mathcal{H}' := H_2 \dots H_h$ from a to b and containing no separating edges of $\mathcal{H}'$ such that $|E(P')| \geq \frac{1}{2}(\frac{(d-1)\sigma(\mathcal{H}')}{d})^r + 2$. Let $P := (P_1 - ab) \cup P'$. Since $\sigma(\mathcal{H}) \leq A(H_1) + \sigma(\mathcal{H}')$ and $|A(H_1)| < \frac{(d-1)\sigma(\mathcal{H})}{d}$, $\frac{|A(H_1)|}{d} < \frac{(d-1)\sigma(\mathcal{H}')}{d}$. Hence by Lemma 3.2,

$$\begin{aligned} |E(P)| &> \frac{1}{2}\left(\frac{|A(H_1)|}{d}\right)^r + \frac{1}{2}\left(\frac{(d-1)\sigma(\mathcal{H}')}{d}\right)^r + 2 \\ &\geq \frac{1}{2}\left((b-1)\frac{|A(H_1)|}{d} + \frac{(d-1)\sigma(\mathcal{H}')}{d}\right)^r + 2 \\ &> \frac{1}{2}\left(\frac{(d-1)\sigma(\mathcal{H})}{d}\right)^r + 2. \end{aligned}$$

So P gives the desired path. □

For the next two lemmas, we define uv and x in a block chain $\mathcal{H} := H_0H_1 \dots H_h$ (in a 2-connected graph). Suppose $h = 0$. If H_0 is 3-connected or H_0 is a cycle, then let $uv \in E(H_0)$ and $x \in V(H_0) - \{u, v\}$, and if $H_0 = C_1 \dots C_k$ is a cycle chain with $k \geq 2$, then let $uv \in E(C_1)$ with $\{u, v\} \neq V(C_1 \cap C_2)$ and let $x \in V(C_k) - V(C_{k-1})$. Now assume $h \geq 1$. If H_0 is 3-connected or H_0 is a cycle, then let $uv \in E(H_0)$ with $\{u, v\} \neq V(H_0 \cap H_1)$; if $H_0 = C_1 \dots C_k$ is a cycle chain with $k \geq 2$ and $V(H_0 \cap H_1) = V(C_k \cap H_1)$, then let $uv \in E(C_1)$ with $\{u, v\} \neq V(C_1 \cap C_2)$; if H_h is a cycle or H_h is 3-connected, then let $x \in V(H_h) - V(H_{h-1})$; and if $H_h = C_1 \dots C_k$ is a cycle chain with $k \geq 2$ and $V(H_{h-1} \cap H_h) = V(H_{h-1} \cap C_1)$, then let $x \in V(C_k) - V(C_{k-1})$.

LEMMA 3.6. *Let $n \geq 6$ be an integer and assume Theorem 1.2 holds for graphs with at most $n-1$ vertices. Let $\mathcal{H} := H_0H_1 \dots H_h$, uv, x be defined as above, and assume $|\mathcal{H}| < n$, $\Delta(H_i) \leq d$ for $0 \leq i \leq h$, and the degree of x in H_h is at most $d-1$. Then there exists a path P in $\mathcal{H} - v$ from u to x and containing no separating edge of $\mathcal{H}$ such that*

(i) $|E(P)| \geq \frac{1}{2}(\sum_{i=0}^{h}(\frac{|A(H_i)|}{d})^r) + 1 \geq \frac{1}{2}(\frac{\sigma(\mathcal{H})}{d})^r + 1$, *and*

(ii) $|E(P)| \geq \frac{1}{2}(\sum\{(\frac{|A(H_i)|}{d})^r : H_i \in B_1(\mathcal{H})\}) + (\sum\{\max\{1, |A(H_i)| - 2\} : H_i \in B_2(\mathcal{H})\}) + 1$.

Proof. We apply induction on h. Suppose $h = 0$. If H_0 is 3-connected and $H_0 \ncong K_4$, then by assumption and because x has degree at most $d-1$, Theorem 1.2(a) holds for $H_0 + \{vx, ux\}$. Hence, $H_0 - v$ contains a path P from u to x such that $|E(P)| \geq \frac{1}{2}(\frac{|A(H_0)|}{d})^r + 1$. If $H_0 \cong K_4$, then we can find a path P from u to x in $H_0 - v$ such that $|E(P)| = 2 \geq \frac{1}{2}(\frac{|A(H_0)|}{d})^r + 1$. If H_0 is a cycle chain, then by Proposition 2.4, there is a path P from u to x in $H_0 - v$ containing $A(H_0) - \{v\}$. Note that $x \notin A(H_0)$ and if $v \in A(H_0)$, then $u \notin A(H_0)$. Thus, $|E(P)| \geq |A(H_0)|$. Because $|E(P)| \geq 1$ and since $|A(H_0)| = 0$ or $|A(H_0)| \geq 2$, we have $|E(P)| \geq \frac{1}{2}(\frac{|A(H_0)|}{d})^r + 1$ (by Lemma 3.4). Clearly, $|E(P)| \geq \max\{1, |A(H_0)| - 2\} + 1$ when $H_0 \in B_2(\mathcal{H})$.

Now assume $h \geq 1$. Let $V(H_0 \cap H_1) = \{u_0, v_0\}$, and assume the notation is chosen so that $u_0 \notin \{u, v\}$. By the above argument for $h = 0$, if H_0 is a cycle chain or $H_0 \cong K_4$, then $H_0 - v$ has a path P_0 from u to u_0 such that $|E(P_0)| \geq \frac{1}{2}(\frac{|A(H_0)|}{d})^r + 1$,

and $|E(P_0)| \geq \max\{1, |A(H_0)| - 2\} + 1$ when $H_0 \in B_2(\mathcal{H})$. (Note in the case H_0 is a cycle chain, $u_0 \notin A(H_0)$ because $h \geq 1$.) Now assume H_0 is 3-connected and $|H_0| \geq 5$. If $v = v_0$, then we apply Theorem 1.2(a) to find a path P_0 from u to u_0 in $(H_0 + uu_0) - v$ such that $|E(P_0)| \geq \frac{1}{2}(\frac{|A(H_0)|}{d})^r + 1$. If $v \neq v_0$, then let H_0' be obtained from H_0 by a T-transform at $\{v, u_0v_0\}$ and let u' denote the new vertex. By Theorem 1.2(a), we find a path P_0^* in $(H_0' + uu') - v$ from u to u' such that $|E(P_0^*)| \geq \frac{1}{2}(\frac{|A(H_0)|}{d})^r + 1$; and let $P_0 := P_0^* - u'$ (in this case $u_0v_0 \notin E(P_0)$).

Let $P_0' := P_0$ if $u_0v_0 \notin E(P_0)$; otherwise, let $P_0' := P_0 - u_0$. Then P_0' is a path in $H_0 - \{v, u_0v_0\}$ from u to $\{u_0, v_0\}$ such that $|E(P_0')| \geq \frac{1}{2}(\frac{|A(H_0)|}{d})^r$, and $|E(P_0')| \geq \max\{1, |A(H_0)| - 2\}$ when $H_0 \in B_2(\mathcal{H})$. Without loss of generality, we may assume that P_0' is from u_0 to u.

By applying induction to $\mathcal{H}' := H_1 \dots H_h$, there is a path P_1 from u_0 to x in $\mathcal{H}' - v_0$ containing no separating edge of $\mathcal{H}'$ such that $|E(P_1)| \geq \frac{1}{2}(\sum_{i=1}^{h}(\frac{|A(H_i)|}{d})^r) + 1 \geq \frac{1}{2}(\frac{\sigma(\mathcal{H}')}{d})^r + 1$ and $|E(P_1)| \geq \frac{1}{2}(\sum\{(\frac{|A(H_i)|}{d})^r : H_i \in B_1(\mathcal{H}) \text{ and } i \neq 0\}) + (\sum\{\max\{1, |A(H_i)| - 2\} : H_i \in B_2(\mathcal{H}) \text{ and } i \neq 0\}) + 1$.

Let $P := P_0' \cup P_1$. Because $h \geq 1$, H_0 or H_1 is not a cycle chain, and hence, $\sigma(\mathcal{H}) \leq |A(H_0)| + \sigma(\mathcal{H}')$. It is easy to see that P satisfies (i) and (ii). Note that the second inequality in (i) follows from the first in (i) by applying Lemma 3.1. □

LEMMA 3.7. *Assume the same hypothesis of Lemma* 3.6. *Then for any* $0 \leq t \leq h$ *and for any* $pq \in E(H_t)$ *such that* $|H_t| \leq n - 3$ *when* $h \geq 1$, *there exists a path* P *in* $\mathcal{H}$ *from* x *to* $\{p, q\}$ *and containing no separating edge of* $\mathcal{H}$ *such that*

(i) $pq \notin E(P)$, *and* $|E(P)| \geq \frac{1}{2}|A(H_0)|^r + \frac{1}{2}(\sum\{(\frac{|A(H_i)|}{d})^r : H_i \in B_1(\mathcal{H})$ *and* $i \neq 0\}) + (\sum\{\max\{1, |A(H_i)| - 2\} : H_i \in B_2(\mathcal{H})$ *and* $i \neq 0\}) + 1$, *and*

(ii) *if we require* $uv \in E(P)$, *then* $pq \notin E(P)$ *unless* $pq = uv$, *and* $|E(P)| \geq \frac{1}{2}(\sum\{(\frac{|A(H_i)|}{d})^r : H_i \in B_1(\mathcal{H})\}) + (\sum\{\max\{1, |A(H_i)| - 2\} : H_i \in B_2(\mathcal{H})\}) + 1 \geq \frac{1}{2}(\frac{\sigma(\mathcal{H})}{d})^r + 1$.

Proof. We apply induction on h. Note that the second inequality in (ii) follows from the first in (ii) by applying Lemma 3.1.

Case 1. $h = 0$.

First, assume H_0 is a cycle chain. Then by Proposition 2.6, there is a path P from x to $\{p, q\}$ in H_0 such that $uv \in E(P)$, $pq \notin E(P)$ unless $pq = uv$, and $A(H_0) \subseteq V(P)$. Because $x \notin A(H_0)$, $|E(P)| \geq |A(H_0)|$. Because $x \notin \{u, v\}$, $|E(P)| \geq 2$. So $|E(P)| \geq \max\{1, |A(H_0)| - 2\} + 1$. Moreover, if $|A(H_0)| \leq 3$, then $|E(P)| \geq 2 > \frac{1}{2}|A(H_0)|^r + 1$, and if $|A(H_0)| \geq 4$, then by Lemma 3.4 we have $|E(P)| \geq |A(H_0)| \geq \frac{1}{2}|A(H_0)|^r + 3$. Clearly (i) and (ii) hold.

Now assume $H_0 \cong K_4$. Let P denote a Hamilton path in H_0 from x to $\{p, q\}$ such that $uv \in E(P)$, and $pq \notin E(P)$ unless $pq = uv$. Then $|E(P)| = 3 > \frac{1}{2}|A(H_0)|^r + 1$ and (i) and (ii) hold.

Finally, assume H_0 is 3-connected and $H_0 \ncong K_4$. Then $5 \leq |H_0| < n$. If $x \in \{p, q\}$, then we apply Theorem 1.2(c) (respectively, Theorem 1.2(b)) to find a cycle C through pq (respectively, pq and uv) such that $|C| \geq \frac{1}{2}|A(H_0)|^r + 3$ (respectively, $|C| \geq \frac{1}{2}(\frac{|A(H_0)|}{d})^r + 3$). Now it is easy to see that (i) and (ii) hold with $P := C - pq$. So assume $x \notin \{p, q\}$. Then let H_0' be obtained from H_0 by a T-transform at $\{x, pq\}$ and let x' denote the new vertex. By Theorem 1.2(c) (respectively, Theorem 1.2(b)), we find a cycle C through xx' (respectively, xx' and uv) such that $|C| \geq \frac{1}{2}|H_0|^r + 3$ (respectively, $|C| \geq \frac{1}{2}(\frac{|H_0|}{d})^r + 3$). Now it is easy to see that (i) and (ii) hold with $P := C - x'$.

Case 2. $h \geq 1$.

Let $\{a, b\} = V(H_0 \cap H_1)$.

Suppose $pq \in \mathcal{H}' := H_1 \dots H_h$. By applying induction to $\mathcal{H}'$ (with ab playing the role of uv), we find a path P' in $\mathcal{H}'$ from x to $\{p, q\}$ and containing no separating edge of $\mathcal{H}'$ such that $ab \in E(P')$, $pq \notin E(P')$ unless $pq = ab$, and $|E(P')| \geq \frac{1}{2}(\sum\{(\frac{|A(H_i)|}{d})^r : H_i \in B_1(\mathcal{H}) \text{ and } i \neq 0\}) + (\sum\{\max\{1, |A(H_i)| - 2\} : H_i \in B_2(\mathcal{H}) \text{ and } i \neq 0\}) + 1$. If H_0 is a cycle chain or $H_0 \cong K_4$, then H_0 has a Hamilton cycle C through ab and uv. If H_0 is 3-connected and $|H_0| \geq 5$, we apply Theorem 1.2(c) (respectively, Theorem 1.2(b)) to find a cycle C through ab (respectively, ab and uv) such that $|C| \geq \frac{1}{2}|H_0|^r + 3$ (respectively, $|C| \geq \frac{1}{2}(\frac{|H_0|}{d})^r + 3$). Then $P := (C - ab) \cup (P' - ab)$ gives the desired path for (i) and (ii).

Therefore, we may assume $pq \in H_0$ and $pq \neq ab$. Let H_0' be obtained from H_0 by an H-transform at $\{pq, ab\}$, and let a', p' denote the new vertices. By Theorem 1.2(c) (respectively, Theorem 1.2(b)) we find a cycle C in H_0' through $a'p'$ (respectively, $a'p'$ and uv) such that $|C| \geq \frac{1}{2}|H_0|^r + 3$ (respectively, $|C| \geq \frac{1}{2}(\frac{|H_0|}{d})^r + 3$). Let $P_0 := C - \{a', p'\}$ and, without loss of generality, let a be the end of P_0. By Lemma 3.6, we can find a path P' in $\mathcal{H}' - b$ from x to a and containing no separating edge of $\mathcal{H}'$ such that $|E(P')| \geq \frac{1}{2}(\sum\{(\frac{|A(H_i)|}{d})^r : H_i \in B_1(\mathcal{H}) \text{ and } i \neq 0\}) + (\sum\{\max\{1, |A(H_i)| - 2\} : H_i \in B_2(\mathcal{H}) \text{ and } i \neq 0\}) + 1$. Now $P := P_0 \cup P'$ gives the desired path, except for (ii) when $pq = uv$. In the exceptional case, we may assume $v \notin \{a, b\}$. Let H_0'' be obtained from H_0 by a T-transform at $\{v, ab\}$, with new vertex a''. We apply Theorem 1.2(a) to find a cycle C in $(H_0'' + ua'') - v$ through ua'' such that $|C| \geq \frac{1}{2}(\frac{|H_0|}{d})^r + 2$. Without loss of generality, we may assume a is the end of $C - a''$. Let P' be found as above. Then $P := ((C - a'') \cup P') + \{v, uv\}$ gives the desired path for (ii). □

4. Cycles through two edges. We reduce Theorem 1.2(a) and (b) to Theorem 1.2 for smaller graphs. Note that finding a long cycle in Theorem 1.2(a) through xy avoiding z is equivalent to finding a long cycle through edges xz and yz. First, we reduce Theorem 1.2(a); our proof implies an $O(E)$ time reduction.

LEMMA 4.1. *Let $n \geq 6$ be an integer, and assume that Theorem* 1.2 *holds for graphs with at most $n - 1$ vertices. Let G be a 3-connected graph with n vertices, let $xy \in E(G)$ and $z \in V(G) - \{x, y\}$, and let t denote the number of neighbors of z distinct from x and y. Assume $\Delta(G) \leq d + 1$, and every vertex of degree $d + 1$ in G (if any) is incident with the edge zx or zy. Then there is a cycle C through xy in $G - z$ such that $|C| \geq \frac{1}{2}(\frac{(d-1)n}{dt})^r + 2$.*

Proof. By Lemma 1.3, we may assume $n \geq 4d + 2$. Since G is 3-connected, $t \geq 1$.

Assume that $G - z$ is 3-connected. By assumption, $\Delta(G - z) \leq d$. Since $n \geq 6$, $|G - z| \geq 5$. So by Theorem 1.2(c), $G - z$ contains a cycle C through xy such that $|C| \geq \frac{1}{2}(n - 1)^r + 3$. By Lemma 3.1, $|C| \geq \frac{1}{2}n^r + 2 > \frac{1}{2}(\frac{(d-1)n}{dt})^r + 2$.

Therefore, we may assume that $G - z$ is not 3-connected. By Theorem 2.1, we decompose $G - z$ into 3-connected components. Let $\mathcal{H} := H_1 \dots H_h$ be a block chain in $G - z$ such that (i) H_h contains an extreme 3-block of $G - z$, (ii) $xy \in E(H_1)$ and $\{x, y\} \neq V(H_1) \cap V(H_2)$ when $h \neq 1$, and if $H_1 = C_1 \dots C_k$ is a cycle chain with $k \geq 2$ and $V(H_1 \cap H_2) = V(C_k \cap H_2)$ (when $h \neq 1$), then $xy \in E(C_1)$ and $\{x, y\} \neq V(C_1 \cap C_2)$, and (iii) subject to (i) and (ii), $\sigma(\mathcal{H})$ is maximum.

We claim that $\sigma(\mathcal{H}) \geq \frac{n-1-2t}{t}$. Since G is 3-connected, each extreme 3-block of $G - z$ distinct from H_1 contains a neighbor of z. Therefore, there are at most $2t$ degree 2 vertices in $G - z$ and at most t extreme 3-blocks of $G - z$ different from H_1. Note that the vertices of $G - z$ with degree at least 3 are counted in $\sigma(\mathcal{K})$ for some

block chain $\mathcal{K}$ (defined as $\mathcal{H}$ above except condition (iii)). It then follows from (iii) that $\sigma(\mathcal{H}) \geq \frac{n-1-2t}{t}$.

Since $n > 4d+1$ and $t \leq d$, $\sigma(\mathcal{H}) \geq 2$. By Lemma 3.5, there is a path P from x to y in $\mathcal{H}$ such that $|E(P)| \geq \frac{1}{2}(\frac{(d-1)\sigma(\mathcal{H})}{d})^r + 2$. Let $C^* := P + xy$. Then

$$\begin{aligned}
|C^*| &= |E(P)| + 1 \\
&\geq \frac{1}{2}\left(\frac{(d-1)\sigma(\mathcal{H})}{d} + (b-1)\right)^r + 2 \quad \text{(by Lemma 3.1)} \\
&\geq \frac{1}{2}\left(\frac{(d-1)(n-1-2t) + dt(b-1)}{dt}\right)^r + 2 \\
&> \frac{1}{2}\left(\frac{(d-1)n}{dt}\right)^r + 2 \quad \text{(since } b \geq 4d+1\text{)}.
\end{aligned}$$

The desired cycle C can now be obtained from C^* by replacing virtual edges in C^* with appropriate paths in G. □

We now reduce Theorem 1.2(b); our proof implies an $O(E)$ time reduction.

LEMMA 4.2. *Let $n \geq 6$ be an integer, and assume that Theorem 1.2 holds for graphs with at most $n-1$ vertices. Suppose G is a 3-connected graph on n vertices and $\Delta(G) \leq d$. Then for any $\{e, f\} \subseteq E(G)$, there is a cycle C through e, f in G such that $|C| \geq \frac{1}{2}(\frac{n}{d})^r + 3$.*

Proof. By Lemma 1.3, we may assume $n \geq 4d+2$. First, assume that e is incident with f. Let $e = xz$ and $f = yz$, and let $G' := G + xy$. Then G' is 3-connected, $\Delta(G') \leq d+1$, and the possible vertices of degree $d+1$ in G' are x and y. By applying Lemma 4.1 to G', xy, z, there is a cycle C' through xy in $G' - z$ such that $|C'| \geq \frac{1}{2}(\frac{(d-1)n}{dt})^r + 2$, where t is the number of neighbors of z in G' distinct from x and y. Since $zx, zy \in E(G)$, $t \leq d-1$. Let $C := (C' - xy) + \{e, f\}$; then $|C| \geq \frac{1}{2}(\frac{(d-1)n}{dt})^r + 3 \geq \frac{1}{2}(\frac{n}{d})^r + 3$. So C gives the desired cycle in G.

Therefore, we may assume that e and f are not incident. Let $e = xy$; then $f \in E(G-y)$. Since G is 3-connected, $G-y$ is 2-connected.

Suppose $G-y$ is 3-connected. Let $y' \neq x$ be a neighbor of y. Then $G' := (G-y)+xy'$ is a 3-connected graph, $\Delta(G') \leq d$, and $5 \leq |G'| < n$. By Theorem 1.2(b), there is a cycle C' through xy' and f in G' such that $|C'| \geq \frac{1}{2}(\frac{n-1}{d})^r + 3$. Let $C := (C' - xy') + \{y, xy, yy'\}$. Then $|C| = |C'| + 1 \geq \frac{1}{2}(\frac{n-1}{d})^r + 4$. By Lemma 3.1, $|C| \geq \frac{1}{2}(\frac{n}{d})^r + 3$. So C gives the desired cycle in G.

Hence, we may assume that $G-y$ is not 3-connected. By Theorem 2.1, we decompose $G-y$ into 3-connected components. Let $\mathcal{H} := H_1 \dots H_h$ be a block chain in $G-y$ such that (a) $f \in E(H_1)$ and $x \in V(H_h)$, (b) if $h = 1$ and $H_1 = C_1 \dots C_k$ is a cycle chain with $k \geq 2$, then $x \in V(C_k) - V(C_{k-1})$, $f \in E(C_1)$, and f is not incident with both vertices in $V(C_1 \cap C_2)$, and (c) if $h \geq 2$, then $x \in V(H_h) - V(H_{h-1})$, if $H_h = C_1 \dots C_k$ is a cycle chain with $k \geq 2$ and $V(H_{h-1} \cap H_h) = V(C_1 \cap H_{h-1})$, then $x \in V(C_k) - V(C_{k-1})$, $f \in E(H_1)$, f is not incident with both vertices in $V(H_1 \cap H_2)$, and if $H_1 = C_1 \dots C_k$ is a cycle chain with $k \geq 2$ and $V(H_1 \cap H_2) = V(C_k \cap H_2)$, then $f \in E(C_1)$ and f is not incident with both vertices in $V(C_1 \cap C_2)$. Define $V(H_s \cap H_{s+1}) = \{a_s, b_s\}$ for $1 \leq s \leq h-1$.

Suppose $V(\mathcal{H}) = V(G-y)$. If $h = 1$, then $G-y$ is a cycle chain, and it is easy to see that G has a Hamilton cycle through e and f, and hence, Theorem 1.2(b) holds. So assume $h \geq 2$. Let $x' \in V(H_1) - V(H_2)$ so that $yx' \in E(G)$, and in addition, if f has an end with degree 2 in $\mathcal{H}$, then choose x' to be that end (in this case, $yx' \in E(G)$).

Let G' be obtained from $G-y$ by adding xx' and then suppressing all degree 2 vertices and deleting separating edges of $\mathcal{H}$. Now G' is 3-connected, $|G'| \geq n-1-(d-2)$ (because degree of y in G is at most d), and $\Delta(G') \leq d$. Therefore, by Theorem 1.2(b), G' has a cycle C' through f and xx' such that $|C'| \geq \frac{1}{2}(\frac{|G'|}{d})^r + 3$. By replacing edges in $(C' - xx') + \{y, yx, yx'\}$ but not in G with appropriate paths in G, we obtain a cycle C in G through e and f such that $|C| \geq |C'| + 1 \geq \frac{1}{2}(\frac{n-d+1}{d})^r + 4 \geq \frac{1}{2}(\frac{n}{d})^r + 3$, where the final inequality follows from Lemma 3.1. So C is the desired cycle.

We thus may assume that $\mathcal{H} \neq G - y$. Then there is a 2-cut $\{p, q\}$ of $G - y$ such that pq is a virtual edge in H_t for some $1 \leq t \leq h$. Define G_1 as the graph obtained from G by deleting those components of $(G - y) - \{p, q\}$ containing a vertex of $\mathcal{H}$. Note that $G_1 - \{p, q, y\}$ contains a neighbor of y. We choose $\{p, q\}$ so that $|G_1|$ is maximum. Because y has degree at most d in G and $yx \in E(G)$, and since all degree 2 vertices of $G - y$ are neighbors of y, we have (from the choice of G_1) the following observation.

Observation 1. $|G_1| \geq \frac{n-\sigma(\mathcal{H})}{d-1}$.

If there is a 2-cut $\{v, w\}$ of $G-y$ such that $\{v, w\} \subseteq V(\mathcal{H} \cup G_1)$ and $(G-y)-\{v, w\}$ has a component not containing any vertex of $\mathcal{H} \cup G_1$, then let G_2 denote the graph obtained from G by deleting those components of $(G-y)-\{v, w\}$ containing a vertex of $\mathcal{H} \cup G_1$. If such a 2-cut does not exist, then let $G_2 = \emptyset$. From the definition of G_1, we see that $\{v, w\} \subseteq V(\mathcal{H})$, $\{v, w\} \neq \{p, q\}$, and $V(G_1 \cap G_2) \subseteq \{p, q, y\} \cap \{v, w, y\}$. Choose $\{v, w\}$ so that $|G_2|$ is maximum. By the same reason for Observation 1, we have the following two observations.

Observation 2. If $\sigma(\mathcal{H}) \geq |G_2|$, then $\sigma(\mathcal{H}) \geq \frac{n-|G_1|}{d-1}$.

Observation 3. If $|G_2| \geq \sigma(\mathcal{H})$, then $|G_2| \geq \frac{n-|G_1|}{d-1}$.

We consider three cases.

Case 1. $\sigma(\mathcal{H}) \geq |G_2|$.

We use $\mathcal{H}$ and G_1 to find the desired cycle. Choose t so that $\{p, q\} \neq \{a_t, b_t\}$. (Note that a_t, b_t are not defined when $t = h$.) Clearly, $|H_t| \leq n - 3$ when $h \geq 2$. By Lemma 3.7(ii), there is a path P from x to $\{p, q\}$ in $\mathcal{H}$ such that $f \in E(P)$, $pq \notin E(P)$ unless $pq = f$, and $|E(P)| \geq \frac{1}{2}(\frac{\sigma(\mathcal{H})}{d})^r + 1$. Assume the notation of $\{p, q\}$ is chosen so that P is from x to p.

Since G is 3-connected, $G_1' := G_1 + \{yp, yq, pq\}$ is 3-connected. If $G_1' \cong K_4$, then we can find a path Q in $G_1' - q$ from p to y such that $|E(Q)| = 2 \geq \frac{1}{2}(\frac{|G_1|}{d})^r + 1$. Now assume that $G_1' \ncong K_4$. Note that $\Delta(G_1') \leq d + 1$, and y, p, q are the only possible vertices with degree $d+1$. By Theorem 1.2(a), there is a cycle C_1 through py in $G_1' - q$ such that $|C_1| \geq \frac{1}{2}(\frac{(d-1)|G_1|}{dt_1})^r + 2$, where $t_1 \leq d - 1$ is the number of neighbors of q in G_1' distinct from p and y. Hence, $|C_1| \geq \frac{1}{2}(\frac{|G_1|}{d})^r + 2$.

Let $C^* := (P \cup (C_1 - py)) + xy$. Then C^* is a cycle through e and f and $|C^*| \geq \frac{1}{2}[(\frac{\sigma(\mathcal{H})}{d})^r + (\frac{|G_1|}{d})^r] + 3$. If $\sigma(\mathcal{H}) \leq |G_1|$, then

$$\begin{aligned}
|C^*| &\geq \frac{1}{2}\left(\frac{(b-1)\sigma(\mathcal{H})}{d} + \frac{|G_1|}{d}\right)^r + 3 \quad \text{(by Lemma 3.1)} \\
&> \frac{1}{2}\left(\frac{n-|G_1|}{d} + \frac{|G_1|}{d}\right)^r + 3 \quad \text{(by Observation 2)} \\
&= \frac{1}{2}\left(\frac{n}{d}\right)^r + 3.
\end{aligned}$$

So we may assume $\sigma(\mathcal{H}) \geq |G_1|$. Then

$$\begin{aligned}|C^*| &\geq \frac{1}{2}\left(\frac{\sigma(\mathcal{H})}{d} + \frac{(b-1)|G_1|}{d}\right)^r + 3 \quad \text{(by Lemma 3.1)}\\ &> \frac{1}{2}\left(\frac{\sigma(\mathcal{H})}{d} + \frac{n-\sigma(\mathcal{H})}{d}\right)^r + 3 \quad \text{(by Observation 1)}\\ &= \frac{1}{2}\left(\frac{n}{d}\right)^r + 3.\end{aligned}$$

The desired cycle C can be obtained from C^* by replacing virtual edges in C^* with appropriate paths in G.

Case 2. $\sigma(\mathcal{H}) < |G_2|$.

Then G_2 is nonempty. We use G_1 and G_2 to find the desired cycle. There exists some $1 \leq u \leq h$ such that $\{v, w\} \subseteq V(H_u)$, and we may choose u so that $\{v, w\} \neq \{a_{u-1}, b_{u-1}\}$. (Note that a_{u-1}, b_{u-1} are not defined when $u = 1$.) We may choose t so that $\{p, q\} \neq \{a_{t-1}, b_{t-1}\}$. Again, a_{t-1}, b_{t-1} are not defined when $t = 1$.

(1) We claim that there is a path P in $\mathcal{H}$ from x to some $z \in \{p, q\} \cup \{v, w\}$ and containing no separating edge of $\mathcal{H}$ such that (i) $f \in E(P)$, (ii) $pq \in E(P)$ or $vw \in E(P)$, (iii) if $pq \in E(P)$, then $z \in \{v, w\}$, and $vw \notin E(P)$ unless $vw = f$, and (iv) if $vw \in E(P)$, then $z \in \{p, q\}$, and $pq \notin E(P)$ unless $pq = f$.

We prove (1) for $t \leq u$; the case $t \geq u$ can be treated in the same way.

First, we define Q. When $t \neq 1$, we find a cycle Q' in $\bigcup_{s=1}^{t-1} H_s$ through $a_{t-1}b_{t-1}$ and f and containing no separating edge of $\mathcal{H}$ (except $a_{t-1}b_{t-1}$). Let $Q := Q' - a_{t-1}b_{t-1}$, which is a path from a_{t-1} to b_{t-1} through f. Let $Q = \emptyset$ when $t = 1$.

Suppose $t < u$. Since removing separating edges of $H_{t+1} \dots H_h$ that are different from vw results in a 2-connected graph, we may choose the notation of $\{a_t, b_t\}$ so that $(\bigcup_{s=t+1}^{h} H_s) - b_t$ contains a path X from a_t to x through vw and containing no separating edge of $\mathcal{H}$ (except possibly vw).

We claim that there is a path C_t in $H_t - a_tb_t$ from a_t to $\{p, q\}$ through $a_{t-1}b_{t-1}$ (or f when $t = 1$), or a path C'_t in H_t from a_t to b_t through $a_{t-1}b_{t-1}$ (or f when $t = 1$) and pq. If $\{p, q\} = \{a_t, b_t\}$, then the existence of C_t follows from 2-connectivity of H_t. So we may assume that $\{p, q\} \neq \{a_t, b_t\}$. Again by 2-connectivity of H_t there is a cycle D in H_t through pq and $a_{t-1}b_{t-1}$ (or f when $t = 1$). If $a_tb_t \in E(D)$, then $C'_t := D - a_tb_t$ is as desired. So we may assume $a_tb_t \notin E(D)$. By 2-connectivity of H_t, there is a path A in H_t from a_t to D and internally disjoint from D. One can easily check that C_t exists in $A \cup D$.

If we find C_t, then let $P_t := C_t - a_{t-1}b_{t-1}$ when $t \neq 1$ and $P_t := C_t$ when $t = 1$. In this case, $P := Q \cup P_t \cup X$ gives the desired path for (1). So assume that we find C'_t. Let $P_t := C'_t$ if $t = 1$, and otherwise let $P_t := C'_t - a_{t-1}b_{t-1}$. Let $H := H_{t+1} \dots H_h$. If $x \in \{v, w\}$, then we find a cycle C' in H through a_tb_t and vw and containing no separating edge of $\mathcal{H}$ (except a_tb_t and vw), and $P := Q \cup P_t \cup (C' - \{a_tb_t, vw\})$ gives the desired path for (1). Therefore, we may assume $x \notin \{v, w\}$. Let H' be obtained from H by a T-transform at $\{x, vw\}$, let x' denote the new vertex, and let H'' be obtained from H' by deleting all separating edges of H different from a_tb_t. Then H'' is a 2-connected graph. So there is a cycle C'' in H'' through a_tb_t and xx'. Now $P := Q \cup P_t \cup (C'' - \{x', a_tb_t\})$ gives the desired path for (1).

Therefore, we may assume $t = u$. We claim that there is a path Q_t in H_t from $\{a_t, b_t\}$ when $t \neq h$, or from x when $t = h$, to some $z \in \{p, q\} \cup \{v, w\}$ such that (i) $a_{t-1}b_{t-1} \in E(Q_t)$ (or $f \in E(Q_t)$ when $t = 1$), (ii) $pq \in E(Q_t)$ or $vw \in E(Q_t)$, (iii) if $pq \in E(Q_t)$, then $z \in \{v, w\}$, and $vw \notin E(Q_t)$ unless $vw = f$, and (iv) if $vw \in E(Q_t)$,

then $z \in \{p, q\}$, and $pq \notin E(Q_t)$ unless $pq = f$. This is easy to see if H_t is a cycle chain (because $pq \neq vw$). Otherwise, it follows from Lemma 2.8 or Lemma 2.9 when $f \notin \{pq, vw\}$, and follows from 3-connectivity of H_t when $f \in \{pq, vw\}$.

Assume without loss of generality that a_t is an end of Q_t. When $t \neq h$, we find a path R from a_t to x in $(H_{t+1} \dots H_h) - b_t$ containing no separating edge of $\mathcal{H}$. When $t = h$, let $R = \emptyset$. Let $P_t := Q_t$ when $t = 1$, and otherwise let $P_t := Q_t - a_{t-1}b_{t-1}$. Then $P := Q \cup P_t \cup R$ gives the desired path for (1).

We may assume that $vw \in E(P)$ and p is an end of P, since the case $pq \in E(P)$ is similar.

(2) Note that $G_1' := G_1 + \{yp, yq, pq\}$ is 3-connected, $\Delta(G_1') \leq d + 1$, and y, p, q are the possible vertices of degree $d + 1$ in G_1'. If $G_1' \cong K_4$, then we can find a path P_1 from p to y in $G_1' - q$ such that $|E(P_1)| = 2 \geq \frac{1}{2}(\frac{|G_1|}{d})^r + 1$. If $G_1' \ncong K_4$, then by Theorem 1.2(a), there is a cycle C_1 through py in $G_1' - q$ such that $|C_1| \geq \frac{1}{2}(\frac{(d-1)|G_1|}{dt_1})^r + 2$, where $t_1 \leq d-1$ is the number of neighbors of q in G_1' distinct from p and y. Let $P_1 := C_1 - py$; then $|E(P_1)| \geq \frac{1}{2}(\frac{|G_1|}{d})^r + 1$.

(3) Note that $G_2' := G_2 + \{yv, yw, vw\}$ is 3-connected, $\Delta(G_2') \leq d + 1$, and y, v, w are the possible vertices of degree $d + 1$ in G_2'. If $G_2' \cong K_4$, then we can find a path P_2 from v to w in $G_2' - y$ such that $|E(P_2)| = 2 \geq \frac{1}{2}(\frac{|G_2|}{d})^r + 1$. If $G_2' \ncong K_4$, then by Theorem 1.2(a), there is a cycle C_2 through vw in $G_2' - y$ such that $|C_2| \geq \frac{1}{2}(\frac{(d-1)|G_2'|}{dt_2})^r + 2$, where $t_2 \leq d-1$ is the number of neighbors of y in G_2' distinct from v and w. Let $P_2 := C_2 - vw$; then $|E(P_2)| \geq \frac{1}{2}(\frac{|G_2|}{d})^r + 1$.

Let $C^* := ((P - vw) \cup P_1 \cup P_2) + e$. Then C^* is a cycle through e and f and

$$\begin{aligned}
|C^*| &\geq |E(P_1)| + |E(P_2)| + 1 \\
&\geq \frac{1}{2}\left(\frac{|G_1|}{d}\right)^r + \frac{1}{2}\left(\frac{|G_2|}{d}\right)^r + 3 \quad \text{(by (2) and (3))} \\
&\geq \frac{1}{2}\left(\frac{|G_1|}{d} + \frac{(b-1)|G_2|}{d}\right)^r + 3 \quad \text{(by Lemma 3.1 and since } |G_1| \geq |G_2|) \\
&> \frac{1}{2}\left(\frac{|G_1|}{d} + \frac{n - |G_1|}{d}\right)^r + 3 \quad \text{(by Observation 3 and since } |G_2| \geq \sigma(\mathcal{H})) \\
&= \frac{1}{2}\left(\frac{n}{d}\right)^r + 3.
\end{aligned}$$

As before, the desired cycle C can be obtained by modifying C^*. □

5. Cycles through one edge. We now reduce Theorem 1.2(c); our proof implies an $O(E)$ time reduction. Here we use Lemmas 3.2 and 3.3, and need $b = \max\{64, 4d + 1\}$.

LEMMA 5.1. *Let $n \geq 6$ be an integer, and assume that Theorem 1.2 holds for graphs with at most $n - 1$ vertices. Let G be a 3-connected graph with n vertices and $\Delta(G) \leq d$. Then for any $e \in E(G)$, there is a cycle C through e in G such that $|C| \geq \frac{1}{2}n^r + 3$.*

Proof. By Lemma 1.3, we may assume $n > (4d+1)^2$. Let $e = xy \in E(G)$. If $G - y$ is 3-connected, then let y' be a neighbor of y other than x. Clearly, $G' := (G - y) + xy'$ is 3-connected, $\Delta(G') \leq d$, and $5 \leq |G'| < n$. By Theorem 1.2(c), there is a cycle C' through xy' in G' such that $|C'| \geq \frac{1}{2}(n-1)^r + 3$. Now let $C := (C' - xy') + \{y, xy, yy'\}$. Then C is a cycle through xy in G and, by Lemma 3.1,

$$|C| = |C'| + 1 \geq \frac{1}{2}(n-1)^r + 1 + 3 \geq \frac{1}{2}n^r + 3.$$

Therefore, we may assume that $G-y$ is not 3-connected. Since $G-y$ is 2-connected, we use Theorem 2.1 to decompose $G-y$ into 3-connected components.

Suppose all 3-blocks of $G-y$ are cycles. Let $\mathcal{L}=L_1\dots L_\ell$ be a cycle chain in $G-y$ such that (i) $x\in V(L_1)$, (ii) L_ℓ is an extreme 3-block of $G-y$, and (iii) subject to (i) and (ii), $|\mathcal{L}|$ is maximum. Because G is 3-connected, each degree 2 vertex in $\mathcal{L}$ is a neighbor of y or is contained in a 3-block of $G-y$ not in $\mathcal{L}$. Hence, it is easy to see that there is some $y'\in V(\mathcal{L})-\{x\}$ such that $\mathcal{L}$ contains a Hamilton path P from x to y' and G has a path Q from y' to y disjoint from $V(\mathcal{L})-\{y'\}$. Let $C:=(P\cup Q)+\{y,xy,yy'\}$, which is a cycle in G. Then $|C|\geq|\mathcal{L}|+1$. If $V(G-y)=V(\mathcal{L})$, then $|C|=n\geq\frac{1}{2}n^r+3$ (since $n\geq 5$ and by Lemma 3.4). So we may assume $V(G-y)\neq V(\mathcal{L})$. Write $B:=L_1$. Because $x\in V(L_1)$ and $xy\in E(G)$, it follows from (iii) that $|\mathcal{L}|\geq\frac{(n-1)-|B|}{t}+|B|=\frac{n+(t-1)|B|-1}{t}$, where t is the number of extreme 3-blocks of $G-y$ distinct from L_1. So $2\leq t\leq d-1$ (because $V(G-y)\neq V(\mathcal{L})$). Then $|C|\geq|\mathcal{L}|+1\geq\frac{n+(t-1)|B|-1}{t}+1$. Note that $|C|-3\geq\frac{n+(t-1)|B|-1}{t}-2\geq\frac{n+t-4}{t}$ (since $|B|\geq 3$). Using elementary calculus, we can show that the function $\frac{x+t-4}{t}-\frac{1}{2}x^r$ is increasing when $x\geq(4d+1)^2$. Hence $\frac{n+t-4}{t}\geq\frac{1}{2}n^r$ (because $t\leq d-1$ and $n\geq(4d+1)^2$). Therefore, $|C|\geq\frac{1}{2}n^r+3$ and C gives the desired cycle in G.

Hence, we may assume that not all 3-blocks of $G-y$ are cycles. We choose a 3-connected 3-block H_0 of $G-y$ with $|H_0|$ maximum. Let $\mathcal{H}=H_0H_1H_2\dots H_h$ be a block chain in $G-y$ such that either $h=0$ and $x\in V(H_0)$, or $h\geq 1$ and $x\in V(H_h)-V(H_{h-1})$, and if $H_h=C_1\dots C_k$ is a cycle chain with $k\geq 2$ and $V(H_{h-1}\cap H_h)=V(C_1\cap C_2)$, then $x\in V(C_k)-V(C_{k-1})$. For $0\leq i\leq h-1$, let $V(H_i\cap H_{i+1})=\{a_i,b_i\}$.

If $V(G-y)\neq V(\mathcal{H})$, there is a block chain $\mathcal{L}:=L_1L_2\dots L_\ell$ in $G-y$ such that $V(\mathcal{H}\cap\mathcal{L})=V(\mathcal{H}\cap L_1)$ consists of two vertices c_0 and d_0, L_ℓ is (or contains) an extreme 3-block of $G-y$, and if $L_1=C_1\dots C_k$ is a cycle chain with $k\geq 2$ and $V(L_1\cap L_2)=V(C_k\cap H_2)$ when $\ell\geq 2$, then $c_0d_0\in E(C_1)$ and $\{c_0,d_0\}\neq V(C_1\cap C_2)$. For $1\leq i\leq\ell-1$, let $V(L_i\cap L_{i+1})=\{c_i,d_i\}$. If $\mathcal{L}$ exists, we choose $\mathcal{L}$ so that $\sigma(\mathcal{L})$ is maximum.

(1) We may assume $V(G-y)\neq V(\mathcal{H})$, and $\sigma(\mathcal{L})+2\geq\frac{n-\sigma(\mathcal{H})-1}{d-1}$.

Suppose $V(G-y)=V(\mathcal{H})$. When $h=0$, let x' be a neighbor of y in H_0-x, otherwise, let x' be a neighbor of y in $H_0-V(H_1)$. Let G' be obtained from $H+xx'$ by suppressing all degree 2 vertices and deleting separating edges of $\mathcal{H}$. Then G' is 3-connected. By Theorem 1.2(c), there is a cycle C' in G' through xx' such that $|C'|\geq\frac{1}{2}|G'|^r+3$. Let $C^*:=(C'-xx')+\{y,yx,yx'\}$. Since $\Delta(G)\leq d$, $|G'|\geq(n-1)-(d-2)$. Hence, $|C^*|=|C'|+1\geq\frac{1}{2}(n-d+1)^r+1+3>\frac{1}{2}n^r+3$ (by Lemma 3.1). Clearly, the desired cycle C can be obtained by modifying C^*.

So we may assume $V(G-y)\neq V(\mathcal{H})$. Note that any vertex of G not contained in any $A(H_i)$, $1\leq i\leq h$, either is counted in $\sigma(\mathcal{L}')+2$ for some block chain $\mathcal{L}'$ defined as $\mathcal{L}$ except the maximum requirement (the constant 2 counts the vertices in $V(\mathcal{H}\cap\mathcal{L}')$), or is a degree 2 vertex in $G-y$ (and hence a neighbor of y). Therefore, since $xy\in E(G)$ and $\Delta(G)\leq d$, $\sigma(\mathcal{L})+2\geq\frac{n-\sigma(\mathcal{H})-1}{d-1}$.

(2) There exists a path P in $\mathcal{H}$ from x to $\{c_0,d_0\}$ such that $c_0d_0\notin E(P)$ and $|E(P)|\geq\frac{1}{2}|H_0|^r+\frac{1}{2}(\sum\{(\frac{|H_i|}{d})^r:i\neq 0\text{ and }H_i\in B_1(\mathcal{H})\})+(\sum\{\max\{1,|A(H_i)|-2\}:i\neq 0\text{ and }H_i\in B_1(\mathcal{H})\})+1$. In particular, $|E(P)|\geq\frac{1}{2}(\sigma(\mathcal{H}))^r+1$.

The first part of (2) follows from Lemma 3.7(i). The second part of (2) follows from Lemma 3.1. When applying Lemma 3.1, we express $\max\{1,|A(H_i)|-2\}$ as the sum of 1, and we use $b\geq 4d+1$, $(b-1)(|A(H_i)|-2)\geq|A(H_i)|$ when $|A(H_i)|\geq 3$,

and the fact that $|H_0| \geq |H_i|$ for all 3-connected H_i.

(3) We may assume $\sigma(\mathcal{H}) < \frac{n-1}{4}$.

Suppose $\sigma(\mathcal{H}) \geq \frac{n-1}{4}$. Without loss of generality, assume c_0 is an end of the path P in (2). By Lemma 3.6(i), there is a path Q in $\mathcal{L} - d_0$ from c_0 to some $y' \in N(y) \cap V(L_\ell)$ such that $|E(Q)| \geq \frac{1}{2}(\frac{\sigma(\mathcal{L})}{d})^r + 1$. Let $C^* := (P \cup Q) + \{y, yy', yx\}$. Then

$$|C^*| = |E(P)| + |E(Q)| + 2 \geq \frac{1}{2}(\sigma(\mathcal{H}))^r + 1 + \frac{1}{2}\left(\frac{\sigma(\mathcal{L})}{d}\right)^r + 3.$$

If $\sigma(\mathcal{H}) \leq \frac{b(b-1)}{4}\frac{\sigma(\mathcal{L})}{d}$, then by Lemmas 3.3 and 3.1,

$$|C^*| \geq \frac{1}{2}(4\sigma(\mathcal{H}) + 1)^r + 3 \geq \frac{1}{2}n^r + 3.$$

If $\sigma(\mathcal{H}) \geq \frac{b(b-1)}{4}\frac{\sigma(\mathcal{L})}{d}$, then by Lemma 3.2 and since $b \geq 4d + 1$,

$$\begin{aligned} |C^*| &> \frac{1}{2}\left(\sigma(\mathcal{H}) + \frac{b(b-1)}{4}\frac{\sigma(\mathcal{L})}{d} + 2(b-1)\right)^r + 3 \\ &\geq \frac{1}{2}(\sigma(\mathcal{H}) + (4d+1)\sigma(\mathcal{L}) + 8d)^r + 3 \\ &> \frac{1}{2}n^r + 3. \end{aligned}$$

The final inequality holds by (1) and $\sigma(\mathcal{H}) < n - 1$. Now the desired cycle C can be obtained from C^* by replacing virtual edges in C^* with appropriate paths in G.

(4) We may assume $|H_0| + 4(\sigma(\mathcal{H}) - |H_0| + \sigma(\mathcal{L})) < n$. In particular, $\sigma(\mathcal{L}) \leq \frac{n-1-|H_0|}{4}$.

Suppose $|H_0| + 4(\sigma(\mathcal{H}) - |H_0| + \sigma(\mathcal{L})) \geq n$. Without loss of generality, assume that the path P in (2) is from x to c_0. By Lemma 3.6(ii), there is a path Q in $\mathcal{L} - d_0$ from c_0 to some $y' \in N(y) \cap V(L_\ell)$ such that $|E(Q)| \geq \frac{1}{2}(\sum\{(\frac{|A(L_i)|}{d})^r : L_i \in B_1(\mathcal{L})\}) + (\sum\{\max\{1, |A(L_i)| - 2\} : L_i \in B_2(\mathcal{L})\}) + 1$.

Let $C^* = (P \cup Q) + \{y, yy', yx\}$. Then by (2) and above, $|C^*| = |E(P)| + |E(Q)| + 2 \geq \frac{1}{2}|H_0|^r + \frac{1}{2}(\sum\{(\frac{|A(H_i)|}{d})^r : i \neq 0 \text{ and } H_i \in B_1(\mathcal{H})\}) + (\sum\{\max\{1, |A(H_i)| - 2\} : i \neq 0 \text{ and } H_i \in B_2(\mathcal{H})\}) + \frac{1}{2}(\sum\{(\frac{|A(L_i)|}{d})^r : L_i \in B_1(\mathcal{L})\}) + (\sum\{\max\{1, |A(L_i)| - 2\} : L_i \in B_2(\mathcal{L})\}) + 4$. Because $|H_0|$ is maximum among all 3-connected 3-blocks of $G - y$, it follows from Lemma 3.1 and the fact that $b \geq 4d + 1$ that

$$\begin{aligned} |C^*| &\geq \frac{1}{2}\left[|H_0| + 4\left(\sum_{i=1}^{h}|A(H_i)| + \sum_{j=1}^{\ell}|A(L_j)|\right)\right]^r + 4 \\ &= \frac{1}{2}[|H_0| + 4(\sigma(\mathcal{H}) - |H_0| + \sigma(\mathcal{L}))]^r + 4 \\ &> \frac{1}{2}n^r + 3. \end{aligned}$$

As before, the desired cycle C can be obtained by modifying C^*. This proves (4).

We need to consider block chains other than $\mathcal{H}$ and $\mathcal{L}$. A block chain $\mathcal{M} := M_1M_2 \dots M_m$ is called an *$\mathcal{HL}$-leg* if M_m contains an extreme 3-block of $G - y$ and $V(\mathcal{M} \cap (\mathcal{H} \cup \mathcal{L}))$ consists of two vertices x_0 and y_0 such that $\{x_0, y_0\} \subseteq V(M_1)$ and $\{x_0, y_0\} \neq V(M_1 \cap M_2)$ when $m \geq 2$, and if $M_1 = C_1 \dots C_k$ is a cycle chain with

$k \geq 2$ and $V(C_k \cap M_2) = V(M_1 \cap M_2)$ when $m \geq 2$, then $\{x_0, y_0\} \subseteq V(C_1)$ and $\{x_0, y_0\} \neq V(C_1 \cap C_2)$. We view degree 2 vertices of $G - y$ (which are neighbors of y) as trivial $\mathcal{HL}$-legs.

(5) We may assume that there is an $\mathcal{HL}$-leg $\mathcal{M}$ such that $\sigma(\mathcal{M}) > \frac{n}{4(d-2)} > 4d+2$.

Note that each extreme 3-block of $G-y$ contains a neighbor of y. Since $\Delta(G) \leq d$, there are at most $d-2$ $\mathcal{HL}$-legs in $G-y$ (including those trivial ones). Choose an $\mathcal{HL}$-leg $\mathcal{M}$ such that $\sigma(\mathcal{M})$ is maximum. Note that every vertex of $G-y$ is either a degree 2 vertex (hence covered in a trivial $\mathcal{HL}$-leg), or is counted in $\sigma(\mathcal{H})$, or in $\sigma(\mathcal{L})+2$, or in $\sigma(\mathcal{M})+2$ for some $\mathcal{HL}$-leg $\mathcal{M}$. Hence, because $\sigma(\mathcal{H}) < \frac{n-1}{4}$ (by (3)) and $\sigma(\mathcal{L}) \leq \frac{n-1-|H_0|}{4} \leq \frac{n-5}{4}$ (by (4) and $|H_0| \geq 4$), $\sigma(\mathcal{M})+2 \geq \frac{n-1-\sigma(\mathcal{H})-\sigma(\mathcal{L})-2}{d-2} > \frac{n-3}{2(d-2)}$. Since we assume $n > (4d+1)^2$, $\sigma(\mathcal{M}) > \frac{n}{4(d-2)} > 4d+2$.

Let $\mathcal{M}$ be an $\mathcal{HL}$-leg in $G-y$ with $\sigma(\mathcal{M}) \geq \frac{n}{4(d-2)}$. By (5), $\mathcal{M}$ is nontrivial. Let x_0 and y_0 be the vertices in $V(\mathcal{M} \cap (\mathcal{H} \cup \mathcal{L}))$. We consider three cases.

Case 1. $\mathcal{M}$ may be chosen so that $x \notin \{x_0, y_0\} \cap \{c_0, d_0\}$ and $\{x_0, y_0\} \not\subseteq V(\mathcal{H})$.

Then we may assume $\{x_0, y_0\} \subseteq V(L_t)$ with $\{x_0, y_0\} \neq \{c_{t-1}, d_{t-1}\}$.

We claim that there is a path P' in $\mathcal{H}$ from x to $z \in \{c_0, d_0\}$ such that (i) $|E(P')| \geq \frac{1}{2}(\frac{|H_0|}{d})^r + 1$, (ii) $c_0d_0 \notin E(P')$, and (iii) if $z \notin \{c_0, d_0\} \cap \{x_0, y_0\}$, then $\{c_0, d_0\} \cap \{x_0, y_0\} = \emptyset$ or $\{c_0, d_0\} \cap \{x_0, y_0\} \not\subseteq V(P')$. Choose $z' \in \{c_0, d_0\}$ such that, if possible, $z' \in \{c_0, d_0\} \cap \{x_0, y_0\}$. Suppose $c_0d_0 \in E(H_1 \dots H_h)$. Deleting separating edges of $H_1 \dots H_h$ results in a 2-connected graph, which contains disjoint paths Q_1, Q_2 from x, z' to a_0, b_0, respectively. In H_0 we use Theorem 1.2(c) to find a cycle C_0 through a_0b_0 such that $|C_0| \geq \frac{1}{2}|H_0|^r + 3$. If $c_0d_0 \in E(Q_2)$, then $P' := (C_0 - a_0b_0) \cup Q_1 \cup (Q_2 - z')$ gives the desired path; otherwise, $P' := (C_0 - a_0b_0) \cup Q_1 \cup Q_2$ gives the desired path. So we may assume $c_0d_0 \notin E(H_1 \dots H_h)$. Suppose $h = 0$. We apply Theorem 1.2(a) to find a cycle C_0 in $(H_0 + \{xc_0, xd_0\}) - (\{c_0, d_0\} - \{z'\})$ through xz' such that $|C_0| \geq \frac{1}{2}(\frac{|H_0|}{d})^r + 2$; then $P' := C_0 - xz'$ gives the desired path. So let $h \geq 1$. Then $c_0d_0 \in E(H_0)$ and $\{c_0, d_0\} \neq \{a_0, b_0\}$. Without loss of generality, we may assume $a_0 \notin \{c_0, d_0\}$. Let Q' be a path in $(H_1 \dots H_h) - b_0$ from x to a_0 and not containing any separating edge of $\mathcal{H}$. If $z' = b_0$ we use Theorem 1.2(c) to find a cycle C_0 through a_0b_0 in H_0 such that $|C_0| \geq \frac{1}{2}|H_0|^r + 3$, and $P' := (C_0 - a_0b_0) \cup Q'$ (when $c_0d_0 \notin E(C_0)$) or $P' := (C_0 - b_0) \cup Q'$ (when $c_0d_0 \in E(C_0)$) gives the desired path. So assume $z' \neq b_0$. If $z' \in \{c_0, d_0\} \cap \{x_0, y_0\}$, then z' has at most $d-1$ neighbors in H_0 (since $\{x_0, y_0\} \neq \{c_0, d_0\}$), and in $(H_0 + \{a_0z', z'b_0\}) - b_0$ we apply Theorem 1.2(a) to find a cycle C_0 through a_0z' such that $|C_0| \geq \frac{1}{2}(\frac{|H_0|}{d})^r + 2$; then $P' := (C_0 - a_0z') \cup Q'$ gives the desired path. So we may assume $z' \notin \{c_0, d_0\} \cap \{x_0, y_0\}$. By the choice of z', $\{c_0, d_0\} \cap \{x_0, y_0\} = \emptyset$ (so (iii) is automatic). Let H_0' be obtained from H_0 by an H-transform at $\{a_0b_0, c_0d_0\}$, and let a', x' denote the new vertices. By applying Theorem 1.2(c) we find a cycle C_0 through $a'x'$ in H_0' such that $|C_0| \geq \frac{1}{2}|H_0|^r + 3$. Now $C_0 - \{a', x'\}$ is a path from some $z \in \{c_0, d_0\}$ to some $b' \in \{a_0, b_0\}$. Then $C_0 - \{a', x'\}$ and a path in $(H_1 \dots H_h) - (\{a_0, b_0\} - \{b'\})$ from x to b' (not containing any separating edge of $\mathcal{H}$) gives the desired path P'.

Without loss of generality, we may assume that P' is from x to c_0. Then $d_0 \notin V(P')$ or $d_0 \notin \{x_0, y_0\}$. Therefore, since each L_i is 3-connected or is a cycle chain, there exists a path Q in $\bigcup_{i=0}^{t} L_i$ from c_0 to some $z \in \{c_t, d_t\} \cup \{x_0, y_0\}$ such that (i) $c_0d_0 \in E(Q)$ Q contains no separating edge of $\mathcal{L}$ except possibly c_td_t and x_0y_0, (ii) Q avoids d_0 if $d_0 \in V(P')$ (since in that case $\{x_0, y_0\} \cap \{c_0, d_0\} = \emptyset$), (iii) if $z \in \{c_t, d_t\}$, then $x_0y_0 \in E(Q)$, and $c_td_t \notin E(Q)$ unless $x_0y_0 = c_td_t$, and (iv) if $z \in \{x_0, y_0\}$, then $c_td_t \in E(Q)$, and $x_0y_0 \notin E(Q)$ unless $x_0y_0 = c_td_t$.

Suppose $z \in \{c_t, d_t\}$, and assume the notation is chosen so that $z = c_t$. By Lemma 3.6(ii) there is a path P_1 in $(L_{t+1} \dots L_\ell) - d_t$ from z to some $y' \in N(y) \cap V(L_\ell)$ and containing no separating edge of $\mathcal{L}$ such that $|E(P_1)| \geq \frac{1}{2}(\sum\{(\frac{|A(L_i)|}{d})^r : t+1 \leq i \leq \ell$ and $L_i \in B_1(\mathcal{L})\}) + (\sum\{\max\{1, |A(L_i)| - 2\} : t+1 \leq i \leq \ell$ and $L_i \in B_2(\mathcal{L})\}) + 1$. By Lemma 3.5, let P_2 be a path from x_0 to y_0 in $\mathcal{M}$ such that $|E(P_2)| \geq \frac{1}{2}(\frac{(d-1)\sigma(\mathcal{M})}{d})^r + 1$. Let $n^* := \sum_{i=t+1}^{\ell} |A(L_i)|$; then by the choice of $\mathcal{L}$, $n^* \geq \sigma(\mathcal{M}) - 2$. Let $C^* := (P' \cup (Q - x_0y_0) \cup P_1 \cup P_2) + \{y, yy', yx\}$. As in the proof of (4),

$$\begin{aligned}
|C^*| &\geq |E(P')| + |E(P_1)| + |E(P_2)| + 2 \\
&\geq \frac{1}{2}\left[\left(\frac{|H_0|}{d}\right)^r + 2 + \sum\left(\frac{|A(L_i)|}{d}\right)^r + \sum \max\{1, |A(L_i)| - 2\}\right. \\
&\quad \left. + \left(\frac{(d-1)\sigma(\mathcal{M})}{d}\right)^r\right] + 4 \\
&> \frac{1}{2}[(2 + (b-1)n^*/d)^r + ((d-1)\sigma(\mathcal{M})/d)^r] + 4 \quad \text{(by Lemma 3.1)} \\
&> \frac{1}{2}[2 + n^* + (b-1)(d-1)\sigma(\mathcal{M})/d]^r + 4 \quad \text{(by Lemma 3.1)} \\
&> \frac{1}{2}(4(d-1)\sigma(\mathcal{M}))^r + 3 \quad \text{(since } b \geq 4d+1\text{)} \\
&> \frac{1}{2}n^r + 3 \quad \text{(by (5))}.
\end{aligned}$$

As before, the desired cycle C may be obtained by modifying C^*.

Now assume $z \in \{x_0, y_0\}$, and that the notation is chosen so that $z = x_0$. By Lemma 3.6(ii), there is a path P_2 in $\mathcal{M} - y_0$ from x_0 to some $y'' \in N(y) \cap V(M_m)$ and containing no separating edge of $\mathcal{M}$ such that $|E(P_2)| \geq \frac{1}{2}(\sum\{(\frac{|A(M_i)|}{d})^r : M_i \in B_1(\mathcal{M})\}) + (\sum\{\max\{1, |A(M_i)| - 2\} : M_i \in B_2(\mathcal{M})\}) + 1$. By Lemma 3.5 there is a path P_1 in $L_{t+1} \dots L_\ell$ from c_t to d_t such that $|E(P_1)| \geq \frac{1}{2}((d-1)n^*/d)^r$. Let $C^* := (P' \cup (Q - c_td_t) \cup P_1 \cup P_2) + \{y, yx, yy''\}$. Then by applying Lemma 3.1 as in the above paragraph (by swapping the roles of L_i and M_i), we have

$$|C^*| \geq \frac{1}{2}[(2 + \sigma(\mathcal{M}))^r + ((d-1)n^*/d)^r] + 4.$$

If $(d-1)n^*/d \leq 2 + \sigma(\mathcal{M})$, then by Lemma 3.1 and because $n^* \geq \sigma(\mathcal{M}) - 2$,

$$\begin{aligned}
|C^*| &\geq \frac{1}{2}(2 + \sigma(\mathcal{M}) + (b-1)(d-1)n^*/d + 2(b-1))^r + 3 \\
&> \frac{1}{2}(4(d-1)\sigma(\mathcal{M}))^r + 3 \quad \text{(since } b \geq 4d+1\text{)} \\
&> \frac{1}{2}n^r + 3 \quad \text{(by (5))}.
\end{aligned}$$

So assume $(d-1)n^*/d \geq 2 + \sigma(\mathcal{M})$. Applying Lemma 3.1 and (5) again, we have

$$|C^*| \geq \frac{1}{2}((d-1)n^*/d + (b-1)(2 + \sigma(\mathcal{M})))^r + 4 > \frac{1}{2}(4d\sigma(\mathcal{M}))^r + 3 > \frac{1}{2}n^r + 3.$$

As before, the desired cycle C can be obtained by modifying C^*.

Case 2. $\mathcal{M}$ may be chosen so that $x \notin \{x_0, y_0\} \cap \{c_0, d_0\}$, $\{c_0, d_0\} \neq \{x_0, y_0\}$, and $\{x_0, y_0\} \subseteq V(\mathcal{H})$.

We may assume that $\{c_0, d_0\} \subseteq V(H_s)$ and $\{c_0, d_0\} \neq \{a_{s-1}, b_{s-1}\}$, and $\{x_0, y_0\} \subseteq V(H_t)$ and $\{x_0, y_0\} \neq \{a_{t-1}, b_{t-1}\}$. Note that a_{-1} and b_{-1} are not defined. We consider only the case $s \leq t$, since the case $s \geq t$ is similar. By the choice of $\mathcal{L}$ and by (5), $\sigma(\mathcal{L}) \geq \sigma(\mathcal{M}) \geq \frac{n}{4(d-2)}$.

We claim that there is a path P_0 in $\mathcal{H}$ from x to some $z \in \{c_0, d_0\} \cup \{x_0, y_0\}$ and containing no separating edge of $\mathcal{H}$ (except possibly c_0d_0 or x_0y_0) such that (a) $|E(P_0)| \geq \frac{1}{2}(|H_0|/d)^r + 1$, (b) $c_0d_0 \in E(P_0)$ or $x_0y_0 \in E(P_0)$, (c) if $c_0d_0 \in E(P_0)$, then $z \in \{x_0, y_0\}$ and $x_0y_0 \notin E(P_0)$, and (d) if $x_0y_0 \in E(P_0)$, then $z \in \{c_0, d_0\}$ and $c_0d_0 \notin E(P_0)$.

Suppose $h = 0$. We may assume $x \notin \{x_0, y_0\}$ (the case $x \notin \{c_0, d_0\}$ is symmetric). Let H_0' be obtained from H_0 by a T-transform at $\{x, x_0y_0\}$, and let x' denote the new vertex. By Theorem 1.2(b) we find a cycle C_0 in H_0' through c_0d_0 and xx' such that $|C_0| \geq \frac{1}{2}(|H_0|/d)^r + 3$. Now $P_0 := C_0 - x'$ gives the desired path.

So we may assume $h \geq 1$. Let H' be obtained from $H_1 \dots H_h$ by deleting all separating edges of $\mathcal{H}$ different from a_0b_0, c_0d_0, and x_0y_0. Note that H' is 2-connected.

Assume $s = t = 0$. Since $c_0d_0 \neq x_0y_0$, we may assume $x_0y_0 \neq a_0b_0$ (the case $c_0d_0 \neq a_0b_0$ is the same). Suppose $c_0d_0 = a_0b_0$. By Theorem 1.2(b) we find a cycle C_0 in H_0 through a_0b_0 and x_0y_0 such that $|C_0| \geq \frac{1}{2}(|H_0|/d)^r + 3$. In $H' - b_0$ we find a path P' from x to a_0. Then $P_0 := (C_0 - a_0b_0) \cup P'$ gives the desired path. So assume $c_0d_0 \neq a_0b_0$. Let H_0' be obtained from H_0 by an H-transform at $\{x_0y_0, a_0b_0\}$, and let x', a' denote the new vertices with a' subdividing a_0b_0. By Theorem 1.2(b), there is a cycle C_0 in H_0' through c_0d_0 and $a'x'$ such that $|C_0| \geq \frac{1}{2}(|H_0|/d)^r + 3$. Let P' be a path in $H' - a_0b_0$ from x to the end, say v, of $C_0 - \{a', x'\}$ adjacent to a' and avoiding $\{a_0, b_0\} - \{v\}$. Then $P_0 := (C_0 - \{a', x'\}) \cup P'$ gives the desired path.

Now assume $s = 0 < t$. Then $x_0y_0 \neq a_0b_0$. By 2-connectivity of H', let P' be a path in $H' - a_0b_0$ from x to $z \in \{a_0, b_0\}$ through x_0y_0. By choosing appropriate notation, we may let $z = a_0$. Suppose $a_0b_0 = c_0d_0$. By Theorem 1.2(c), we find a cycle C_0 in H_0 through a_0b_0 such that $|C_0| \geq \frac{1}{2}|H_0|^r + 3$. Now $P_0 := (C_0 - a_0b_0) \cup P'$ gives the desired path. So we may assume $a_0b_0 \neq c_0d_0$, and let $c_0 \notin \{a_0, b_0\}$ (by choosing appropriate notation). If $d_0 \in \{a_0, b_0\}$, then let $z' \in \{a_0, b_0\} - \{d_0\}$, and apply Theorem 1.2(a) to find cycle C_0 in $(H_0 + z'c_0) - b_0$ through a_0c_0 such that $|C_0| \geq \frac{1}{2}(|H_0|/d)^r + 2$; then $P_0 := (C_0 - a_0c_0) \cup P'$ gives the desired path. Now assume $d_0 \notin \{a_0, b_0\}$. Let H_0' be obtained from H_0 by a T-transform at $\{a_0, c_0d_0\}$, and let c' denote the new vertex. We apply Theorem 1.2(a) to find a cycle C_0 in $(H_0' + b_0c') - b_0$ through a_0c' such that $|C_0| \geq \frac{1}{2}(|H_0|/d)^r + 2$. Now $P_0 := (C_0 - c') \cup P'$ gives the desired path.

Finally, we may assume $s \geq 1$. By exactly the same argument as for (1) of Case 2 in the proof of Lemma 4.2, with a_0b_0, c_0d_0, x_0y_0 playing the roles of f, pq, vw, respectively, we find a path P' through a_0b_0 in H' from x to $z \in \{c_0, d_0\} \cup \{x_0, y_0\}$ such that P' satisfies (b), (c), and (d). By Theorem 1.2(c), we find a cycle C_0 in H_0 through a_0b_0 such that $|C_0| \geq \frac{1}{2}|H_0|^r + 3$. Now $P_0 := (C_0 - a_0b_0) \cup (P' - a_0b_0)$ gives the desired path.

Suppose $x_0y_0 \in E(P_0)$. Without loss of generality, assume $z = c_0$. By Lemma 3.6(ii) there is a path P_1 in $\mathcal{L} - d_0$ from c_0 to some $y' \in N(y) \cap V(L_\ell)$ and containing no separating edge of $\mathcal{L}$ such that $|E(P_1)| \geq \frac{1}{2}(\sum\{(|A(L_i)|/d)^r : L_i \in B_1(\mathcal{L})\}) + (\sum\{\max\{1, |A(L_i)| - 2\} : L_i \in B_2(\mathcal{L})\}) + 1$. By Lemma 3.5, there is a path P_2 from x_0 to y_0 in $\mathcal{M}$ such that $|E(P_2)| \geq \frac{1}{2}((d-1)\sigma(\mathcal{M})/d)^r + 2$. Let C^* be the cycle obtained from $(P_0 \cup P_1) + \{y, yy', yx\}$ by replacing x_0y_0 with P_2. Then, as in Case 1 (with $\sigma(\mathcal{L})$ playing the role of n^*), by Lemma 3.1, and since $\sigma(\mathcal{L}) \geq \sigma(\mathcal{M})$,

we have

$$\begin{aligned}|C^*| &\geq |E(P_0)| + |E(P_1)| + |E(P_2)| + 1\\ &> \frac{1}{2}[(2+\sigma(\mathcal{L}))^r + ((d-1)\sigma(\mathcal{M})/d)^r] + 4\\ &> \frac{1}{2}[(b-1)(d-1)\sigma(\mathcal{M})/d]^r + 3\\ &> \frac{1}{2}n^r + 3 \quad \text{(by (5))}.\end{aligned}$$

Now assume $c_0d_0 \in E(P_0)$. Without loss of generality, assume $z = x_0$. By Lemma 3.6(ii), there is a path P_1 from x_0 to some $y' \in N(y) \cap V(M_m)$ in $\mathcal{M} - y_0$ such that $|E(P_1)| \geq \frac{1}{2}(\sum\{(|A(M_i)|/d)^r : M_i \in B_1(\mathcal{M})\}) + (\sum\{\max\{1, |A(M_i)| - 2\} : M_i \in B_2(\mathcal{M})\}) + 1$. By Lemma 3.5, there is a path P_2 from c_0 to d_0 in $\mathcal{L}$ such that $|E(P_2)| \geq \frac{1}{2}(\frac{(d-1)\sigma(\mathcal{L})}{d})^r + 2$. Let C^* be the cycle obtained from $(P_0 \cup P_1) + \{y, yy', yx\}$ by replacing c_0d_0 with P_2. Then as in Case 1 and by Lemma 3.1,

$$\begin{aligned}|C^*| &\geq |E(P_0)| + |E(P_1)| + |E(P_2)| + 1\\ &\geq \frac{1}{2}[(2+\sigma(\mathcal{M}))^r + ((d-1)\sigma(\mathcal{L})/d)^r] + 4.\end{aligned}$$

If $(d-1)\sigma(\mathcal{L})/d \geq 2 + \sigma(\mathcal{M})$, then by Lemma 3.1 and by (5),

$$|C^*| > \frac{1}{2}((b-1)\sigma(\mathcal{M}))^r + 4 > \frac{1}{2}(4d\sigma(\mathcal{M}))^r + 3 > \frac{1}{2}n^r + 3.$$

Now assume $(d-1)\sigma(\mathcal{L})/d \leq 2 + \sigma(\mathcal{M})$. Then by Lemma 3.1 and (5) and because $\sigma(\mathcal{L}) \geq \sigma(\mathcal{M})$,

$$|C^*| > \frac{1}{2}((b-1)(d-1)\sigma(\mathcal{L})/d)^r + 4 > \frac{1}{2}(4(d-1)\sigma(\mathcal{M}))^r + 3 > \frac{1}{2}n^r + 3.$$

As before, the desired cycle C can be obtained by modifying C^*.

Case 3. For every choice of $\mathcal{M}$ with $\sigma(\mathcal{M}) \geq \frac{n}{4(d-2)}$, we have $x \in \{c_0, d_0\} \cap \{x_0, y_0\}$ or $\{c_0, d_0\} = \{x_0, y_0\}$.

Let $\mathcal{M}_i$, $1 \leq i \leq k$, denote the $\mathcal{HL}$-legs with $\sigma(\mathcal{M}_i) \geq \frac{n}{4(d-2)}$. Since $\sum\{\sigma(\mathcal{M}) : \mathcal{M}$ is an $\mathcal{HL}$-leg as in Case 1 or Case 2$\} \leq \frac{(d-2-k)n}{4(d-2)}$ and because $n > (4d+1)^2$, it follows from (3) and (4) that

$$\sum_{i=1}^{k} \sigma(\mathcal{M}_i) \geq (n-1) - \frac{n-1}{4} - \frac{n-1}{4} - \frac{(d-2-k)n}{4(d-2)} - 2d > \frac{n}{4}.$$

Let G_i denote the graph obtained from G by deleting all components of $(G-y) - V(\mathcal{M}_i \cap (\mathcal{H} \cup \mathcal{L}))$ not containing any vertex of $\mathcal{M}_i$. Let $z \in \{c_0, d_0\}$ such that $z = x$ if $x \in \{c_0, d_0\}$. Since we are in Case 3, $z \in V(\mathcal{M}_i)$ for $1 \leq i \leq k$. Let t_i be the number of neighbors of z in G_i different from y and not in $V(\mathcal{M}_i \cap (\mathcal{H} \cup \mathcal{L}))$. Then $t_i \geq 1$. We claim that $\sum_{i=1} t_i \leq d-1$. This is clear when $z = x$ because yx is an edge of G. Now suppose $z \neq x$. Then $\{c_0, d_0\} \subseteq V(\mathcal{M}_i)$ for all $1 \leq i \leq k$. Since z is incident with edges in both $\mathcal{H} - c_0d_0$ and $\mathcal{L} - c_0d_0$, we have $\sum_{i=1}^{k} t_i \leq d-1$.

Let $1 \leq s \leq k$ such that $\frac{|G_s|}{t_s}$ is maximum. Then $\frac{|G_s|}{t_s} \geq \frac{n}{4(d-1)}$. This follows from the following result (which can be proved by induction on k): If $\alpha_1 + \cdots + \alpha_k \geq \alpha$ and $t_1 + \cdots + t_k = m$, then $\max\{\frac{\alpha_i}{t_i} : 1 \leq i \leq k\} \geq \frac{\alpha}{m}$.

For convenience, let $\{x_s, y_s\} = V(\mathcal{M}_s \cap (\mathcal{H} \cup \mathcal{L}))$, and assume, without loss of generality, $z = x_s = c_0$. Note that $G_s^* := G_s + \{yx_s, yy_s, x_sy_s\}$ is 3-connected, $\Delta(G_s^*) \le d+1$, and any vertex of degree $d+1$ must be incident with x_sy or x_sy_s. By Theorem 1.2(a), there is a path Q_2 from y_s to y in $G_s^* - x_s$ such that

$$|E(Q_2)| \ge \frac{1}{2}\left(\frac{(d-1)|G_s|}{dt_s}\right)^r + 1 \ge \frac{1}{2}\left(\frac{n}{4d}\right)^r + 1.$$

Suppose $y_s \in V(\mathcal{H})$. By 2-connectivity of $\mathcal{H}$, there is a path Q_0 from x to y_s in $\mathcal{H}$ through c_0d_0. By Lemma 3.5, there is a path Q_1 from c_0 to d_0 in $\mathcal{L}$ such that $|E(Q_1)| \ge \frac{1}{2}(\frac{(d-1)\sigma(\mathcal{L})}{d})^r + 2$. Let $C^* := ((Q_0 - c_0d_0) \cup Q_1 \cup Q_2) + yx$. Then

$$|C^*| \ge |E(Q_1)| + |E(Q_2)| \ge \frac{1}{2}\left[\left(\frac{(d-1)\sigma(\mathcal{L})}{d}\right)^r + \left(\frac{n}{4d}\right)^r\right] + 3.$$

If $\frac{(d-1)\sigma(\mathcal{L})}{d} \ge \frac{n}{4d}$, then by Lemma 3.1 and since $b \ge 4d+1$,

$$|C^*| \ge \frac{1}{2}\left(\frac{(b-1)n}{4d}\right)^r + 3 \ge \frac{1}{2}n^r + 3.$$

Now assume $\frac{(d-1)\sigma(\mathcal{L})}{d} \le \frac{n}{4d}$. By Lemma 3.1 and since $\sigma(\mathcal{L}) \ge \sigma(\mathcal{M}) > \frac{n}{4(d-2)}$ (by (5)),

$$|C^*| \ge \frac{1}{2}\left[\frac{(b-1)(d-1)\sigma(\mathcal{L})}{d}\right]^r + 3 > \frac{1}{2}n^r + 3.$$

As before, the desired cycle C can be obtained by modifying C^*.

Thus, we may assume $y_s \notin V(\mathcal{H})$. Then $z = x$ and $y_s \in V(L_t)$ for some $1 \le t < \ell$ ($t \ne \ell$ by the choice of $\mathcal{L}$). Let $n^* := \sum_{i=t+1}^{\ell} |A(L_i)|$. Note that $n^* \le \sigma(L_{t+1} \dots L_\ell)$. By our choice of $\mathcal{L}$, $n^* \ge \sigma(\mathcal{M}) - 2$. By 2-connectivity, let Q_0 be a path from x to y_s through c_td_t in $L_1 \dots L_t$. Note that $|E(Q_0)| \ge 2$. By Lemma 3.5 there is a path Q_1 from c_t to d_t in $L_{t+1} \dots L_\ell$ such that $|E(Q_1)| \ge \frac{1}{2}(\frac{(d-1)n^*}{d})^r + 1$. Let $C^* := ((Q_0 - c_td_t) \cup Q_1 \cup Q_2) + yx$. Then

$$|C^*| \ge |E(Q_1)| + |E(Q_2)| + 2 \ge \frac{1}{2}\left[\left(\frac{(d-1)n^*}{d}\right)^r + 2 + \left(\frac{n}{4d}\right)^r\right] + 3.$$

If $\frac{(d-1)n^*}{d} \ge \frac{n}{4d}$, then by Lemma 3.1 and since $b \ge 4d+1$,

$$|C^*| \ge \frac{1}{2}\left(\frac{(b-1)n}{4d}\right)^r + 3 \ge \frac{1}{2}n^r + 3.$$

Now assume $\frac{(d-1)n^*}{d} \le \frac{n}{4d}$. Then by Lemma 3.1 and since $n^* \ge \sigma(\mathcal{M}) - 2 > \frac{n}{4(d-1)} - 2$ (by (5)),

$$|C^*| > \frac{1}{2}\left(\frac{(b-1)(d-1)n^*}{d} + 2(b-1)\right)^r + 3 > \frac{1}{2}[4(d-1)n^* + 2(b-1)]^r + 3 > \frac{1}{2}n^r + 3.$$

Again, the desired cycle C can be obtained by modifying C^*. □

6. Conclusions. We now complete the proof of Theorem 1.2.

Proof of Theorem 1.2. Let n, d, r, G be given as in Theorem 1.2. We apply induction on n. When $n = 5$, G is isomorphic to one of the following three graphs: K_5, K_5 minus an edge, or the wheel on five vertices. In each case, we can verify that Theorem 1.2 holds. So assume that $n \geq 6$ and Theorem 1.2 holds for all 3-connected graphs with at most $n - 1$ vertices. Then Theorem 1.2(a) holds by Lemma 4.1, Theorem 1.2(b) holds by Lemma 4.2, and Theorem 1.2(c) holds by Lemma 5.1. This completes the proof of Theorem 1.2. □

Our proof of Theorem 1.2 implies a polynomial time algorithm which, given a 3-connected n-vertex graph, finds a cycle of length $\frac{1}{2}n^r + 3$. When combined with the next two results [8], our proof implies a cubic algorithm.

LEMMA 6.1. *Let G be a k-connected graph where k is a positive integer. Then G contains a k-connected spanning subgraph with $O(V)$ edges; such a subgraph can be found in $O(E)$ time.*

LEMMA 6.2. *Let G be a 2-connected graph and let $e, f \in E(G)$. Then there is a cycle through e and f in G, and such a cycle can be found in $O(V)$ time.*

Lemma 6.2 is actually an easy consequence of a result in [8] which states that, in a 2-connected graph G, one can find, in $O(V)$ time, two disjoint paths linking two given vertices. Our algorithm is similar to that in [3]. Therefore, we give only an outline and omit complexity analysis.

ALGORITHM. Let G be a 3-connected graph with $\Delta(G) \leq d$, and assume $|G| \geq 5$. The following procedure finds a cycle C in G with $|C| \geq \frac{1}{2}|G|^r + 3$.

1. **Preprocessing.** Replace G with a 3-connected spanning subgraph of G with $O(|G|)$ edges.
2. We either find the desired cycle C, or we reduce the problem to Theorem 1.2 for some 3-connected graphs G_i, for which $|G_i| < |G|$ and each G_i contains a vertex which does not belong to any other G_i.
3. Replace each G_i with a 3-connected spanning subgraph of G_i with $O(|G_i|)$ edges.
4. Apply Lemma 4.1 to those G_i for which Theorem 1.2(a) needs to be applied. Apply Lemma 4.2 to those G_i for which Theorem 1.2(b) needs to be applied. Apply Lemma 5.1 to those G_i for which Theorem 1.2(c) needs to be applied.
5. Repeat steps 3 and 4 for new 3-connected graphs.
6. In the final output, replace all virtual edges by appropriate paths in G to complete the desired cycle C.

Acknowledgment. We thank the referees for their suggestions that helped improve the presentation of this paper.

REFERENCES

[1] A. BJÖRKLUND AND T. HUSFELDT, *Finding a path of superlogarithmic length*, SIAM J. Comput., 32 (2003), pp. 1395–1402.

[2] G. CHEN, Z. GAO, X. YU, AND W. ZANG, *Approximating the longest cycle problem on graphs with bounded degree*, Lecture Notes in Comput. Sci. 3595, 2005, pp. 870–884.

[3] G. CHEN, J. XU, AND X. YU, *Circumference of graphs with bounded degree*, SIAM J. Comput., 33 (2004), pp. 1136–1170.

[4] T. FEDER, R. MOTWANI, AND C. SUBI, *Approximating the longest cycle problem in sparse graphs*, SIAM J. Comput., 31 (2002), pp. 1596–1607.

[5] T. FEDER AND R. MOTWANI, *Finding a long cycle in a graph with a degree bound and a 3-cyclable minor*, manuscript, 2004.

[6] H. N. Gabow, *Finding paths and cycles of superpolylogarithmic length*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC), 2004, pp. 407–416.
[7] J. E. Hopcroft and R. E. Tarjan, *Dividing a graph into triconnected components*, SIAM J. Comput., 2 (1973), pp. 135–158.
[8] T. Ibaraki and H. Nagamochi, *A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph*, Algorithmica, 7 (1992), pp. 583–596.
[9] B. Jackson and N. C. Wormald, *Longest cycles in* 3*-connected graphs of bounded maximum degree*, in Graphs, Matrices, and Designs, R. S. Rees, ed., Dekker, New York, 1993, pp. 237–254.
[10] D. Karger, R. Motwani, and G. D. S. Ramkumar, *On approximating the longest path in a graph*, Algorithmica, 18 (1997), pp. 82–98.
[11] W. T. Tutte, *Connectivity in Graphs*, University of Toronto Press, Toronto, 1966.

SIAM J. COMPUT.
Vol. 36, No. 3, pp. 657–680

FAIRNESS MEASURES FOR RESOURCE ALLOCATION*

AMIT KUMAR† AND JON KLEINBERG‡

Abstract. In many optimization problems, one seeks to allocate a limited set of resources to a set of individuals with demands. Thus, such allocations can naturally be viewed as vectors, with one coordinate representing each individual. Motivated by work in network routing and bandwidth assignment, we consider the problem of producing solutions that simultaneously approximate *all* feasible allocations in a coordinate-wise sense. This is a very strong type of "global" approximation guarantee, and we explore its consequences in a wide range of discrete optimization problems, including facility location, scheduling, and bandwidth assignment in networks. A fundamental issue—one not encountered in the traditional design of approximation algorithms—is that good approximations in this global sense need not exist for every problem instance; there is no a priori reason why there should be an allocation that simultaneously approximates all others. As a result, the existential questions concerning such good allocations lead to a new perspective on a number of fundamental problems in resource allocation, and on the structure of their feasible solutions.

Key words. fairness, approximation algorithms, scheduling, facility location, bandwidth allocation

AMS subject classification. 68W25

DOI. 10.1137/S0097539703434966

1. Introduction.

Fair allocations. In many optimization problems, one seeks to allocate a limited set of resources to a set of individuals with demands. For example, given n users of a network, each seeking to transmit data between some pair of nodes, we must decide how to route these connection requests and allocate a portion of the available bandwidth to each. Much work on exact and approximate algorithms for such resource allocation problems has sought allocations that optimize a particular aggregate figure of merit. In the case of routing connections, for example, we could seek a solution that maximizes the minimum amount of bandwidth allocated to any request, or the total bandwidth allocated to all requests.

But these types of problems genuinely involve n distinct objectives—the demands of each individual—and a rich set of issues emerges when we adopt this perspective, treating the feasible solutions as vectors with a separate coordinate to represent the allocation to each individual. In the networking literature, this approach underlies the notion of *max-min fairness* [1, 15, 16], a widely used mechanism which produces an allocation for a fixed set of routes that is *lexicographically maximum*: the minimum bandwidth assigned to any connection is maximized; subject to this, the minimum bandwidth assigned to any of the remaining connections is maximized; and so forth.

*Received by the editors September 19, 2003; accepted for publication (in revised form) February 7, 2006; published electronically October 10, 2006. A preliminary abstract of this work appeared in the *Proceedings of the* 41*st Annual IEEE Symposium on Foundations of Computer Science* [19].
http://www.siam.org/journals/sicomp/36-3/43496.html

†Department of Computer Science and Engineering, Indian Institute of Technology, Delhi 110016, India (amitk@cse.iitd.ernet.in). This work was done while the first author was at Cornell University, and was supported in part by the second author's ONR Young Investigator Award.

‡Department of Computer Science, Cornell University, Ithaca, NY 14853 (kleinber@cs.cornell.edu). This author's work was supported in part by a David and Lucile Packard Foundation Fellowship, an ONR Young Investigator Award, and NSF Faculty Early Career Development Award CCR-9701399.

Such a vector is often called the *max-min fair vector*. The same notion was formulated in the context of network flow problems by Megiddo [21].

There are cases in which it is computationally intractable to determine the max-min fair vector of allocations. In such settings, Kleinberg, Rabani, and Tardos [17] proposed the notion of a *coordinate-wise approximation*: We say that a vector X is a coordinate-wise c-approximation to a vector X' if for each j, the jth largest coordinate of X is at least $1/c$ times the values of the jth largest coordinate of X'. One can then look for efficiently computable vectors X that form good approximations in this sense to the max-min fair vector.

Recently, Goel, Meyerson, and Plotkin [9] developed algorithms for on-line path selection and bandwidth allocation satisfying a stronger notion of fairness. They were motivated by the following problem: Given a set of requests in a network, we would like to allocate bandwidth to them so that the allocation vector is max-min fair and the total throughput (i.e., the sum of the values in the vector) is maximized. But it is easy to construct examples showing that a max-min fair solution can have low total throughput and, conversely, that a solution maximizing the total throughput can be quite far from any max-min fair solution.

One way to get around this problem is to take the average of the max-min fair allocation and the maximum total throughput allocation. But examples in [9] show that even such an allocation vector may not be good in terms of fairness. This led them to define a stronger notion of fairness: Compute an allocation vector X that is a coordinate-wise c-approximation to *every* allocation vector. Such a vector satisfies the following natural and strong type of guarantee: For any value of m, if there is any allocation in which m users each get bandwidth at least b, then at least m users get bandwidth at least b/c under allocation X. We will therefore refer to X as a *global c-approximation* for this instance.

A simple example from [9] shows that the max-min fair vector does not perform well under this measure of quality. Suppose our network is a single path P; for an arbitrary number k, we have k "long" requests that need to use all the links of P, and we have k^2 "short" requests, each of which needs to use only a distinct edge of P. We wish to assign a bandwidth value to each connection in such a way that the sum of the values assigned to any set of connections passing through a common edge is at most 1. Then the max-min fair vector X^* assigns bandwidth $1/(k+1)$ to each request, while the vector X_0 that maximizes the sum of all values assigns 1 unit to each short connection and 0 units to each long connection. Note that X^* is only a coordinate-wise $(k+1)$-approximation to X_0, while there is no c for which X_0 is a coordinate-wise c-approximation to X^*. However, as we will see later, there is a vector X for this instance which is a coordinate-wise $O(\log k)$-approximation to *every* feasible vector of bandwidths; in other words, it exhibits the following natural and strong type of guarantee: For any value of m, if there is any allocation in which m users each get bandwidth at least b, then at least m users get bandwidth at least $b/O(\log k)$ under allocation X. We will therefore refer to X as a *global $O(\log k)$-approximation* for this instance.

The present work. In this work, we develop algorithms that produce allocations with this stronger type of guarantee: global approximation to all possible allocations. We consider a wide range of problems that can be cast in the framework of resource allocation, with an emphasis on three particular domains.

(i) *Bandwidth allocation.* We consider a version of the setting discussed above, in which there are n given paths in a network, and we must assign bandwidth to each one in a way that respects capacity constraints. Such an assignment

induces an *allocation vector* of length n, whose ith coordinate is the amount of bandwidth assigned to path i.

(ii) *Scheduling.* Given a set of n jobs that need to be processed, a schedule for these jobs induces a *completion time vector*, whose ith coordinate is the completion time of job i. Using the above notions, we can compare different completion time vectors in a coordinate-wise sense.

(iii) *Facility location.* Given a set of n demand points in a metric space, opening k facilities to serve these points induces a *distance vector* of length n: The ith coordinate is the distance from the ith demand point to its nearest facility. Again, we can compare the distance vectors induced by different sets of k facilities, so as to find the "fairest" solution.

What we find particularly interesting is that the notion of approximation actually plays a fundamentally different role in this type of allocation problem than it does in standard approximation algorithms for NP-complete problems. In the traditional design of approximation algorithms, the existence of an optimum (or of optima) is not in question; rather, we lack the computational power to find these optima. The definition of global approximation, on the other hand, comes with the following crucial point: There is no a priori guarantee that a good global approximation exists for a particular instance of a problem. Indeed, global 1-approximations—allocations that are as good as every allocation in every coordinate of a sorted order—turn out to be quite rare and often indicate something striking in the structure of the instance.

Thus, our study of this notion has two related components: an existential aspect, in which we try to determine whether good global approximations exist for all instances of a particular problem, and an algorithmic aspect, in which we look for efficient ways to compute global approximations. Each is important for our understanding of this issue, existential results illustrating the kinds of fair allocations that are achievable, even when we do not yet know of corresponding efficient algorithms that produce them.

Formulating the problem. We now introduce some precise notation for expressing the basic questions and results. All three types of problems above have the property that a solution to the problem induces a vector of length n; thus, for each instance $\mathcal{I}$ of such a problem, we have a set $V(\mathcal{I})$ consisting of all vectors that are induced by some feasible solution to the instance. Now, for a vector $X \in V(\mathcal{I})$, we define $\overrightarrow{X}$ to be the vector in which the coordinates of X have been reordered so that they are in nondecreasing order, and we define $\overleftarrow{X}$ to be the vector in which the coordinates of X have been reordered so that they are in nonincreasing order. For example, if $X = \{4, 1, 5, 10, 3\}$, then $\overrightarrow{X} = \{1, 3, 4, 5, 10\}$, $\overleftarrow{X} = \{10, 5, 4, 3, 1\}$. For two vectors $X, Y \in V(\mathcal{I})$, we write $X \prec_c Y$ if $X_i \leq Y_i$ for all i. Note that for purposes of coordinate-wise comparison, we have $\overrightarrow{X} \prec_c \overrightarrow{Y}$ if and only if $\overleftarrow{X} \prec_c \overleftarrow{Y}$—it clearly does not matter for the comparison whether we write the small values at the left or the right. Note an important fact: Often the coordinates of vectors in $V(\mathcal{I})$ correspond to individual users in the problem instance (e.g., in the bandwidth allocation problem, each coordinate may correspond to a request); when we compare vectors $\overrightarrow{X}$ and $\overrightarrow{Y}$ (or $\overleftarrow{X}$ and $\overleftarrow{Y}$), we are not comparing the allocations in the two solutions to the same user. But such a comparison makes sense because according to our notion of fairness we would like to compare, for example, the allocation to the kth least favored users in the two solutions, respectively.

As is common in discussing approximation, our terminology will be slightly different depending on whether we are discussing a minimization problem or a maximization problem (note that both arise in our list of problems above). For an allocation vec-

tor X in a minimization problem, we define $c(X)$ to be the infimum of α such that $\overrightarrow{X} \prec_c \alpha \overrightarrow{Y}$ for all $Y \in V(\mathcal{I})$. For an allocation vector X in a maximization problem, we define $c(X)$ to be the infimum of α such that $\overrightarrow{Y} \prec_c \alpha \overrightarrow{X}$ for all $Y \in V(\mathcal{I})$. In both cases, this can be informally viewed as the *global approximation ratio of* X: It is the smallest α for which $\overrightarrow{X}$ is an α-approximation, in the natural coordinate-wise sense, to every vector $Y \in V(\mathcal{I})$. The best global approximation ratio achievable on the instance $\mathcal{I}$ is then defined as

$$c^*(\mathcal{I}) = \inf_{X \in V(\mathcal{I})} c(X).$$

For several types of problems, we will find that good global approximations do not necessarily exist, but in many of these cases there will always exist solutions satisfying a weaker global guarantee, in which we compare the *prefix sums* of one allocation to the prefix sums of other allocations [17]. Thus, for a vector $X \in V(\mathcal{I})$, we define $\sigma(X)$ to be the vector whose ith coordinate is $\sum_{j \le i} X_j$. We say that $X \prec_s Y$ if $\sigma(X) \prec_c \sigma(Y)$. Now for a minimization (resp., maximization) problem, we define $s(X)$ to be the infimum of α such that $\overleftarrow{X} \prec_s \alpha \overleftarrow{Y}$ (resp., $\overrightarrow{Y} \prec_s \alpha \overrightarrow{X}$) for all $Y \in V(\mathcal{I})$. In this case, we say that X is a *global* α*-approximation under prefix sums*. In a bandwidth allocation problem, for example, we can view such a vector X as conveying the following guarantee: For any m, if at least m users receive an *average bandwidth* of b in X, then at least m users receive an average allocation of at most αb in any allocation. Note, crucially, that in a minimization (resp., maximization) problem, we begin computing prefix sums with the largest (resp., smallest) coordinates. Finally, for an instance $\mathcal{I}$, we define the best possible global approximation ratio under prefix sums as

$$s^*(\mathcal{I}) = \inf_{X \in V(\mathcal{I})} s(X).$$

Note that since $s(X) \le c(X)$ for any vector X, we have $s^*(\mathcal{I}) \le c^*(\mathcal{I})$ for any instance $\mathcal{I}$. The notion of s^* as an approximation ratio has also been used by Bhargava, Goel, and Meyerson [2] and Goel, Meyerson, and Plotkin [10], where they denote this by *approximate majorization*.

Sometimes we will have an instance $\mathcal{I}$ and want to evaluate the quality of an allocation $X \notin V(\mathcal{I})$. Thus, we allow the definitions of $c(X)$ and $s(X)$, exactly as written above, to apply to vectors $X \notin V(\mathcal{I})$; there is no difficulty in doing this, and the meaning will be clear from the context.

An illustrative example. Before discussing our results, we consider a very simple example to illustrate how the optimal allocations can differ qualitatively under the c^* and s^* measures. Suppose we have n users of a network, each trying to transmit data across a single shared link of unit capacity. (Thus the underlying network consists of two nodes joined by a single edge.)

First, we claim that the bandwidth allocation vector X in which each coordinate is equal to $1/n$ satisfies $s(X) = 1$. Indeed, in any other vector Y, and for any $j \le n$, the j smallest coordinates of Y must add up to at most j/n, while in X they add up to exactly j/n.

However, the uniform allocation vector X has a very large global approximation ratio under coordinate-wise comparison; specifically, $c(X) = n$. To see this, consider the vector Y_1 in which the first coordinate is 1 and all others are 0. A much better global approximation ratio is achieved by the vector X', whose jth coordinate is equal

to $1/jH_n$, where $H_n = \sum_{i=1}^{n} 1/i$. We claim that $c(X') = H_n$; for in any other vector Y, the jth largest coordinate is at most $1/j$. In fact, $c^*(\mathcal{I}) = H_n$ for this simple instance $\mathcal{I}$: Any vector Y must have some coordinate, say the jth largest, that is at most the value of corresponding coordinate in X', and hence this coordinate is H_n times smaller than the jth largest coordinate of the vector assigning $1/j$ to j of the coordinates, and 0 to all others.

In some sense, of course, the optimal allocation X under the s^* measure has a more intuitive meaning—equal division of resources among all individuals—than the optimal allocation X' under the c^* measure. However, it is interesting that the allocation X' also corresponds to a well-studied distribution of wealth: It is an instance of the *Pareto distribution* [22], one of the simplest types of allocations to follow an inverse power law. This connection can also be justified at an intuitive level. Pareto distribution basically states that a larger portion of the wealth in a society is owned by a smaller percentage of people in the same society. The allocation X' must also follow a similar principle—there are allocations which allocate all the resources to a few coordinates, but X' has to be close to such solutions under the c^* measure.

2. Overview of results.

Bandwidth allocation. We consider the problem that underlies the Hayden–Jaffe model of *max-min fairness* [1, 15, 16]. We are given a graph $G = (V, E)$ and paths $P_1, \ldots, P_n$ in this graph, representing users transmitting data. We must assign a *rate* x_i to each path P_i so that the following *capacity condition* is satisfied: If $P_{i_1}, \ldots, P_{i_t}$ all pass through a common edge, then $\sum_{j=1}^{t} x_{i_t} \leq 1$. Thus, a solution to an instance of this problem is a vector X of rates. We will refer to this problem as the *bandwidth allocation problem.* Note that our capacity condition implies that all links have the same capacity (set to 1); we can obtain bounds for the more general case where edges have different capacities but do not go into the details here.

For an instance $\mathcal{I}$, let U denote the *congestion* of the set of paths: the maximum number of paths that pass through any single edge. Note that $U \leq n$. Although Goel, Meyerson, and Plotkin [9] considered a problem in which both the paths and the rates had to be chosen, their on-line algorithm can be applied to this setting with fixed paths as well; it shows that $c^*(\mathcal{I}) = (\log^2 |V| \log U)$ for every instance $\mathcal{I}$.

Here, we are not restricted to using on-line algorithms. Nevertheless, there are instances where a constant-factor global approximation does not exist, even in the sense of prefix sums. We can show the following (we omit the details in this version):

- For arbitrarily large U, there exist instances $\mathcal{I}$ of the bandwidth allocation problem with congestion U for which $s^*(\mathcal{I}) = \Omega(\frac{\log U}{\log \log U})$.

Since $c^*(\mathcal{I}) \geq s^*(\mathcal{I})$ for every instance, this is a lower bound for c^* as well.

However, we can produce upper bounds that nearly match this lower bound. Specifically,

- $s^*(\mathcal{I}) \leq 2\lceil \log U \rceil + 2$ for every instance $\mathcal{I}$ of the bandwidth allocation problem.

We can produce a vector of rates X achieving this bound via a polynomial-time algorithm. More strongly, we can show that the same bound holds for the c^* measure:

- $c^*(\mathcal{I}) \leq 2\lceil \log U \rceil + 2$ for every instance $\mathcal{I}$ of the bandwidth allocation problem.

Here, however, we do not have an accompanying polynomial-time algorithm.

One can obtain strong bounds on s^* for certain related problems. For the problem of single-source fractional flow to each of n sinks—where we consider the vector of flow values to each sink—Megiddo showed that there is a maximum flow that induces the max-min fair vector of flows to the sinks [21]. By adapting some of the analysis from his proof of this, one can in fact show that $s^*(\mathcal{I}) = 1$ for all instances of this problem.

For the uniform load balancing problem considered by Kleinberg, Rabani, and Tardos, one can extend the analysis of [17] to show that $s^*(\mathcal{I}) = 1$ for all instances $\mathcal{I}$.

Scheduling problems. There are a large number of possible scheduling problems, but we can begin with some general ideas. First, in all the scheduling problems we will consider, a solution S induces a vector X_S of completion times. Two traditional objective functions are the *makespan*—the first coordinate in $\overleftarrow{X_S}$—and the sum of all completion times. A sequence of papers beginning with Stein and Wein [25] have shown that there always exist solutions to many types of scheduling problems that are good approximations to both objectives simultaneously. The algorithm of [25] does not provide a simultaneous approximation to all prefix sums of $\overleftarrow{X_S}$, however.

We follow the standard notation for scheduling problems proposed by Graham et al. [13]. Under this notation, a scheduling problem is usually denoted by three parameters $\alpha|\beta|\gamma$, where α designates the number of machines (and whether they are identical or different); β generally designates special conditions such as precedence constraints, release dates, or fixed processing times; and γ denotes an objective function such as the makespan or the sum of completion times. In our formulations, γ will be set equal to the symbol *all*, to denote the fact that we are considering the full vector of completion times, rather than an aggregate objective function.

There are approximation algorithms for scheduling that perform coordinate-by-coordinate comparison with respect to the completion time vector of the optimal schedule. If this analysis implicitly holds with respect to *any* schedule, rather than just the optimal one, then one is, in fact, giving a bound for c^*. The most basic example of this is Smith's rule for 1-processor scheduling, in which jobs are scheduled in nondecreasing order of processing time [24] $(1||all)$. The optimality proof for this schedule in fact shows that $c(X_S) = 1$. Also, for 1-processor scheduling with release dates $(1|r_j|all)$, the analysis of Phillips, Stein, and Wein [23] shows that their schedule S achieves $c(X_S) \leq 2$; for the case of $m > 1$ identical parallel machines with release dates $(P|r_j|all)$, their schedule achieves $c(X_S) \leq 3$.

Here we focus on problems with precedence constraints: We have a set J of n jobs, with a partial order $<_J$ representing dependencies, and must produce a schedule in which each job must end before the jobs that depend on it can begin. For many types of precedence-constrained scheduling problems, Hall et al. [14] produced a coordinate-wise $O(1)$-approximation to an optimal vector of *fractional completion times*, corresponding to a linear programming relaxation. However, their analysis does not, in fact, yield a global $O(1)$-approximation, for we show the following:

- For every number α, there exists an instance $\mathcal{I}$ of precedence-constrained 1-processor scheduling $(1|prec|all)$ in which the optimal vector of fractional completion times X in the linear programming formulation of Hall et al. does not satisfy $c(X) \leq \alpha$.

Thus, it is natural to ask whether there is a constant upper bound on $c^*(\mathcal{I})$ for all instances of this problem or its more general variants.

We show the following general result:

- $c^*(\mathcal{I}) \leq 4$ for all instances $\mathcal{I}$ of precedence-constrained scheduling with release dates on m identical parallel machines $(P|prec, r_j|all)$ for any $m \geq 1$.

Interestingly, our construction of the schedule S that achieves $c(X_S) \leq 4$ does not yield a polynomial-time algorithm even in the 1-processor case; in fact we show that it may be a difficult problem to obtain such a polynomial-time algorithm. Specifically, we show that any polynomial-time algorithm producing a schedule S that satisfies $c(X_S) = O(1)$ for instances of precedence-constrained 1-processor scheduling

$(1|prec|all)$ implies a polynomial-time constant-factor approximation algorithm for the *densest subgraph problem*: Given a graph G and a number k, find an induced k-node subgraph with the maximum possible number of edges. For this problem, the current best approximation ratios have the form $O(k^c)$ for a constant $0 < c < 1$ [18].

Interestingly enough, we show that there is a polynomial-time algorithm which produces a schedule S such that $s(X_S) = O(1)$ for any instance of $P|prec, r_j|all$.

We consider the special case of precedence-constrained scheduling in which there are no release dates and all jobs have unit processing times $(P|p_j = 1, prec|all)$. We show that

- $c^*(\mathcal{I}) = 1$ for every instance $\mathcal{I}$ of precedence-constrained 2-processor scheduling with unit-size jobs.

It is not difficult to show that there are instances with $m \geq 3$ machines for which $c^*(\mathcal{I}) > 1$, and a simple greedy algorithm can be used to establish the following:

- $c^*(\mathcal{I}) \leq 2$ for every instance $\mathcal{I}$ of precedence-constrained m-processor scheduling with unit jobs, for every m.

This dichotomy between $m = 2$ and $m \geq 3$ forms an interesting parallel to the fact that the 2-processor case is known to be polynomially solvable [5, 7], while the complexity of the m-processor case for fixed $m \geq 3$ is an open question.

As a final note, one can view traveling salesman problems within this perspective on scheduling, since tours yield vectors of *arrival times* at each city. In this context, the approximation of the minimum latency problem by k-trees [3, 11] shows that there is a constant upper bound on $c^*(\mathcal{I})$ for all instances of the traveling salesman problem.

Facility location problems. Finally, we consider facility location problems. Suppose we have a metric space M, a set $P = \{p_1, \ldots, p_n\}$ of *demand points* in M, and a set F of n' *candidate facility points*. The *k-center problem* and the *k-median problem* are two classical facility location problems. The first asks, Choose k facilities so that the maximum distance from any point to its nearest facility is minimized; the second asks, Choose k facilities so that the total distance from all points to their nearest facility is minimized.

Guided by the present framework, we say that opening facilities at locations $F' \subseteq F$ induces a *distance vector* $X_{F'}$ whose ith coordinate is $d(p_i, F')$, the distance from p_i to its nearest facility in F'. Thus, when we speak of a *k-facility instance* $\mathcal{I}$, we mean a metric space M, with sets P and F, and we will compare solutions to the set $V(\mathcal{I})$ of all vectors $X_{F'}$ such that $|F'| = k$. Now, the k-center problem seeks to minimize the first coordinate of $\overleftarrow{X_{F'}}$, while the k-median problem seeks to minimize the sum of all coordinates in $X_{F'}$. What can be said about global approximations for arbitrary k-facility instances?

First we consider the case $k = 1$. We show that

- $s^*(\mathcal{I}) \leq 4$ for every instance $\mathcal{I}$ of the 1-facility location problem.

The c^* measure is trickier. Consider an instance with a set P of five points in the plane—two at the ends of a very short line segment, and three at the corners of a moderately sized equilateral triangle, arbitrarily far away. If $F = P$, it is easy to see that such an instance can be constructed in which c^* is arbitrarily large. By extending this idea, we can show the following:

- For any number α, there are 1-facility instances $\mathcal{I}$ for which no set F' of fewer than $\log n$ facilities achieves $c(X_{F'}) \leq \alpha$.

At the same time, this lower bound is essentially tight:

- For every 1-facility instance $\mathcal{I}$, there is a set F' of at most $1 + \log n$ facilities for which $c(X_{F'}) \leq 3$.

Note that here we are comparing the distance vector $X_{F'}$, for a set F' of $\leq 1 + \log n$ facilities, to all possible distance vectors obtained by opening a *single* facility. Also, we note that our two upper bounds here are achieved by polynomial-time algorithms.

Now consider a larger value of k. If there were a constant bound on $s^*(\mathcal{I})$ for every k-facility instance $\mathcal{I}$, it would imply that in every instance there is a set of k facilities that provides a simultaneous constant-factor approximation for the k-center and k-median objective functions. But this is not true. Consider an instance with a set P of $2m + 1$ points in the plane: m are clustered in a small circle, m more are clustered in a nearby small circle, and one is very far away from all others. If $F = P$ and $k = 2$, then we see that any good solution for the k-median problem must put one facility in each circle, while any good solution for the k-center problem must put one facility near the far-away point.

However, we can prove the following upper bound:

- For every k-facility instance $\mathcal{I}$, there is a set F' of $O(k \log n + \varepsilon^{-1})$ facilities for which $s(X_{F'}) \leq 3 + \varepsilon$. Moreover, we can compute an F'' of this size in polynomial time for which $s(X_{F''})) \leq 9 + \varepsilon$.

We have a lower bound showing this is almost tight:

- For every α, there are k-facility instances where no set F' of fewer than $\Omega(k \frac{\log(n/k)}{\log \alpha})$ facilities can achieve $s(F') \leq \alpha$.

For the c^* measure, we can show that there are essentially no nontrivial upper bounds when $k > 1$:

- For every α, there are 2-facility instances for which no proper subset $F' \subseteq F$ can achieve $c(F') \leq \alpha$.

There has been work in discrete location theory on defining notions of "fairness" in facility location problems [6, 20]. Most of the measures to arise from this previous work are concerned with making the distance vector approximately uniform in some cases relative to a priori weights on the points, rather than optimizing the types of approximation measures we are considering here.

3. Bandwidth allocation. Recall that in our bandwidth allocation problem from section 1, we have a graph $G = (V, E)$ and a set $\mathcal{P}$ of paths $P_1, \ldots, P_n$ in this graph. We wish to assign a vector X of rates to these paths, respecting the (unit) capacities in the graph. For a set of paths $\mathcal{P}' \subseteq \mathcal{P}$, we use $\nu(\mathcal{P}')$ to denote their congestion; we write $U = \nu(\mathcal{P})$.

We first show the following theorem.

THEOREM 3.1. *$s^*(\mathcal{I}) \leq 2\lceil \log U \rceil + 2$ for every instance $\mathcal{I}$ of the bandwidth allocation problem. Further, given an instance $\mathcal{I}$, a solution X satisfying $s(X) \leq 2\lceil \log U \rceil + 2$ can be found in polynomial time.*

Proof. We describe an algorithm that constructs a vector X of rates so that $s(X) \leq 2\lceil \log U \rceil + 2$.

We call a set of paths $\mathcal{P}'$ a *block* if they all contain a common edge e; we call e a *blocking edge* of $\mathcal{P}'$. We divide our algorithm into $1 + \lceil \log U \rceil$ phases. We say that two blocks are *disjoint* if they do not contain a common path. In phase $i = 0, 1, \ldots, \lceil \log U \rceil$, we find a maximal collection of disjoint blocks, each of which has size greater than $\frac{U}{2^{i+1}}$. This can be easily done in a greedy manner—first find a block of size greater than $\frac{U}{2^{i+1}}$, delete all the paths in this block, and iterate this procedure as long as possible. For every path P in one of these blocks $\mathcal{P}'$ we allocate it a rate of $\frac{1}{|\mathcal{P}'|}$. We then delete all these paths but—at this point in the construction—do not reduce the capacity of any edges.

We observe that the following invariant is maintained by this algorithm: At the end of phase $i \leq \lceil \log U \rceil$, the overall congestion of the remaining paths is at most $\frac{U}{2^{i+1}}$. Thus after phase $1 + \lceil \log U \rceil$, all paths have been assigned a rate.

Let X_0 denote the resulting vector of rates. Although X_0 does not respect the capacities, it is useful to compare it to feasible rate vectors. Let $\mathcal{P}_1, \ldots, \mathcal{P}_r$ be the set of blocks found by the above algorithm in descending order of size. Consider any feasible rate vector Y. For any i, let $X_0|\mathcal{P}_i$ and $Y|\mathcal{P}_i$ denote the vectors X_0 and Y restricted to coordinates corresponding to paths in $\mathcal{P}_i$. We claim that $\overrightarrow{Y|\mathcal{P}_i} \prec_s \overrightarrow{X_0|\mathcal{P}_i}$. Indeed, the rates assigned by Y to paths in $\mathcal{P}_i$ must sum to at most one—since they all pass through the blocking edge e—and so the j smallest coordinates in $\overrightarrow{Y|\mathcal{P}_i}$ must sum to at most $j/|\mathcal{P}_i|$. X_0 assigns rates to the paths in $\mathcal{P}_i$ so that the sum of the j smallest coordinates is exactly $j/|\mathcal{P}_i|$.

Now, $\overrightarrow{X_0}$ consists of vectors $\overrightarrow{X_0|\mathcal{P}_1}, \overrightarrow{X_0|\mathcal{P}_2}, \ldots, \overrightarrow{X_0|\mathcal{P}_r}$ concatenated in this order. Consider the vector Y with its coordinates arranged in this same order. Each prefix sum of $\overrightarrow{X_0}$ is at least as large as the corresponding prefix sum of Y in this order, and if we now rearrange the coordinates of Y so that they are in ascending order, the prefix sums of Y cannot increase. Thus $\overrightarrow{Y} \prec_s \overrightarrow{X_0}$.

Now we claim that the rate vector X obtained by dividing each coordinate of X_0 by $2\lceil \log U \rceil + 2$ respects all the capacity conditions; this will complete the proof. Indeed, to prove this, we need only show that the total of all rates assigned to paths using edge e in a given phase i is at most 2. At the start of phase i, the congestion is at most $\frac{U}{2^i}$. Each set of paths $\mathcal{P}'$ which is selected in this phase has congestion at least $\frac{U}{2^{i+1}}$; thus, all paths that get assigned a rate in this phase get rate at most $\frac{2^{i+1}}{U}$. Hence at most 2 extra units of rate are assigned in this phase to all paths using e. □

Note that the procedure in the theorem above does not give an allocation X for which $c(X)$ is logarithmic. Indeed consider an instance consisting of a single edge and n paths using it. The procedure in Theorem 3.1 will allocate $1/n$ units of rate to each of the n paths—call this allocation X. Clearly, $c(X) = n$.

We now prove the following stronger statement. It, too, is based on a construction in which we allocate bandwidth according to an exponentially decaying sequence; however, for this construction, we do not know how to run each iteration efficiently, since it involves the solution of an independent set problem.

THEOREM 3.2. *$c^*(\mathcal{I}) \leq 2\lceil \log U \rceil + 2$ for every instance $\mathcal{I}$ of the bandwidth allocation problem.*

Proof. We give a procedure for assigning rates to paths. Initially, let $\mathcal{P}_0 = \mathcal{P}$. Let $\mathcal{Q}_0 \subseteq \mathcal{P}_0$ be a maximum cardinality subset of paths of congestion 1. Each path in $\mathcal{Q}_0$ is assigned a rate of 1, and we define $\mathcal{P}_1 = \mathcal{P}_0 \setminus \mathcal{Q}_0$. Now suppose we have a set of paths $\mathcal{P}_i$, $i \geq 1$. Define $\mathcal{Q}_i \subseteq \mathcal{P}_i$ to be a maximum cardinality subset of congestion at most 2^i. Each path in $\mathcal{Q}_i$ is assigned a rate of $\frac{1}{2^i}$, and we define $\mathcal{P}_{i+1} = \mathcal{P}_i \setminus \mathcal{Q}_i$. We define these sets of paths for all i, $0 \leq i \leq \lceil \log U \rceil$; it is clear that after the end of this process, every path gets assigned a rate. Let X^* denote the rate vector produced (note that it does not necessarily satisfy the capacity constraints) and let n_i denote $|\mathcal{Q}_i|$.

Let Y be any vector of rates that satisfies the capacity constraints. We claim that (1) $\overleftarrow{Y} \prec_c 2\overleftarrow{X^*}$.

To prove this, we fix an i, $0 \leq i \leq \log U$. Let $\mathcal{Q}'$ denote the set of paths which get rate at least $\frac{1}{2^i}$ in Y. Define $n' = |\mathcal{Q}'|$. Clearly, the congestion of $\mathcal{Q}'$ is at most 2^i. We claim that $n_0 + \cdots + n_i \geq n'$. Indeed, while selecting $\mathcal{Q}_i$, we must have considered the paths in $\mathcal{Q}' - (\mathcal{Q}_1 \cup \cdots \cup \mathcal{Q}_{i-1})$.

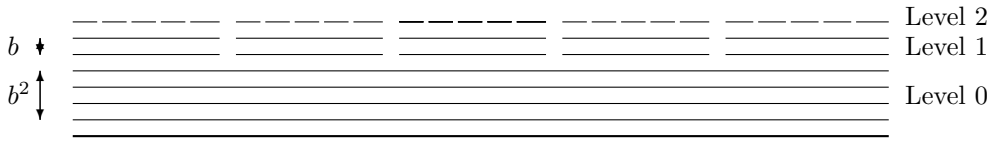


FIG. 1. *The instance when* $k = 2$.

Since the $(n_0 + \cdots + n_i)$th coordinate of $\overleftarrow{X^*}$ is 2^{-i}, it is at least as large as the $(n_0 + \cdots + n_i + 1)$st coordinate of $\overleftarrow{Y}$. Now consider a coordinate j. If $j \leq n_0$, then $\overleftarrow{X^*}_j \geq \overleftarrow{Y}_j$. Otherwise, there is some i so that $n_0 + n_1 + \cdots + n_i + 1 \leq j \leq n_0 + n_1 + \cdots + n_{i+1}$, and $\overleftarrow{X^*}_j = \frac{1}{2^{i+1}}$. But $\overleftarrow{Y}_j \leq \overleftarrow{Y}_{n_0+\cdots+n_i+1} \leq \overleftarrow{X^*}_{n_0+\cdots+n_i} = \frac{1}{2^i}$. Thus, $\overleftarrow{Y}_j \leq 2\overleftarrow{X^*}_j$; since this applies to an arbitrary coordinate j, we have $\overleftarrow{Y} \prec_c 2\overleftarrow{X^*}$.

We also claim the following:

(2) Given an edge e, the total rate assigned in X_0 to paths containing e is at most $1 + \lceil \log U \rceil$.

To see this, note that in each phase at most one additional unit of rate can be assigned to paths passing through a single edge e, and thus (2) follows.

Thus, if we define the rate vector X by dividing each coordinate of X_0 by $1 + \lceil \log U \rceil$, we obtain the theorem. □

We now show that the above result is almost optimal.

THEOREM 3.3. *For arbitrarily large U, there exist instances $\mathcal{I}$ of the bandwidth allocation problem with congestion U for which $s^*(\mathcal{I}) = \Omega(\frac{\log U}{\log \log U})$.*

Proof. The construction involves a number of parameters defined in terms of U; the relationships among these parameters will be specified as needed. We define an instance $\mathcal{I}$ as follows. Let k be a constant to be specified later in terms of U. We define constants $b = k + 1$, $d = (k + 1)^2$, and $a_i = d^{2i}$ for $i = 0, 1, \ldots, k$. The underlying graph G is a simple path on $n = a_k$ vertices. The set of paths $\mathcal{P}$ is partitioned into $k+1$ sets of paths, which we refer to as *level* 0 through *level* k. For each r, the paths in level r are grouped into a_r edge-disjoint *bundles* of b^{k-r} identical paths, each having length n/a_r; the union of these bundles covers the entire graph G. Each path in level $r + 1$ is entirely contained in some path in level r. Note that

$$U = b^k + b^{k-1} + \cdots + b + 1 = \frac{(k+1)^{k+1} - 1}{k},$$

and hence $k = \Theta(\frac{\log U}{\log \log U})$. For a schematic example of the construction (with smaller constants), see Figure 1.

Let X be an assignment of rates to paths. We define $\tilde{X}$ to be the vector obtained from X by arranging the coordinates as follows. The coordinates for paths in the same bundle occur contiguously in $\tilde{X}$, beginning with level 0, then all the bundles in level 1, then all the bundles in level 2, and so on. Moreover, the bundles in a fixed level r are arranged so that if bundle β occurs before bundle β', then the sum of the rates of paths in β is at most the sum of the rates of paths in β'.

We set $c = k/3$ and suppose that $s(X) \leq c$. First observe that $\sigma(\overrightarrow{X}) \prec_c \sigma(\overrightarrow{\tilde{X}})$. Indeed, the ith coordinate in $\overrightarrow{X}$ is the sum of the i lowest values in X, and so it must be smaller than the corresponding coordinate in $\overrightarrow{\tilde{X}}$. It follows that for any assignment Y of rates to paths in $\mathcal{P}$, $\sigma(\overrightarrow{Y}) \prec_c c\sigma(\tilde{X})$. Now consider $k + 1$ different assignments of rates to the paths. Y_0 assigns $(b^k + b^{k-1} + \cdots + 1)^{-1}$ to all paths. Similarly, Y_r,

for $0 < r \leq k$, assigns rate $(b^{k-r} + \cdots + 1)^{-1}$ to all paths in levels $r, r+1, \ldots, k$ and 0 to paths in levels $0, \ldots, r-1$.

The fact that $\sigma(\overrightarrow{Y}) \prec_c c\sigma(\tilde{X})$ for all vectors Y implies that the vector $\tilde{X}$ assigns each path in level 0 a rate of at least $c^{-1}(b^{k-r} + \cdots + 1)^{-1}$. Now consider the vector Y_r; let p_r denote the total number of paths in levels 0 through $r-1$, and define a constant $\ell_r = d^{2r-1}$. The coordinates of Y_r for the bundles in levels 0 to $r-1$ are 0. The sum of the next $\ell_r b^{k-r}$ coordinates in $\overrightarrow{Y_r}$ is equal to $\ell_r b^{k-r}(b^{k-r} + \cdots + 1)^{-1}$. Thus the sum of the first $p_r + \ell_r b^{k-r}$ coordinates in $\overrightarrow{Y_r}$ is $\ell_r b^{k-r}(b^{k-r} + \cdots + 1)^{-1}$. In $\tilde{X}$, the sum of the coordinates for paths at level j can be at most a_j, because the sum of the rates in any one bundle can be at most 1. So the sum of the first p_r coordinates in $\tilde{X}$ is at most $a_0 + a_1 + \cdots + a_{r-1}$. Therefore, the sum of the next ℓ_r bundles—each of which is from level r—must be at least

$$\frac{\ell_r b^{k-r}}{c(b^{k-r} + \cdots + 1)} - (a_0 + \cdots + a_{r-1}).$$

Hence one of these ℓ_r bundles must have the property that the sum of the rates assigned to paths in it in $\tilde{X}$ is at least

$$\frac{b^{k-r}}{c(b^{k-r} + \cdots + 1)} - \frac{a_0 + \cdots + a_{r-1}}{\ell_r}.$$

Now, the bundles in $\tilde{X}$ are arranged in ascending order of their total sums, and so we have the following fact: For at least $a_r - \ell_r$ bundles β in level r, the sum of the rates assigned to β in $\tilde{X}$ is at least

$$\frac{b^{k-r}}{c(b^{k-r} + \cdots + 1)} - \frac{a_0 + \cdots + a_{r-1}}{\ell_r}. \tag{1}$$

Call a bundle in level $r > 0$ *bad* if it does not satisfy the bound in (1); hence there are at most ℓ_r bad bundles in level r. The total number of edges covered by bad bundles of all levels is at most

$$\sum_{r=1}^{k} \frac{\ell_r a_k}{a_r} = \sum_{r=1}^{k} d^{2k-1} < d^{2k} = n,$$

and so there is at least one edge e of G that is not contained in any bad bundle.

Applying the capacity constraint to this edge e, we have

$$\begin{aligned}
1 &\geq \sum_{r=0}^{k} \left(\frac{b^{k-r}}{c(b^{k-r} + \cdots + 1)} - \frac{a_0 + \cdots + a_{r-1}}{\ell_r} \right) \\
&\geq \sum_{r=0}^{k} \left(\frac{b-1}{cb} - \frac{r a_{r-1}}{\ell_r} \right) = \sum_{r=0}^{k} \left(\frac{b-1}{cb} - \frac{r}{d} \right) \\
&\geq \frac{(k+1)(b-1)}{cb} - \frac{(k+1)^2}{d} \\
&\geq \frac{(k+1)k}{\frac{k}{3}(k+1)} - \frac{(k+1)^2}{(k+1)^2} = 3 - 1 = 2.
\end{aligned}$$

This contradiction completes the proof. □

We also consider global approximations for the problem of single-source fractional flow to n sinks, for which Megiddo showed that there is a maximum flow that is also lexicographically optimal. In fact, by adapting some of the analysis from his proof of this fact, one can show that $s^*(\mathcal{I}) = 1$ for all instances $\mathcal{I}$ of this problem. (We can also show that $c^*(\mathcal{I}) = O(\log n)$ for all instances of this problem, using an adaptation of the proof of Theorem 3.2.)

4. Scheduling. We first consider the problem of precedence-constrained scheduling, with release dates, on m identical parallel machines ($P|prec, r_j|all$). Thus, we are given a set of jobs $J = \{j_1, \ldots, j_n\}$ and a partial order $<_J$ on the jobs indicating precedence constraints. Job j_i has processing time p_i and release date r_i. We say that j_i "precedes" j_ℓ if $j_i <_J j_\ell$, and we say j_i is *minimal* if no job precedes it. For any subset $J' \subseteq J$, we can restrict the partial order to just the jobs in J'; we denote this by $<_{J'}$. We say that a set of jobs $K \subseteq J'$ is *independent* under $<_{J'}$ if there is no element in $J' \setminus K$ that precedes an element in K in the ordering $<_{J'}$.

As mentioned in the introduction, Hall et al. [14] considered many types of precedence-constrained scheduling problems. They produced a coordinate-wise $O(1)$-approximation to an optimal vector of *fractional completion times*, corresponding to a linear programming relaxation for these problems. However, we show that the linear programming formulation of Hall et al. [14] does not yield a global $O(1)$-approximation.

THEOREM 4.1. *For every number α, there exists an instance $\mathcal{I}$ of precedence-constrained 1-processor scheduling in which the optimal vector of fractional completion times X in the linear programming formulation of Hall et al. does not satisfy $c(X) \leq \alpha$.*

Proof. We choose a number β so that $\beta > \alpha^2$. We define an instance $\mathcal{I}$ with a set of $n = \beta + 2$ jobs $\{j_1, j_2, \ldots, j_n\}$. Job j_1 has processing requirement $p_1 = 1$, job j_2 has processing requirement $p_2 = \alpha$, and jobs $j_3, \ldots j_n$ each have processing requirement 0. The precedence constraints are simply the following: j_2 precedes j_i for every $i \geq 3$.

For any schedule S, let $X(S)$ denote its vector of completion times. Consider the schedule S that chooses the jobs in the order of their indices. We have $X(S) = (1, \alpha + 1, \alpha + 1, \ldots, \alpha + 1)$.

Now consider the linear programming formulation given by [14] for $\mathcal{I}$. It seeks a vector of completion times $(C_1, \ldots, C_n)$ so that $C_k \geq C_i + p_i$ when p_i precedes p_k, and

$$\sum_{j \in S} p_j C_j \geq \frac{1}{2}\left(p^2(S) + p(S)^2\right) \tag{2}$$

for every subset S of jobs. The objective is to minimize $\sum_i C_i$.

In our instance $\mathcal{I}$, the integer optimum for the objective value is $(\beta+1)\alpha+(\alpha+1)$; this is obtained by scheduling in the order $j_2, j_3, \ldots, j_n, j_1$. Thus, the optimum value of the linear program is at most this quantity.

Let $\tilde{C}$ denote an optimal solution to the linear program, represented as a vector of completion times. Note that the inequalities concerned with precedence constraints enforce $\tilde{C}_1 \geq 1$ and $\tilde{C}_i \geq \alpha$ for $i > 1$.

First we claim that $\tilde{C}_2 \leq \alpha + \frac{\alpha}{\beta}$. To see why this is so, suppose $\tilde{C}_2 > \alpha + \frac{\alpha}{\beta}$. We would then have $\tilde{C}_i > \alpha + \frac{\alpha}{\beta}$ for all $i > 2$, so the optimum value of the linear program would be greater than $(\beta+1)\alpha+\alpha+\frac{\alpha}{\beta}+1$, which is larger than the integer optimum.

(1) Initialize $J_0 \leftarrow J$ (J_0 denotes the jobs that haven't been processed yet and t denotes the current time).
(2) For $T = 1, 2, 4, \ldots$ do
  (i) Find a subset $J' \subseteq J_0$ such that all jobs in J' can be completely processed in the interval $[T, 2T)$, and $|J'|$ is maximum.
  (ii) Starting from time T schedule the jobs in J' such that they finish by time $2T$.
  (iii) $J_0 \leftarrow J_0 - J'$.

FIG. 2. *Algorithm that shows $c^* \leq 4$ for $P|prec, r_j$.*

Next we claim that $\tilde{C}_1 \geq \alpha$. We prove this by considering inequality (2) applied to the set $S = \{j_1, j_2\}$. We have $p(S) = 1 + \alpha$ and $p^2(S) = 1 + \alpha^2$. By our claim in the previous paragraph, we know that $\tilde{C}_2 \leq \alpha + \frac{\alpha}{\beta}$. Now suppose $\tilde{C}_1 < \alpha$. Then the left-hand side of (2) is at most $\alpha + \alpha^2 + \frac{\alpha^2}{\beta}$, but we have

$$\begin{aligned}\alpha + \alpha^2 + \frac{\alpha^2}{\beta} &< \alpha + \alpha^2 + 1 \\ &= \frac{1}{2}(1+\alpha)^2 + \frac{1}{2}(1+\alpha^2),\end{aligned}$$

which is a contradiction.

So in the solution $\tilde{C}$, the fractional completion time of each job is at least α. In the schedule S, on the other hand, there is a job with completion time 1. Thus $c(\tilde{C}) \geq \alpha$. □

THEOREM 4.2. *$c^*(\mathcal{I}) \leq 4$ for all instances $\mathcal{I}$ of precedence-constrained scheduling with release dates on m identical parallel machines for any $m \geq 1$.*

Proof. The scheduling algorithm is given in Figure 2. The For loop runs till J_0 becomes empty. The algorithm divides the time into exponentially growing intervals and tries to finish as many jobs as possible in each such interval. Let S denote the schedule it produces, and let X denote the vector of completion times of jobs under S. We write $\overrightarrow{X} = (C_1, C_2, \ldots, C_n)$. Let A_T denote the set J' chosen in iteration T of step (2)(ii). Let D_T denote the set J_0 at the beginning of this iteration.

Now consider some other schedule S' for this instance, let X' denote the vector of completion times of jobs under S', and write $\overrightarrow{X'} = (C'_1, C'_2, \ldots, C'_n)$. We focus on a particular coordinate k and let B' denote the first k jobs to finish under the schedule S'.

We wish to show that $C_k \leq 4C'_k$. Suppose $2^i \leq C'_k < 2^{i+1}$. Consider the iteration $T = 2^{i+1}$ of the For loop in Figure 2. At time T, all jobs in B' have been released. Further, all jobs in $B' - D_T$ can be completely processed in $[T, 2T)$ (because B' can be processed completely in $[0, T)$ in S'). So at least $|B'|$ jobs must finish by time $2T$ in S. But then $C_k \leq 2T \leq 4C'_k$, which is what we wanted to prove. □

We note that there exist instances $\mathcal{I}$ for which $c^*(\mathcal{I}) \geq 2$; finding the tightest upper bound is an open question. The construction in the proof of Theorem 4.2 does not yield a polynomial-time algorithm, since computing the set J' in step (2)(i) is an NP-hard problem. In fact, we can show that efficiently producing a schedule S satisfying $c(X_S) = O(1)$ for this problem would yield a greatly improved approximation algorithm for the *densest subgraph problem* described in the introduction: Given

a graph G and a number k, find an induced k-node subgraph with the maximum possible number of edges.

THEOREM 4.3. *Suppose there exists a polynomial-time algorithm $\mathcal{A}$ that, for any instance of precedence-constrained 1-processor scheduling $(1|prec|all)$, produces a schedule S satisfying $c(X_S) \leq \alpha$. Then there exists a randomized polynomial-time $8\alpha^2$-approximation algorithm for the densest subgraph problem.*

Proof. Suppose we are given an instance of the densest subgraph problem, specified by a graph $G = (V, E)$, with $|V| = n$, and a number $k \leq n$. We assume that $k \geq 2$ and that G contains at least $k/2$ edges; otherwise the instance is simple. We also assume for simplicity that α is an integer.

We construct the following instance $\mathcal{I}$ of $1|prec|all$. The set of jobs J will be partitioned into sets $J_V = \{j_v : v \in V\}$ and $J_E = \{j_e : e \in E\}$. Each job $j_v \in J_V$ has processing time 1, and each job $j_e \in J_E$ has processing time 0. The precedence constraints are as follows: If $e = (u, v)$, then j_u and j_v must precede j_e.

Using the algorithm $\mathcal{A}$, we construct a schedule S for the instance $\mathcal{I}$. Let X denote the set of jobs that finish by time αk. Let V' and E' denote the vertices and edges corresponding to jobs in X; note that since the endpoints of any edge in E' must belong to V', (V', E') is a subgraph of G. If $|V'| > k$, then, using a simple randomized algorithm, we can produce a set of vertices $V'' \subseteq V'$ so that $|V''| = k$ and the number of edges in the subgraph induced by V'' is at least

$$|E'| \cdot \frac{k(k-1)}{\alpha k(\alpha k - 1)} \geq \frac{|E'|}{4\alpha^2}.$$

Indeed, V'' is formed by picking k random vertices independently (and without replacement) from V. The probability that an edge $e \in E'$ has both endpoints in V'' is $\frac{k(k-1)}{|V'|(|V'|-1)}$. Using the fact $|V'| \leq \alpha k$ and linearity of expaction implies the bound mentioned above.

If the number of edges induced by V'' is less than $k/2$, then we replace V'' by a set of vertices inducing at least $k/2$ edges. We return V'' as the approximate solution to the instance of densest subgraph.

Now, consider an optimal solution to the densest subgraph instance; it consists of an induced subgraph (V^*, E^*) where $|V^*| = k$. We write $q = |E^*|$. Now, consider a schedule S' which first orders the jobs corresponding to V^* arbitrarily, followed by the jobs corresponding to E^*. In S', all $k + q$ of these jobs finish by time k; hence in our schedule S, at least $k + q$ jobs must finish by time αk. Since $|V'| \leq \alpha k$, it follows that $|E'| \geq q + k - \alpha k$.

If $q \leq 2\alpha k$, then our solution is a 4α-approximation, since it induces at least $k/2$ edges. Otherwise, $|E'| \geq q/2$; since our solution induces at least $|E'|/(4\alpha^2)$, it is an $8\alpha^2$-approximation. ∎

Surprisingly enough, we show that it is indeed possible to produce a schedule S in polynomial time such that $s(X_S) = O(1)$ for any instance of $P|prec, r_j|all$. We begin with an observation of Goel and Meyerson [8], which we prove for sake of completeness.

LEMMA 4.4. *Let $\mathcal{Y}$ be the set of feasible solutions to a linear program $\mathcal{P} = \{A \cdot Y \leq b\}$, where $Y \in \mathbf{R}^n$. We can find a vector $Y^* \in \mathcal{Y}$ in polynomial time such that $s(Y^*) = s^*(\mathcal{Y})$.*

Proof. We use the definition of s^* for minimization problems. We construct a vector $Z = (Z_1, \ldots, Z_n)$ such that Z_i will be the smallest value of $\sigma(\overleftarrow{Y})_i$ for any i.

Let us now show how to find Z_i. We add a variable z_i to the linear program $\mathcal{P}$. Further, for every subset $S \subseteq \{1, \ldots, n\}$, $|S| = i$, we add a constraint $z_i \geq \sum_{j \in S} Y_j$.

It is easy to see that minimizing z_i subject to these constraints gives us Z_i.

Now, suppose $\alpha = s^*(\mathcal{Y})$. Then, the vector Y^* can be found by adding the following constraints to $\mathcal{P}$: for every i, $1 \le i \le n$, and $S \subseteq \{1, \ldots, n\}$, where $|S| = i$, $\sum_j Y_j \le \alpha Z_i$. Now, treating α as a variable and minimizing its value gives us the vector Y^*. □

Hall et al. [14] gave a linear programming relaxation for $P|prec, r_j|\sum_j C_j$ for minimizing the sum of completion times. This linear program $\mathcal{P}$ had the completion time C_j of the jobs as variables. Note that here C_j denotes the completion time of job j. They showed the following nice fact about this linear program.

THEOREM 4.5 (see [14]). *Let $(C_1, \ldots, C_n)$ be a fractional solution to $\mathcal{P}$. Then, there exists a schedule S such that the completion time C'_j of job j in S is at most $2C_j$.*

We are now ready to prove the following theorem.

THEOREM 4.6. *Given any instance $\mathcal{I}$ of precedence-constrained scheduling with release dates on m identical parallel machines $(P|prec, r_j|all)$, we can find in polynomial time a schedule S such that $s(X_S) \le 16$.*

Proof. Consider the set of all solutions $C = (C_1, \ldots, C_n)$ to the linear program $\mathcal{P}$ given by Hall et al. [14]. We claim that there exists a solution C' such that $s(C') \le 8$.

Indeed, consider a schedule S' such that $c(X_{S'}) \le 4$. Now, given any solution C to $\mathcal{P}$, Theorem 4.5 states that there is a schedule S such that $X_S \prec_c 2\overleftarrow{C}$. But then, $X_{S'} \prec_c 4X_S \prec_c 8\overleftarrow{C}$. Thus, C' is a sequence of completion times in the schedule S'.

It follows from Lemma 4.4 that we can find such a solution C' in polynomial time. Now, one more application of Theorem 4.5 gives a schedule S with the desired properties. □

We now consider the special case in which there are two identical machines and all jobs have unit size $(2|p_j = 1, prec|all)$.

THEOREM 4.7. *$c^*(\mathcal{I}) = 1$ for every instance $\mathcal{I}$ of precedence-constrained two-processor scheduling with unit jobs.*

Proof. For a schedule S, define $X(S)$ to be its vector of completion times. We will consider only schedules that place at least one job in each time slot until all jobs are completed. Define $Z(S)$ to be the vector whose ith coordinate is the time at which the ith gap occurs. In other words, $Z(S)$ is the sorted (in increasing order) sequence of time slots in which only one job is scheduled.

Here is a basic property of the vectors of gaps.

LEMMA 4.8. *Let S and S' be two schedules. $\overrightarrow{X(S)} \not\prec_c \overrightarrow{X(S')}$ if and only if at least one of the following two conditions holds:* (i) *$Z(S')$ has fewer coordinates than $Z(S)$; or* (ii) *there is an index i so that $Z(S')_i > Z(S)_i$.*

We define the following two orderings on vectors.

DEFINITION 4.9. *Suppose $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$ are two vectors. We say that A is lexicographically greater than B if either of the following holds:*

- *$n < m$ and $a_1 = b_1, \ldots, a_n = b_n$.*
- *There is an index i such that $a_1 = b_1, \ldots, a_{i-1} = b_{i-1}$ and $a_i > b_i$.*

DEFINITION 4.10. *Let A and B be vectors defined as above. We say that A is greater than B in the colex ordering if either of the following holds:*

- *$n < m$.*
- *$n = m$ and there is an index i such that $a_m = b_m, \ldots, a_{i+1} = b_{i+1}, a_i > b_i$.*

Let S be a solution to $\mathcal{I}$ with lexicographically maximum vector $Z(S)$ (i.e., the first gap occurs as late as possible as possible; subject to this, the second gap occurs as late as possible; and so on). We show that $\overrightarrow{X(S)} \prec_c \overrightarrow{X(S')}$ for every schedule S'.

Suppose this is not true, and let Ω denote the set of S' for which $\overrightarrow{X(S)} \not\prec_c \overrightarrow{X(S')}$. Thus, every schedule $S' \in \Omega$ satisfies one of conditions (i) or (ii) in Lemma 4.8.

Let S' be the schedule in Ω whose vector $Z(S')$ is maximum in the colex ordering. Note that S' schedules at least one job in each time slot before its completion. We write $Z(S) = (g_1, \ldots, g_m)$ and $Z(S') = (g'_1, \ldots, g_{m'})$. We know that $Z(S)$ is greater than $Z(S')$ in lexicographic order, but if the first condition in Definition 4.9 were to hold, this would contradict Lemma 4.8. Thus there is an index i so that $g_1 = g'_1, \ldots, g_{i-1} = g'_{i-1}$ and $g_i > g'_i$.

From our precedence relation, we define the following undirected graph G_f on the set of jobs: (j, j') is an edge if and only if neither j nor j' precedes the other. The following result is due to Fujii, Kasami, and Ninomiya [7].

LEMMA 4.11 (see [7]). *Suppose G_f has a perfect matching. Then all jobs can be scheduled with no gaps.*

For our analysis, we create $i-1$ fictitious jobs $k_1, \ldots, k_{i-1}$, which are involved in no precedence relations. We consider the schedules that result from augmenting S and S' with the jobs $k_1, \ldots, k_{i-1}$ placed in gap positions $g_1 = g'_1, \ldots, g_{i-1} = g'_{i-1}$, respectively.

Define A to be the set of jobs which start before time g_i in S. Define B to be the set of jobs which start at time g'_i or before in S'. Let x denote the job in B which is run at time g'_i. In both cases, we include the fictitious jobs; thus A and $B \setminus \{x\}$ are each scheduled with no gaps. Note that $|A| > |B|$.

Now, there exists a matching M in G_f covering the set A, and there exists a matching M' in G_f covering the set $B \setminus \{x\}$. Let G'_f denote the subgraph of G_f on nodes $A \cup B$ and edges $M \cup M'$. All vertices of G'_f have degree at most 2, and x has degree 0 or 1 depending on whether it belongs to A. Thus, the component of G'_f containing x is a path P; note that P may be a single-node path if $x \notin A$. Let P consist of nodes $x = v_0, \ldots, v_r$ in order, with $e_i = (v_i, v_{i+1})$.

We now consider two possible cases.

(1) Suppose that P has an even number of nodes so that v_r is not matched in M. Hence $v_r \in B \setminus A$. Consider the set of jobs in $A \setminus B$, and let y be a minimal element of this set. We claim that (v_r, y) is an edge of G_f. Indeed, y occurs before v_r in the schedule S, and v_r occurs before y in the schedule S'. We define the matching M'' to consist of $M' \setminus E(P)$, together with the edges $(v_0, v_1), (v_2, v_3), \ldots, (v_r, y)$. Note that M'' covers $B \cup \{y\}$. Also, there cannot be a pair of jobs j, j' so that $j \notin B \cup \{y\}$, $j' \in B \cup \{y\}$, and j precedes j'. For if there were, then $j' = y$—since B was an initial set of jobs in S'—and this contradicts the minimality of y among the set of jobs in $A \setminus B$.

(2) Suppose that P has an odd number of nodes, so that v_r is matched in M but not in M'; thus, $v_r \in A \setminus B$. Among all jobs in $A \setminus B$ that precede v_r, including v_r, choose a minimal job y. (Thus we may have $y = v_r$.) We claim that (v_{r-1}, y) is an edge of G_f. For if v_{r-1} preceded y, then it would precede v_r, which is not the case, and we know that y cannot precede v_{r-1} since v_{r-1} precedes y in S'. We define the matching M'' to consist of $M' \setminus E(P)$, together with the edges $(v_0, v_1), (v_2, v_3), \ldots, (v_{r-1}, y)$. M'' covers $B \cup \{y\}$, and as in (a) there cannot be a pair of jobs j, j' so that $j \notin B \cup \{y\}$, $j' \in B \cup \{y\}$, and j precedes j'.

Thus in both cases, we have a job $y \in A \setminus B$ so that no job outside $B \cup \{y\}$ precedes any job in $B \cup \{y\}$, and there is a matching in G_f that covers $B \cup \{y\}$. We construct a new schedule S'' as follows: Using Lemma 4.11, we schedule all the jobs in $B \cup \{y\}$ initially with no gaps. We then schedule the rest of the jobs as in S',

replacing fictitious jobs by gaps and introducing a gap where y was in S'. (If this results in a time slot having two gaps, we slide all later jobs forward.)

We write $Z(S'') = (g''_1, \ldots, g''_{m''})$. We can view S'' as having been constructed from S' as follows: We first rearrange the order of jobs in B, within the first g'_i time slots; we then move y forward; we then eliminate any resulting time slots with two gaps. The first of these steps affects only the first i gaps, the second step pushes the ith gap later, and the third step does not push any gap earlier (and may lead to fewer gaps). Thus we have the following lemma.

LEMMA 4.12. *$m'' \leq m'$, $g''_i > g'_i$, $g''_j \geq g'_j$ for $i+1 \leq j \leq m''$, and hence $Z(S'')$ is greater than $Z(S')$ in the colex ordering.*

Now, since $S' \in \Omega$, we know that $Z(S')$ is shorter than $Z(S)$, or there is a j so that $g'_j > g_j$. In the former case, we see that $Z(S'')$ is also shorter than $Z(S)$. In the latter case, we know that $j > i$, since $g_1 = g'_1, \ldots, g_{i-1} = g'_{i-1}$ and $g_i > g'_i$. Now, either $j > m''$, in which case $Z(S'')$ is again shorter than $Z(S)$, or else by Lemma 4.12 it follows that $g''_j \geq g'_j > g_j$. Thus $Z(S'')$ satisfies condition (i) or (ii) in Lemma 4.8, and thus we have the following lemma.

LEMMA 4.13. *$S'' \in \Omega$.*

But Lemmas 4.12 and 4.13 contradict our choice of S', which establishes that Ω must be empty. We have therefore proved the theorem. □

The situation changes for precedence-constrained scheduling with unit jobs ($P|p_j = 1, prec|all$) when we have $m \geq 3$ machines. First, let $m = 3$ and consider an instance $\mathcal{I}$ with eight jobs and the following precedence relations: $j_1 < j_2 < j_3$ and $j_a < j_b$ for $a \in \{4,5,6\}$ and $b \in \{7,8\}$. Then the only schedule with makespan 3 leaves a machine idle in the second time slot, while there is a schedule with makespan 4 that completes six jobs in the first two time slots. Thus $c^*(\mathcal{I}) = 4/3$ for this instance. Similar examples can be constructed for larger numbers of machines.

However, it is easy to show that the greedy algorithm presented in the following theorem and its proof, based on the ideas of Graham [12], provides a global 2-approximation for every instance.

THEOREM 4.14. *$c^*(\mathcal{I}) \leq 2$ for every instance $\mathcal{I}$ of precedence-constrained m-processor scheduling with unit jobs.*

Proof. We produce a greedy schedule. Define the *depth* of a job j as the length of a longest chain in $<_J$ which ends at j (this can be easily computed in linear time). Order the jobs in a list L such that if job j comes before j', then the depth of j is at most that of j'. List schedule the jobs in L in a greedy manner while respecting the precedence constraints. Assume without loss of generality that the jobs are scheduled in the order $j_1, \ldots, j_n$. The completion time of job j_r in this schedule is at most $depth(j_r) + \frac{r}{m}$. It is not difficult to show that in any other schedule, the rth smallest completion time is at least $depth(j_r)$ and $\frac{r}{m}$. This proves the theorem. □

Finally, recall Smith's algorithm for 1-processor scheduling, which simply sequences jobs in nondecreasing order of processing time [24]. The standard proof of optimality in fact shows that $c^*(\mathcal{I}) = 1$ for all instances of this problem. To better understand where this strong global approximation result comes from, we can consider the following problem. We say that a precedence relation P (represented as a partially ordered set) is *universally fair* if, for any choice of processing times for the underlying jobs, we obtain an instance for which $c^*(\mathcal{I}) = 1$. Thus, Smith's algorithm shows that empty precedence relations are universally fair. For trivial reasons, precedence relations that are equal to chains are universally fair, since there is only one feasible schedule. We now state a result that completely characterizes all universally fair precedence relations.

THEOREM 4.15. *P is universally fair if and only if it can be partitioned into antichains $A_1, A_2, \ldots, A_k$ so that for every i from 1 to $k-1$, every $x \in A_i$, and every $y \in A_{i+1}$, we have $x \leq y$.*

Proof. To see the "if" direction, note that every valid schedule must do all of A_1, then all of A_2, and so forth up through A_k. Applying Smith's algorithm to each A_i separately, we see that $c^* = 1$.

Conversely, suppose that P does not have this "layered antichain" property, and assume by way of contradiction that $c^* = 1$ for every set of lengths assigned to the jobs of P. Let $n = |P|$. We first decompose P into a sequence of antichains by defining B_1 to be all minimal elements, B_2 to be all minimal elements of $P - B_1$, and so forth up to an antichain B_k. Since P does not have the layered property above, we can find the smallest index i for which there is an $x \in B_i$ and a $y \in B_{i+1}$ for which $x \not\leq y$. Note that there is some $x' \in B_i$ for which $x' \leq y$; otherwise, our decomposition procedure would have placed $y \in B_i$. Also, $y \not\leq x$, since x was minimal in $P - (B_1 \cup \cdots \cup B_{i-1})$, and $y \in P - (B_1 \cup \cdots \cup B_{i-1})$. Now consider assigning the following lengths to the jobs in P: We give each job other than $\{x, x', y\}$ a weight of $6n$ and we give x a length of 2, x' a length of 3, and y a length of 1. Note that because i was the minimum index for which $\{x, x', y\}$ could be found, any schedule must complete the jobs in $B_1 \cup \cdots \cup B_{i-1}$ before choosing any job in $B_i \cup \cdots \cup B_k$. Let $m = |B_1 \cup \cdots \cup B_{i-1}|$. Then there is a schedule which chooses x as job $m+1$ in sequence, and it has an $(m+1)$st coordinate of $6nm + 2$. There is another schedule which chooses x' as job $m+1$ and y as job $m+2$; this schedule has an $(m+2)$nd coordinate equal to $6nm + 4$. But there is no schedule whose $(m+1)$st coordinate is at most $6nm + 2$ and whose $(m+2)$nd coordinate is at most $6nm + 4$, and this contradicts our assumption that $c^* = 1$ for this set of weights. ∎

5. Facility location.

One facility. Recall that in the facility location problems we consider, we have a metric space M, a set $P = \{p_1, \ldots, p_n\}$ of *demand points* in M, and a set F of n' *candidate facility points*. First, we consider the case of 1-facility instances. For each point $f \in F$, let X_f denote the distance vector that arises when the facility is placed at f.

THEOREM 5.1. *For every 1-facility instance, there is a point $f^* \in F$ such that $s(X_{f^*}) \leq 4$.*

Proof. For simplicity, we assume n is even. We define the radius of $P' \subseteq P$ to be $\text{rad}(P') = \min_{f \in F} \max_{p' \in P} d(f, p')$. We choose a set $P' \subseteq P$ of $n/2$ points of minimum radius r, and define $f^* \in F$ to be a point for which $\max_{p' \in P} d(f^*, p') = \text{rad}(P')$.

Let $\overleftarrow{X_{f^*}} = (d_1, d_2, \ldots, d_n)$. Now, consider any facility $f' \in F$; let $\overleftarrow{X_{f'}} = (d'_1, d'_2, \ldots, d'_n)$. We must show that for each i, we have $d_1 + \cdots + d_i \leq 4(d'_1 + \cdots + d'_i)$. We observe that $d'_{n/2} \geq r$ since P' has the minimum radius r of any set of $n/2$ points, and f^* achieves this radius. Also, consider the set Q' consisting of the $i-1$ points at maximum distance from f' and the set Q^* consisting of the i points at maximum distance from f^*. There is at least one point $q \in Q^* \setminus Q'$, and we have

$$d_i \leq d(f^*, q) \leq d(f^*, f') + d(f', q) \leq d(f^*, f') + d'_i,$$

where the first inequality follows from the fact that $q \in Q^*$, and the third inequality follows since $q \notin Q'$.

If $p' \in P'$, then

$$r + d(p', f') \geq d(f^*, p') + d(p', f') \geq d(f^*, f'),$$

and so $d(p, f') \geq d(f^*, f') - r$. Since there are at least $n/2$ such points, it follows that for any index $i \leq n/2$, we have $d'_i \geq d(f^*, f') - r$. Hence $d(f^*, f') \leq d'_i + r \leq 2d'_i$, and so $d_i \leq d'_i + d(f', f^*) \leq 3d'_i$. For an index $i > n/2$, we have

$$\begin{aligned} d_1 + \cdots + d_i &\leq d_1 + \cdots + d_{n/2} + (i - n/2)r \\ &\leq 3(d'_1 + \cdots + d'_{n/2}) + (d'_1 + \cdots + d'_{n/2}) \\ &\leq 4(d'_1 + \cdots + d'_i). \qquad \square \end{aligned}$$

For the measure c^*, we have the following complementary pair of results for 1-facility instances. For the first, we construct a set of at most $1 + \log n$ facilities by iteratively identifying sets of size $1, 2, 4, 8, \ldots$ of minimum radius and including one facility for each of these sets that achieves this radius.

THEOREM 5.2. *Let $\mathcal{I}$ be an arbitrary 1-facility instance. There exists a set $F^* \subseteq F$ of size at most $1 + \log n$ so that $c(X_{F^*}) \leq 3$.*

Proof. We construct the set F^* as follows. Initialize the set S_0 to the set of all demand points P. We inductively define sets S_i, $i > 0$, and sets A_i, $i \geq 0$, as follows: A_i is a set of 2^i points from S_i of minimum radius. Let f_i be the facility which achieves the radius of A_i. Define S_i as $S_{i-1} \setminus A_{i-1}$.

We stop this process when some set $S_i = \phi$. This defines $m = O(\log n)$ sets A_i total. Define $F^* = \{f_0, f_1, \ldots, f_m\}$. Note that the last set A_m may have fewer than 2^m points. Also, note that in the distance vector X_{F^*}, points in A_i are not necessarily assigned to f_i but to the closest facility in F^*; this observation is important in the proof.

Now, consider any facility point $f \in F$, and let us label the points in P so that $d(f, p_1) \leq d(f, p_2) \leq \cdots \leq d(f, p_n)$. We fix an index t and consider the distance $d = d(f, p_t)$; we must show that at least t coordinates of X_{F^*} are each at most $3d$.

Let $P' = \{p_1, \ldots, p_t\}$; note that since $d(f, p_i) \leq d$ for all $p_i \in P'$, the radius of P' is at most d. We define s so that $2^s \leq t < 2^{s+1}$ and consider the following two cases.

First, suppose $P' \cap A_i = \phi$ for each i, $0 \leq i \leq s$. Now, A_i has minimum radius of all subsets of size 2^i in S_i; since we have $P' \subseteq S_i$ and $2^i \leq t$, it follows that A_i has radius at most d. Thus, all the coordinates in X_{F^*} corresponding to points in A_i have value at most d. Hence at least $|A_0| + \cdots + |A_s| = 2^{s+1} - 1 \geq t$ coordinates of X_{F^*} have value at most d.

Now, suppose instead that there is an index $i \leq s$ for which $P' \cap A_i \neq \phi$; in this case, choose the smallest such i. Let p' be a point in $P' \cap A_i$. Since P' does not intersect $A_0 \cup \cdots \cup A_{i-1}$, it is a subset of S_i; thus the radius of A_i is at most the radius of P' and hence at most d. Now, consider the vector X_{f_i}; since $f_i \in F^*$, we have $X_{F^*} \prec_c X_{f_i}$, and so it is enough to show that X_{f_i} contains t coordinates of value at most $3d$. But for any $p'' \in P'$, we have

$$d(p'', f_i) \leq d(p'', f) + d(f, p') + d(p', f_i) \leq 3d. \qquad \square$$

We have an essentially matching lower bound, which has a structure paralleling the construction that proves the previous theorem; essentially we build an instance consisting of widely separated clusters of size $1, 2, 4, 8, \ldots$; by defining distances appropriately in each, we can argue that each cluster must contain a facility.

THEOREM 5.3. *For any number α, there are 1-facility instances $\mathcal{I}$ for which no set F' of fewer than $\log n$ facilities achieves $c(X_{F'}) \leq \alpha$.*

Proof. The point set is divided into m "clusters." Cluster C_i (for $0 \leq i \leq m$) has 2^i demand points and one facility point. The distance between any two points in C_i

(including the facility point) is a number d_i, and we choose these distances so that $d_i < \alpha d_{i+1}$. The distance between any two points in different clusters is defined to be a number $d > \alpha d_1$. It is not difficult to show that this defines a metric. Note that there are $n = 2^{m+1} - 1$ demand points.

Suppose F' is a set of facility points such that $|F'| < m+1$. Then there is a cluster C_i such that $F' \cap C_i = \phi$. Let f be the facility point in C_i, and define $j = n - 2^i + 1$. The jth coordinate of $\overleftarrow{X_f}$ is d_i. On the other hand, all the points in $C_i \cup \cdots \cup C_m$ have coordinates in $X_{F'}$ that are at least $d_{i+1} > \alpha d_i$. As the number of such points is $n - 2^i + 1 = j$, the result follows. □

Multiple facilities. Now we consider a k-facility instance $\mathcal{I}$, where $k > 1$ is arbitrary.

THEOREM 5.4. *Fix numbers k and ε such that $\varepsilon \leq 1$. There exists a set $F^* \subseteq F$ of size $O(k \log n + \varepsilon^{-1})$ so that $\overleftarrow{X_{F^*}} \prec_c (3+\varepsilon)\overleftarrow{X_{F'}}$ for every set F' of at most k facilities.*

Proof. We first describe the construction of the set F^* that achieves this bound. For a point p and a set of points Q, we define $d(p, Q) = \min_{q \in Q} d(p, q)$. We define the *$k$-radius* of a set of demand points P' as follows:

$$\mathrm{rad}_k(P') = \min_{|F'|=k} \max_{p \in P'} d(p, F').$$

We choose a sufficiently large constant $\alpha = \Theta(\varepsilon^{-1})$, and perform the following construction. Define $x_0 = n$. Let $P_1 = P$ be the initial set of demand points, let $n_1 = |P_1|$, let $x_1 = \lceil n_1/2 \rceil$, and let C_1 be a subset of x_1 points of P_1 of minimum k-radius. In general, having defined P_i, n_i, x_i, C_i, we set $P_{i+1} = P_i \setminus C_i$, $n_{i+1} = |P_{i+1}|$, and $x_{i+1} = \lceil n_{i+1}/2 \rceil$, and let C_{i+1} be a subset of x_{i+1} points of P_{i+1} of minimum k-radius. Let r_{i+1} denote the k-radius of C_{i+1}.

We stop this process at the first s for which $x_s \leq \alpha$. Let $R \subseteq P$ denote the set of demand points not contained in $C_1 \cup \cdots \cup C_s$; note that $|R| \leq \alpha$. We define our set F^* of facilities as follows. For each set C_i, we include a set of k facilities that achieves the k-radius of C_i. For each point $p \in R$, we include a facility that is closest to p.

Note that $s = O(\log n)$, and hence $|F^*| = O(k \log n + \alpha)$. Now, consider any set F' of k facilities. We write $\overleftarrow{X_{F^*}} = (d_1, d_2, \ldots, d_n)$ and $\overleftarrow{X_{F'}} = (d'_1, d'_2, \ldots, d'_n)$.

The following lemma now directly implies Theorem 5.4.

LEMMA 5.5. *For each j, we have $d_1 + \cdots + d_j \leq (3+\varepsilon)(d'_1 + \cdots + d'_j)$.*

Proof. We claim first that

(1) for each i, $1 \leq i \leq s$, we have $d'_{x_i} \geq r_i$.

Now suppose this is not true for some i. Then there exists a set Q of at least $n - x_i + 1$ demand points of k-radius less than r_i. Since

$$|Q \setminus (C_1 \cup \cdots \cup C_{i-1})| \geq (n - x_i + 1) - (x_1 + \cdots + x_{i-1}) = n_i - x_i + 1 \geq x_i,$$

P_i contains a set Q' of at least x_i points from Q. But the k-radius of Q' is less than r_i, which contradicts our choice of C_i. This proves (1).

Now, consider an index i. For any $j < i$, we have $x_j \geq x_i$, and thus $d'_{x_i} \geq d'_{x_j} \geq r_j$. Thus we have

(2) $d'_{x_i} \geq \max(r_1, \ldots, r_i)$.

Using this, we show that

(3) for each $j \leq x_s$, we have $d_j \leq d'_j$.

We consider two cases in proving (3). First, if $d_j \leq \max(r_1, \ldots, r_s)$, then $d_j \leq d'_j$, since (2) implies that $d'_j \geq d'_{x_s} \geq \max(r_1, \ldots, r_s)$. On the other hand,

if $d_j > \max(r_1, \ldots, r_s)$, then each of $d_1, \ldots, d_j > \max(r_1, \ldots, r_s)$, and hence the points achieving these distances must belong to R. For each of these points, its coordinate in $X_{F'}$ is at least as large as its coordinate in X_{F^*}. It follows that the vector $\overleftarrow{X_{F'}}$ contains at least j coordinates of value at least d_j, and hence $d_j \leq d'_j$.

To handle boundary cases, we define $x_0 = n$, $x_{s+1} = 0$, $C_{s+1} = \phi$, and $n_{s+1} = n_s - x_s$. We now claim the following:

(4) Suppose j is an integer satisfying $x_i < j \leq x_{i-1}$, where $1 \leq i \leq s$. Then $d_j \leq d'_{x_i}$.

To prove this, we observe that

$$|C_{i+1}| + |C_{i+2}| + \cdots + |C_s| + |R| = x_{i+1} + \cdots + x_s + \alpha = n_{i+1} = n_i - x_i \leq x_i.$$

Now, $j > x_i$, so among the points with the j largest coordinates in X_{F^*}, there is at least one point that does not belong to $C_{i+1} \cup \cdots \cup C_s \cup R$. Thus, $d_j \leq \max(r_1, \ldots, r_i)$. But by (2), $d'_{x_i} \geq \max(r_1, \ldots, r_i)$, and so $d_j \leq d'_{x_i}$, as desired.

We now complete the proof of the lemma. First, if α is a sufficiently large multiple of ε^{-1}, then

$$x_{i-1} - x_i \leq (2 + \varepsilon)(x_i - x_{i+1})$$

for each i, $1 \leq i \leq s$. Now, for each such i, we have

$$\sum_{x_i < \ell \leq x_{i-1}} d_\ell \leq (x_{i-1} - x_i) d'_{x_i} \leq (2 + \varepsilon)(x_i - x_{i+1}) d'_{x_i} \leq (2 + \varepsilon) \sum_{x_{i+1} < \ell \leq x_i} d'_\ell.$$

Now, let j be any index between 1 and n. For $j \leq x_s$, we have $d_1 + \cdots + d_j \leq d'_1 + \cdots + d'_j$ by (3). Otherwise, there is an i so that $x_i < j \leq x_{i-1}$.

$$\begin{aligned} d_1 + \cdots + d_j &= \sum_{1 \leq \ell \leq x_s} d_\ell + \sum_{x_s < \ell \leq x_{s-1}} d_\ell + \cdots + \sum_{x_{i+1} < \ell \leq x_i} d_\ell + \sum_{x_i < \ell \leq j} d_\ell \\ &\leq \sum_{1 \leq \ell \leq x_s} d'_\ell + (2 + \varepsilon) \left(\sum_{1 \leq \ell \leq x_s} d'_\ell + \cdots + \sum_{x_{i+1} < \ell \leq x_i} d'_\ell \right) \\ &\leq (3 + \varepsilon)(d'_1 + \cdots + d'_{x_i}) \\ &\leq (3 + \varepsilon)(d'_1 + \cdots + d'_j). \end{aligned}$$

This proves the theorem. □

Note that the construction described in the proof does not yield a polynomial-time algorithm since it involves computing the k-radius of subsets of points. However, a recent result of Charikar et al. [4] provides a polynomial-time 3-approximation algorithm for the following problem: Given a set P and a parameter x, find a subset C of size x of minimum k-radius. We can use this algorithm to construct the sets C_i in the proof; as a result we lose a factor of 3 in each of the claims (1)–(4) above, and hence obtain a set F^* of size $O(k \log n + \varepsilon^{-1})$ for which $s(F^*) \leq 9 + \varepsilon$.

As mentioned in section 1, we have a lower bound that nearly matches the result of Theorem 5.4.

THEOREM 5.6. *For any number α, there are k-facility instances $\mathcal{I}$ for which no set F' of fewer than $\Omega(k \frac{\log(n/k)}{\log \alpha})$ facilities achieves $s(X_{F'}) \leq \alpha$.*

Proof. For simplicity we will consider values of k that are even, and write $k = 2k'$. We choose a constant $a > 12\alpha$. For a number m, we define $n_1, n_2, \ldots, n_m$ via $n_i = a^{4(m-i)}$ and $n_0 = n_1$.

Our instance $\mathcal{I}$ consists of a set of $n = n_0 + n_1 + \cdots + n_m$ points P, partitioned into *clusters* $C_0, \ldots, C_m$, where C_i has n_i points. We call a cluster *large* if it has at least k points; we divide each large cluster C_i into k' *subclusters* $C_{i,1}, \ldots, C_{i,k'}$ such that the difference in the size of any two subclusters of C_i is at most one. Note that each such subcluster has size at least $\frac{n_i}{k'} - 1$. For the clusters that are not large, we will say that they consist of a single subcluster. We define $F = P$, so that the set of candidate facility points is the same as the set of demand points.

We define the interpoint distances as follows. The distance between any two points in the same subcluster is 0 (or a very small number). The distance from a point in $C_{i,s}$ to a point in $C_{i,s'}$, with $s \neq s'$, is $y_i = a^{2(i-1)}$ for $i > 0$ and $y_0 = 1$ for $i = 0$. The distance from a point in C_i to a point in any cluster C_j for $j < i$ is equal to a number $x_i = a^{2i-1}$. For simplicity of notation, we will define $x_0 = 0$.

Note that $n \leq 3a^{4(m-1)} \leq a^{4m}$, and hence $m \geq \frac{\log n}{4 \log a}$. Since C_j is large whenever $a^{4(m-j)} \geq k$, and hence whenever $m - j \geq \frac{\log k}{4 \log a}$, there are at least

$$m' = m - \frac{\log k}{4 \log a} \geq \frac{\log n}{4 \log a} - \frac{\log k}{4 \log a} = \frac{\log(n/k)}{\log \alpha}$$

large sets.

Let F' be a set of facility points such that $|F'| \leq m' \lfloor \frac{k'}{2} \rfloor$. Then there is some large C_i such that $|F' \cap C_i| \leq \lfloor \frac{k'}{2} \rfloor$. Since there are k' subclusters in C_i and at most $\lfloor \frac{k'}{2} \rfloor$ contain a facility, the distance of at least

$$\frac{k'}{2} \left(\frac{n_i}{k'} - 1 \right) = \frac{n_i}{2} - \frac{k'}{2}$$

points in C_i from the nearest facility in F' is at least y_i. So, if ℓ denotes $\frac{n_i}{2} - \frac{k'}{2}$, then the ℓth coordinate of $\sigma(\overleftarrow{X_{F'}})$ is at least

$$\sigma(\overleftarrow{X_{F'}})_\ell = y_i \left(\frac{n_i}{2} - \frac{k'}{2} \right).$$

Consider the following set F'' of k facility points: If $i > 0$, we place k facilities in clusters C_i and C_{i-1}, one in each subcluster. If $i = 0$, we place k facilities in C_0 and C_1, again one in each subcluster. Since the cases $i = 0$ and $i = 1$ are similar, we will assume in what follows that $i > 0$.

Now, observe that if C_j is large, then

$$n_{j+1} + \cdots + n_m \leq 2n_{j+1} \;\leq\; \frac{n_j}{4} \;\leq\; \frac{n_j - k'}{2} \;=\; \ell,$$

and thus the ℓth coordinate of $\sigma(\overleftarrow{X_{F''}})$ is equal to

$$\sigma(\overleftarrow{X_{F''}})_\ell = \sum_{j>i} n_j x_j + \left(\frac{n_i}{2} - \frac{k'}{2} - \sum_{j>i} n_j \right) x_{i-1}.$$

Now we can compare these two quantities as follows:

$$\begin{aligned}
\sigma(\overleftarrow{X_{F''}})_\ell &= \sum_{j>i} n_j x_j + \left(\frac{n_i}{2} - \frac{k'}{2} - \sum_{j>i} n_j\right) x_{i-1} \\
&\le \sum_{j>i} a^{4m-2j-1} + \frac{n_i x_{i-1}}{2} \\
&\le 3a^{4m-2i-3} \\
&< \frac{1}{12\alpha} \cdot 3a^{4m-2i-2} \\
&= \frac{1}{\alpha} \cdot \frac{a^{4m-2i-2}}{4} \\
&= \frac{1}{\alpha} \cdot \frac{y_i n_i}{4} \\
&\le \frac{1}{\alpha} \cdot y_i \left(\frac{n_i}{2} - \frac{k'}{2}\right) \\
&\le \frac{1}{\alpha} \cdot \sigma(\overleftarrow{X_{F'}})_\ell. \qquad \square
\end{aligned}$$

For the c^* measure, we show that in general the lower bound is essentially as bad as possible when $k > 1$.

THEOREM 5.7. *For any number α, there are 2-facility instances $\mathcal{I}$ for which no proper subset $F' \subseteq F$ achieves $c(X_{F'}) \le \alpha$.*

Proof. Consider the following instance $\mathcal{I}$. The n demand points $P = \{p_1, \ldots, p_n\}$ and $n' = n$ candidate facilities $F = \{f_1, \ldots, f_n\}$ are partitioned into n *clusters* $C_1, \ldots, C_n$ so that $C_i = \{p_i, f_i\}$. The distance between p_i and f_i is equal to a number y_i, and the distance from a point in C_i to a point in any cluster C_j for $j < i$ is equal to a number x_i. We choose these distances so that $\alpha x_i < y_{i+1}$ and $\alpha y_i < x_i$ for each i.

Now consider any set $F' \subseteq F$ of fewer than n facilities, and suppose it does not contain the facility in C_i. Define $F'' = \{f_{i-1}, f_i\}$. Then the ith coordinate of the distance vector $\overrightarrow{X_{F''}}$ is equal to $d(p_i, f_i) = y_i$. On the other hand, among the first i coordinates of the distance vector $\overrightarrow{X_{F'}}$, there is at least one corresponding to the distance to some point in $C_i \cup \cdots \cup C_n$. But all these distance are at least $x_i > \alpha y_i$, and thus the theorem follows. $\square$

REFERENCES

[1] D. BERTSEKAS AND R. GALLAGER, *Data Networks*, Prentice-Hall, Englewood Cliffs, NJ, 1987.

[2] R. BHARGAVA, A. GOEL, AND A. MEYERSON, *Using approximate majorization to characterize protocol fairness*, in ACM SIGMETRICS/Performance, 2001, pp. 330–331.

[3] A. BLUM, P. CHALASANI, D. COPPERSMITH, W. PULLEYBLANK, P. RAGHAVAN, AND M. SUDAN, *The minimum latency problem*, in Proceedings of the 26th Annual ACM Symposium on Theory of Computing, 1994, pp. 163–171.

[4] M. CHARIKAR, S. KHULLER, D. M. MOUNT, AND G. NARASIMHAN, *Algorithms for facility location problems with outliers*, in Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, Washington, DC, 2001, pp. 642–651.

[5] E. G. COFFMAN AND R. L. GRAHAM, *Optimal scheduling for two-processor systems*, Acta Inform., 1 (1972), pp. 200–213.

[6] H. EISELT AND G. LAPORTE, *Objectives in location problems*, in Facility Location: A Survey of Applications and Methods, Z. Drezner, ed., Springer, New York, 1995, pp. 151–180.

[7] M. FUJII, T. KASAMI, AND K. NINOMIYA, *Optimal sequencing of two equivalent processors*, SIAM J. Appl. Math., 17 (1969), pp. 784–789.

[8] A. Goel and A. Meyerson, *private communication.*
[9] A. Goel, A. Meyerson, and S. Plotkin, *Combining fairness with throughput: Online routing with multiple objectives*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, 2000, pp. 670–679.
[10] A. Goel, A. Meyerson, and S. Plotkin, *Approximate majorization and fair online load balancing*, in Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, Washington, DC, 2001, pp. 384–390.
[11] M. Goemans and J. Kleinberg, *An improved approximation ratio for the minimum latency problem*, in Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, Atlanta, 1996, pp. 152–158.
[12] R. L. Graham, *Bounds for certain multiprocessing anomalies*, Bell System Tech. J., 45 (1966), pp. 1563–1581.
[13] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinooy Kan, *Optimization and approximation in determinsitic sequencing and scheduling: A survey*, Ann. Discrete Math., 5 (1979), pp. 287–326.
[14] L. A. Hall, A. Schulz, D. B. Shmoys, and J. Wein, *Scheduling to minimize average completion time: Off-line and on-line algorithms*, Math. Oper. Res., 22 (1997), pp. 513–544.
[15] H. Hayden, *Voice Flow Control in Integrated Packet Networks*, Technical report LIDS-TH-1152, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, 1981.
[16] J. Jaffe, *Bottleneck flow control*, IEEE Trans. Comm., 29 (1981), pp. 954–962.
[17] J. Kleinberg, Y. Rabani, and E. Tardos, *Fairness in routing and load balancing*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, 1999, pp. 568–578.
[18] G. Kortsarz and D. Peleg, *On choosing a dense subgraph*, in Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science, 1993, pp. 692–701.
[19] A. Kumar and J. Kleinberg, *Fairness measures for resource allocation*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, 2000, pp. 75–85.
[20] M. Marsh and D. Schilling, *Equity measurement in facility location analysis: A framework and analysis*, European J. Oper. Res., 74 (1994), pp. 1–17.
[21] N. Megiddo, *Optimal flows in networks with multiple sources and sinks*, Math Programming, 7 (1974), pp. 97–107.
[22] V. Pareto, *Cours d'Economie Politique*, Vol. 1, F. Rouge & Cie, Lausanne, Switzerland, 1896.
[23] C. Phillips, C. Stein, and J. Wein, *Scheduling jobs that arrive over time*, in Proceedings of the Fourth Workshop on Algorithms and Data Structures, Lecture Notes in Comput. Sci. 955, Springer, Berlin, 1995, pp. 86–97.
[24] W. Smith, *Various optimizers for single-stage production*, Naval Res. Logist. Quart., 3 (1956), pp. 59–66.
[25] C. Stein and J. Wein, *On the existence of schedules that are near-optimal for both makespan and total weighted completion time*, Oper. Res. Lett., 21 (1997), pp. 115–122.

SIAM J. COMPUT.
Vol. 36, No. 3, pp. 681–694

DYNAMIC SUBGRAPH CONNECTIVITY WITH GEOMETRIC APPLICATIONS*

TIMOTHY M. CHAN†

Abstract. Inspired by dynamic connectivity applications in computational geometry, we consider a problem we call *dynamic subgraph connectivity*: design a data structure for an undirected graph $G = (V, E)$ and a subset of vertices $S \subseteq V$ to support insertions/deletions in S and connectivity queries (are two vertices connected?) in the subgraph induced by S. We develop the first sublinear, fully dynamic method for this problem for general sparse graphs, using a combination of several simple ideas. Our method requires $\widetilde{O}(|E|^{4\omega/(3\omega+3)}) = O(|E|^{0.94})$ amortized update time, and $\widetilde{O}(|E|^{1/3})$ query time, after $\widetilde{O}(|E|^{(5\omega+1)/(3\omega+3)})$ preprocessing time, where ω is the matrix multiplication exponent and $\widetilde{O}$ hides polylogarithmic factors.

Key words. data structures, dynamic graph algorithms, connectivity, computational geometry

AMS subject classifications. 68Q25, 68P05, 68U05

DOI. 10.1137/S009753970343912X

1. Introduction.

1.1. Geometric motivation. *Dynamic graph connectivity*—maintaining an undirected graph under edge insertions and deletions to answer queries of the form, "Are two vertices connected?"—is a basic problem in the area of graph data structures. In the same way, connectivity problems concerning a dynamic collection of geometric objects are fundamental in computational geometry. However, unlike dynamic graph connectivity, which has been extensively studied and has enjoyed much recent success with the discovery of near-logarithmic algorithms [17, 21, 32], progress in *dynamic geometric connectivity* has been scarce. For this reason, we decide to start our investigation with a simple version of the problem:

> Maintain a set of n axis-parallel rectangles in the plane, under insertions and deletions, to answer queries of the form, "Given two points a and b, is there a path from a to b that lies inside the union of these rectangles?"

Rectangular connectivity queries have numerous applications, for example, in VLSI design, communication networks, and geographic information systems. (See Figure 1.1.) Solving this problem might pave the way for the study of dynamic connectivity of other objects and, perhaps, tougher questions such as dynamic shortest paths and motion planning.

No fully dynamic solution for rectangular connectivity has been reported even for the special case of orthogonal line segments. Existing geometric data structuring techniques (notably, range searching) [2, 29] seem insufficient, because connectivity queries are more global in nature. On the other hand, although our problem is equivalent to dynamic connectivity in the intersection graph (where we place an edge between

*Received by the editors December 29, 2003; accepted for publication (in revised form) April 17, 2006; published electronically October 10, 2006. A preliminary version of this paper appeared in *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, 2002, Montreal, QC, Canada, pp. 7–13. This work was supported in part by an NSERC research grant.
http://www.siam.org/journals/sicomp/36-3/43912.html
†School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada (tmchan@uwaterloo.ca).

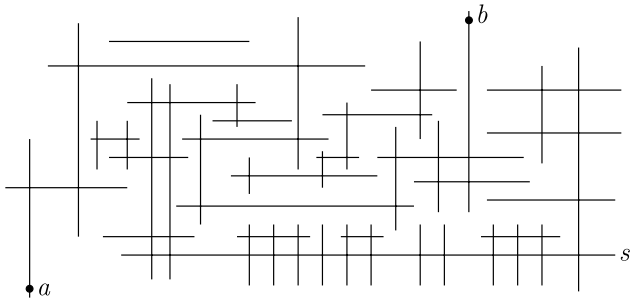


FIG. 1.1. *A collection of "roads." Is b reachable from a?*

every pair of intersecting rectangles), a straightforward application of dynamic graph connectivity results would not lead to an efficient solution either, because the intersection graph can have quadratic size, and an insertion/deletion of a rectangle can cause a linear number of edge updates in the worst case. (Imagine deleting and reinserting a segment such as s into Figure 1.1.) Thus, work on dynamic graph connectivity [13, 15, 18, 17, 21, 31, 32] is only the beginning if we want to tackle the more challenging dynamic connectivity problems from geometry.

In this paper, we show that a sublinear time bound is indeed theoretically possible for dynamic connectivity of rectangles and, in fact, axis-parallel boxes in any fixed dimension d. Specifically, we achieve $\widetilde{O}(n^{4\omega/(3\omega+3)})$ amortized time for insertions and deletions, and $\widetilde{O}(n^{1/3})$ time for queries, using $\widetilde{O}(n)$ space. Here and throughout the paper, the $\widetilde{O}$ notation hides polylogarithmic factors in n, and ω denotes the matrix multiplication exponent. The current best matrix multiplication result with $\omega < 2.376$ by Coppersmith and Winograd [7] implies an $O(n^{0.94})$ upper bound for updates, but any subcubic method with $\omega < 3$ (such as Strassen's) would already imply a sublinear upper bound.

This result is striking in two respects: (i) the independence of the exponent of our time bounds on the dimension d and (ii) the usage of fast matrix multiplication, which is rare among algorithms in computational geometry. (Applications of fast matrix multiplication are more common in dynamic graph algorithms (see, e.g., [9, 24]), but the sublinearity of our update bound is still unusual.) In section 7, we partially explain why a polylogarithmic solution is unlikely given the current state of the art and why points (i) and (ii) might be inherent to the problem itself, at least for $d \geq 3$.

1.2. Previous geometric work. Statically, connectivity for n rectangles in the plane can be decided in $O(n \log n)$ time [23]. Agarwal and van Kreveld [3] gave an approach for the *incremental* (insertion-only) case that, in particular, yielded an $O(\log^2 n)$ amortized time bound for orthogonal segments.

For unit squares and unit hypercubes (and thus for near-equal-size boxes of bounded aspect ratios), a fully dynamic, polylogarithmic method can be obtained by an easy reduction to the maintenance of the L_∞-minimum spanning tree of a point set, which was solved by Eppstein [12] in any fixed dimension (using known dynamic minimum spanning tree results for graphs [21]). For arbitrary rectangles and arbitrary hypercubes, Hershberger and Suri [19, 20] considered the *kinetic* problem of maintaining connectivity as objects move continuously according to given flight plans: allowing a quadratic number of events, we can easily reduce the kinetic problem to dynamic graph connectivity using the intersection graph; Hershberger and Suri showed that

the space complexity can be decreased from quadratic to linear while still supporting efficient updating.

1.3. The dynamic subgraph connectivity problem. As we have pointed out, the intersection graph cannot directly be used for dynamic rectangular connectivity. However, existing geometric range searching results allow us to compress the intersection graph, using so-called biclique covers [1, 14]. Although this technique is familiar, it has not been used in connectivity applications and, in our opinion, leads to a conceptually cleaner description of geometric connectivity algorithms because of the separation of geometric and nongeometric elements.

As it turns out (see section 2), the problem on the compressed intersection graph leads to an interesting generalization of dynamic graph connectivity, which is not as well studied but that we believe is equally fundamental:

> Maintain an undirected graph $G = (V, E)$ with n vertices and m edges, and a subset $S \subseteq V$ of vertices, under the following operations: insert an edge into E, delete an edge from E, insert a vertex into S, and delete a vertex from S. Queries to be answered are of the form, "Given two vertices $u, v \in S$, is there a path from u to v that uses only vertices from S?"

We call this problem *dynamic subgraph connectivity*. The difficulty lies not in edge updates to E but in vertex updates to the subset S. If each vertex is inserted and deleted only once, then we can directly apply data structures for dynamic graph connectivity to maintain the subgraph induced by S; the amortized cost would be near-logarithmic per edge, since each edge is inserted and deleted $O(1)$ times. In particular, we can therefore handle the *incremental* (insertion-only) or *decremental* (deletion-only) case. However, in general, vertices may be deleted and reinserted into S as often as we like; this naive approach would have cost proportional to the degree of the vertex, which can be linear in n for every update in the worst case. We give a nontrivial sublinear solution for arbitrary update sequences in general sparse graphs with $\widetilde{O}(m^{4\omega/(3\omega+3)})$ amortized update time, $\widetilde{O}(m^{1/3})$ query time, $\widetilde{O}(m^{(5\omega+1)/(3\omega+3)})$ preprocessing time, and $\widetilde{O}(m)$ space.

To appreciate the result, note that existing polylogarithmic dynamic graph techniques [17, 21, 32] do not work, because the usual "certificate" [13, 24], a spanning forest, can change drastically in a single vertex update. Alternatively, it is possible to reduce the problem to reachability in a dynamic *directed* graph, so that a vertex update can be simulated by a single edge change (by creating two copies of each vertex joined by a directed edge). However, dynamic directed graph reachability [16, 30] is computationally more demanding (a main goal there was to obtain an $o(n^2)$ update bound).

1.4. Previous graph work. The dynamic subgraph connectivity problem has indeed been proposed before, in a paper by Frigioni and Italiano [16]. The motivation there was on the connectivity of communication networks, where processors can become faulty and can later go back up; viewed alternatively, vertices can be "switched" on and off. Frigioni and Italiano used the term *complete dynamic graph model* to describe the setting where both edge and vertex updates are supported. They described polylogarithmic connectivity results for the special case of planar graphs (relying on separators) but left the general case (which is necessary in our geometric application) open.

1.5. How to solve it. Our solution to dynamic subgraph connectivity involves several techniques. Each in itself is not difficult, and when put together in the right way, they yield the winning combination. Our recipe includes the following:

- Processing the update sequence in blocks. Khanna, Motwani, and Wilson [24] have used this trick to design various dynamic graph algorithms for *offline* updates (when the update sequence is given in advance), while the author [5, 6] has applied essentially the same idea to several dynamic geometric problems under *semi-online* updates (when only the deletion times are given in advance) as well as general updates.
- The "high-low" trick of Alon, Yuster, and Zwick [4]. This was originally developed for the problem of finding triangles in sparse graphs. We observe how the idea can yield faster multiplication algorithms for sparse rectangular matrices (see section 3).
- Precomputing information for high-degree vertices, so that their repeated deletions and reinsertions would not be as costly.
- Finally, amortizing deletion cost via a standard "weighted split" idea, which is analogous to the standard "weighted union" heuristic (e.g., see [8, Chapter 21]).

To illustrate the effects of these ideas progressively, we present the algorithm in stages (in the actual order of discovery), starting with an $O(m^{0.82})$ offline update method (section 4), extending it to an $O(m^{0.91})$ semi-online update method (section 5), and ending with the $O(m^{0.94})$ fully dynamic update method (section 6).

2. From geometry to graphs. We first describe how to reduce dynamic geometric connectivity problems to dynamic subgraph connectivity. This section is the only place where computational geometry techniques are used; afterwards, we can devote all our attention to the graph problem.

The reduction is a simple consequence of known compact representations of geometric intersection graphs. Specifically, any intersection graph of boxes can be represented as a union of bicliques (complete bipartite subgraphs $A_i \times B_i$ over some vertex subsets A_i and B_i) so that the size of the representation (the total number of vertices in the bicliques) is near-linear.

LEMMA 2.1. *Fix any constant d. Given a set of n (axis-parallel) boxes in $\mathbb{R}^d$, we can form subsets A_i and B_i of total size $\widetilde{O}(n)$ in $\widetilde{O}(n)$ time, such that two boxes a and b intersect iff (a, b) or (b, a) is in $A_i \times B_i$ for some index i. This property can be maintained dynamically: each insertion/deletion of a box causes an amortized $\widetilde{O}(1)$ number of insertions/deletions in the A_i's and B_i's.*

Proof. We first consider the static case. The following *orthogonal range (or intersection) searching* problem [2, 11, 26, 27, 28, 29] is well studied in computational geometry: preprocess a given set of n boxes in a data structure so that given a query box b, we can quickly report all boxes intersecting b. A standard solution to this problem is the *range tree* (or a multilevel *segment tree*). The precise details behind the data structure are not important here (see the above references for more information). The main properties to note are that the data structure consists of $O(n \log^{d-1} n)$ "nodes," each of which stores a subset of boxes which we refer to as a *canonical subset*, and that given a query box b, the set of boxes intersecting b can be returned as the union of the canonical subsets at $O(\log^d n)$ nodes of the data structure. The total size of the canonical subsets is $O(n \log^d n)$; the preprocessing time is $O(n \log^d n)$ and the query time is $O(\log^d n)$.

To form the subsets A_i and B_i, we build the data structure for the given input boxes and perform queries for all the boxes. For each node i, we let A_i be the canonical subset at i and B_i be all boxes whose query returns the canonical subset at i. The total size of the A_i's and B_i's is $O(n \log^d n)$, and correctness is obvious.

For the dynamic case, we need a version of the orthogonal range searching problem that supports insertion. A weight-balanced range tree, for instance, can be used [26, 28]. Again the precise details behind the data structure are not important. The main additional properties are that although each canonical subset does not change, new nodes and canonical subsets may be created, and the total size of all canonical subsets created by a sequence of n insertions is bounded by $O(n \log^{d+1} n)$. (With more care, a log factor can probably be saved.)

To update the subsets A_i and B_i under the insertion of a new box b, we perform a query for b and insert b into every set B_i such that the query returns the canonical subset at node i. We then insert b into the data structure itself, and for every new node i created, we insert all elements of the new canonical subset into A_i and initialize $B_i = \emptyset$. (We do not delete canonical subsets of old nodes destroyed.) The amortized number of insertions into the A_i's and B_i's is $O(\log^{d+1} n)$.

To update the subsets A_i and B_i under the deletion of a box b, we simply remove b from each subset that contains b. The cost of deletion can be "charged" to the insertion cost by amortization. (If n is to denote the number of current boxes instead of the number of insertions, we need another standard amortization trick: rebuilding the whole data structure whenever the value of n is halved [26, 28].) □

THEOREM 2.2. *Fix any constant d. The dynamic connectivity problem for boxes in $\mathbb{R}^d$ can be reduced to the dynamic subgraph connectivity problem on a graph with $m = \widetilde{O}(n)$ edges. Each box update causes an amortized $\widetilde{O}(1)$ number of graph updates.*

Proof. Given a set of boxes, we apply Lemma 2.1 and define the following graph G: for each box, we create a vertex; in addition, for each subset pair (A_i, B_i), we create a new vertex v_i and place edges from v_i to all members of $A_i \cup B_i$. (See Figure 2.1 for an idealized example.) Initially, all vertices are put in the subset S. Whenever A_i or B_i becomes empty, we delete the vertex v_i from S. When both A_i and B_i are nonempty, we reinsert v_i into S.

Each insertion/deletion of a box causes the subsets A_i and B_i to undergo $\widetilde{O}(1)$ amortized number of insertions/deletions, which in turn causes the above graph G and subset S to undergo $\widetilde{O}(1)$ amortized number of edge and vertex insertions/deletions.

To test whether two query boxes are connected in the intersection graph, we just test whether the two boxes are connected in the subgraph of G induced by S.

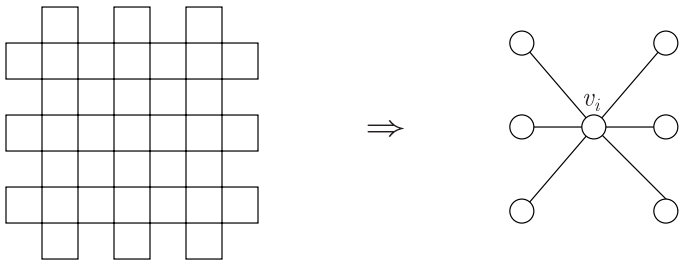


FIG. 2.1. *From dynamic box connectivity to dynamic subgraph connectivity.*

Correctness is evident (if A_i and B_i are nonempty, the members of $A_i \cup B_i$ are indeed all connected to each other in the intersection graph). To test whether two query points are connected, we first find a box containing each point (which takes polylogarithmic time by known data structures for orthogonal range searching) and then test whether these two boxes are connected. □

Remarks. The above reduction also works in the static, incremental (insertion-only), decremental (deletion-only), offline, and semi-online settings.

In the static case, graph and subgraph connectivity can be solved in linear time by depth-first search, so we automatically get an $O(n \operatorname{polylog} n)$ algorithm for the box connectivity problem. This result was known before, as noted in section 1.

In the incremental/decremental case, subgraph connectivity reduces to graph connectivity, as we have observed in section 1 (by explicitly maintaining the induced subgraph). Incremental graph connectivity is equivalent to the union-find problem [8], and decremental graph connectivity can be solved in near-logarithmic time [31, 32], so we automatically get $O(\operatorname{polylog} n)$ incremental/decremental box connectivity algorithms. (We do not state the precise polylogarithmic bounds, because slight improvements are likely possible by a more direct approach.) The incremental box connectivity result was already known [3], but the decremental box connectivity has not been addressed before.

In the offline, semi-online, and fully dynamic cases, the subgraph connectivity results in sections 4–6 will imply corresponding box connectivity results, up to polylogarithmic factors.

Other classes of objects can be considered. For example, we can obtain the same results for line segments that have a fixed number of slopes, by using variants of orthogonal range searching in the proof of Lemma 2.1.

For arbitrary line segments in the plane, we need to use known nonorthogonal range searching data structures [2] in the proof of Lemma 2.1, but these data structures require canonical subsets of total size $\widetilde{O}(n^{4/3})$ instead of $O(n \operatorname{polylog} n)$; as a result, the graph in Theorem 2.2 now has $m = \widetilde{O}(n^{4/3})$ edges, and each segment update causes an amortized $\widetilde{O}(n^{1/3})$ number of graph updates. Unfortunately, our dynamic subgraph connectivity results are too weak to yield meaningful bounds in this case. Still, we can obtain an $\widetilde{O}(n^{4/3})$ result for static connectivity and an $\widetilde{O}(n^{1/3})$ result for incremental and decremental connectivity for arbitrary line segments. This static result was known [25], so was the incremental result [3], but the decremental result appears new.

3. Fast sparse matrix multiplication. Our sublinear results for dynamic subgraph connectivity will require a time bound for matrix multiplication that is sensitive to the sparseness of the given matrices. Although obtaining such an input-sensitive bound is in general an outstanding problem for the standard square-matrix case (see [34] for an independent, recent development), in this section we note a simple input-sensitive bound for a rectangular-matrix case that is sufficient for our application.

Specifically, we consider the complexity $M(n, q \,|\, m)$ of multiplying a $q \times n$ 0-1 matrix A with an $n \times q$ 0-1 matrix B, where m is the number of nonzero entries per matrix, with $m = \Omega(n)$. This is asymptotically equivalent to the complexity of the following graph problem: given an m-edge bipartite graph with a set P of n vertices on one side and a set Q of q vertices on the other, count the number $C[u, v]$ of vertices adjacent to both u and v for every vertex pair $u, v \in Q$. (To reduce the graph problem to matrix multiplication, let $a_{vw} = 1$ iff $b_{wv} = 1$ iff $w \in P$ and $v \in Q$ are adjacent;

then $C[u,v] = \sum_{w \in P} a_{uw} b_{wv}$. Conversely, to reduce matrix multiplication to the graph problem, let $P = \{1, \ldots, n\}$ and $Q = \{1, \ldots, q\} \times \{1, 2\}$ (of size $2q$), and place an edge between k and $(j,1)$ if $a_{jk} = 1$ and an edge between k and $(j,2)$ if $b_{jk} = 1$; then $C[(i,1),(j,2)] = \sum_{k=1}^{n} a_{ik} b_{kj}$.)

We are primarily interested in the case when q is small. For dense matrices, we have the upper bound $M(n, q \,|\, m) = O(nq^{\omega-1} + q^{\omega})$, since we can solve the problem by multiplying $\lceil n/q \rceil$ pairs of $q \times q$ square submatrices. To take the sparseness m into account, we adapt a simple trick by Alon, Yuster, and Zwick [4].

LEMMA 3.1. $M(n, q \,|\, m) = O(mq^{(\omega-1)/2} + q^{\omega})$.

Proof. Consider the graph formulation. Divide P into two groups: P_H, vertices of degree $> r$, and P_L, vertices of degree $\leq r$. Then $|P_H| = O(m/r)$. We first set $C[u,v]$ to be the number of vertices in P_H adjacent to both u and v for every pair $u, v \in Q$; this takes time $O(M(m/r, q \,|\, m)) = O(mq^{\omega-1}/r + q^{\omega})$ by the dense-matrix bound. To complete the overall count, we can examine each edge uw incident to a vertex $w \in P_L$, go through all adjacent edges wv, and increment $C[u,v]$; this takes $O(mr)$ time. Setting $r = q^{(\omega-1)/2}$ yields the desired bound. □

Remark. There are improved rectangular matrix multiplication methods [22] for dense matrices, which may lead to slight improvements to Lemma 3.1 for certain ranges of parameters. However, these improvements do not seem to matter here, since we will eventually select parameters to equalize the contribution of the terms $mq^{(\omega-1)/2}$ and q^{ω}, and the critical case will occur when the dense submatrices are essentially square matrices.

4. An offline solution. In the next three sections, we present our algorithms for dynamic subgraph connectivity. We begin by considering the offline case, where we are given the update sequence in advance (or the ability to look sufficiently far ahead in the update sequence). The approach is simple: we divide a dynamic set into two subsets, P and Q, where P is static and Q is dynamic but small; to keep Q small, we rebuild the data structure from scratch periodically, after a certain number of updates. Khanna, Motwani, and Wilson [24] have adopted this very approach for strongly connected components and reachability in directed graphs. We obtain a sublinear result in m by, in addition, employing Lemma 3.1.

In what follows, let $G = (V, E)$ be the input graph with n vertices and m edges. Without loss of generality, assume that $m = \Omega(n)$. For clarity's sake, we focus only on the more difficult update operations, i.e., vertex insertions/deletions in S; edge insertions/deletions in G will be treated later in the remarks after Theorem 6.2.

LEMMA 4.1. *Given a static subset $P \subseteq V$ and a static subset $Q_0 \subseteq V$ of size q_0, we can design a data structure to maintain a subset $Q \subseteq Q_0$, where preprocessing takes $\widetilde{O}(mq_0^{(\omega-1)/2} + q_0^{\omega})$ time, updates to Q take $\widetilde{O}(q_0)$ amortized time, and connectivity queries on the subgraph induced by $P \cup Q$ take $\widetilde{O}(q_0)$ time.*

Proof. We first compute the connected components of the subgraph induced by P in linear time by depth-first search [8]. Our data structure consists of three parts:

- We form a bipartite multigraph Γ with the connected components as vertices on one side and V as vertices on the other side: for each edge $uv \in E$ with $u \in P$, we place a corresponding edge γv in Γ for the component γ containing u. The $O(m)$ edges of Γ are stored in a dictionary [8]. (For example, a balanced search tree can support updates and lookups in $O(\log n)$ time; alternatively, hashing can support these operations in $O(1)$ randomized time.)

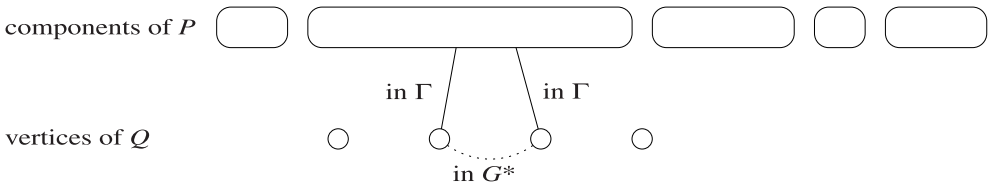


FIG. 4.1. *Overview of the data structure.*

- We precompute and store the following values:

$$C[u,v] \;=\; \text{number of components adjacent to both } u \text{ and } v \text{ in } \Gamma \tag{4.1}$$

  over all $u, v \in Q_0$. This preprocessing step takes $O(mq_0^{(\omega-1)/2} + q_0^\omega)$ time by applying Lemma 3.1 to a bipartite subgraph of Γ (with Q_0 on one side and edge multiplicities ignored).
- Finally, we maintain a small dynamic graph G^* over the vertex set Q, where for every $u, v \in Q$,

$$uv \text{ is in } G^* \text{ iff } C[u,v] > 0 \text{ or } uv \in E. \tag{4.2}$$

  (See Figure 4.1.) This graph is stored in a polylogarithmic data structure for dynamic graph connectivity. (For example, the method of Holm, de Lichtenberg, and Thorup [21] can support updates in $O(\log^2 n)$ amortized time and queries in $O(\log n / \log\log n)$ time; alternatively, Thorup's improved method [32] can support updates in $O(\log n \log^3 \log n)$ randomized amortized time and queries in $O(\log n / \log\log\log n)$ time.)

INSERTIONS/DELETIONS IN Q: To insert a vertex u to Q, we simply insert edges uv to G^* for all $v \in Q$ with $C[u,v] > 0$ or $uv \in E$. To delete u from Q, we delete all edges incident to u from G^*. For $|Q| = q$, this requires at most q edge updates to the dynamic graph connectivity structure for G^* and costs $\widetilde{O}(q)$ amortized time.

QUERIES ON $P \cup Q$: Given vertices $u, v \in P \cup Q$, we want to test whether u and v are connected in the subgraph induced by $P \cup Q$.

- EASIEST CASE: $u, v \in Q$. We can simply test whether u and v are connected in G^* in $\widetilde{O}(1)$ time by the dynamic graph connectivity structure for G^*. Correctness is evident (if there is a path connecting u and v in the subgraph induced by $P \cup Q$, then the path must alternate between some vertices of Q and some component of P, so there must be a corresponding path in G^*).
- HARDEST CASE: $u, v \in P$. Here we first find the components γ_u and γ_v containing u and v. We then find any $u' \in Q$ with $\gamma_u u' \in \Gamma$ and any $v' \in Q$ with $\gamma_v v' \in \Gamma$ by performing $O(q)$ dictionary lookups in $\widetilde{O}(q)$ time. If u' or v' does not exist, the answer is no, unless $\gamma_u = \gamma_v$. Otherwise, we can just test whether u' and v' are connected by the previous case.

The remaining cases are similar. ∎

THEOREM 4.2. *We can design a data structure to maintain a subset $S \subseteq V$ under any* offline *update sequence, where preprocessing takes $\widetilde{O}(m^{2\omega/(\omega+1)}) = O(m^{1.41})$ time, updates to S take $\widetilde{O}(m^{2(\omega-1)/(\omega+1)}) = O(m^{0.82})$ amortized time, and connectivity queries on the subgraph induced by S take $\widetilde{O}(m^{2/(\omega+1)}) = O(m^{0.60})$ time.*

Proof. At the beginning of each block of q_0 updates, we set Q_0 to be the vertices involved in the coming q_0 updates, set $P = S \setminus Q_0$, set $Q = S \cap Q_0$, and rebuild the data structure from Lemma 4.1. Updates to S are applied to Q, with amortized cost

$$\widetilde{O}\left(\frac{mq_0^{(\omega-1)/2} + q_0^{\omega}}{q_0} + q_0\right),$$

which is asymptotically minimized by setting $q_0 = m^{2/(\omega+1)}$. Queries can be answered in $\widetilde{O}(q_0)$ time, since $S = P \cup Q$ at all times. ∎

5. A semi-online solution. It is perhaps not surprising that an offline problem can be solved more quickly by batching and performing fast matrix multiplication. It is interesting, however, that with more effort a similar strategy can lead to a fully dynamic solution to our problem. Although we cannot predict which vertices are about to be inserted in advance, we can concentrate preprocessing on vertices of high degrees, since these vertices are the costly ones. To obtain a sublinear bound, we now have to rebuild the data structure more frequently, as the setting of parameters becomes more delicate.

Before describing the fully dynamic algorithm, we consider the semi-online case [10], where we are told when a vertex will next be deleted at the time it is inserted. Because of the extra information, we can force all deletions to occur in the small subset Q, thereby ensuring that P is static (see [6] for more examples of this kind of dynamization).

LEMMA 5.1. *Given a subset $P \subseteq V$ and a parameter $q_0 \leq m$, we can design a data structure to maintain a subset $Q \subseteq V$ of size $q \leq q_0$, where preprocessing takes $\widetilde{O}(mq_0^{(\omega-1)/2} + q_0^{\omega})$ time, updates to Q take $\widetilde{O}(mq/q_0)$ amortized time, and connectivity queries on the subgraph induced by $P \cup Q$ take $\widetilde{O}(q)$ time.*

Proof. Set Q_0 to contain all vertices in V of degree $> m/q_0$ in Γ. Then $|Q_0| = O(q_0)$. As in the proof of Lemma 4.1, we form the same bipartite multigraph Γ and precompute the same values $C[u, v]$ (as defined by (4.1)) for all $u, v \in Q_0$. In addition, we now maintain $C[u, v]$ for all $u, v \in Q$ as well, in order to keep track of the graph G^* over the vertex set Q (as defined by (4.2)). Preprocessing time is still $O(mq_0^{(\omega-1)/2} + q_0^{\omega})$, and queries can be answered in the same way in $\widetilde{O}(q)$ time.

INSERTIONS/DELETIONS IN Q: Deletions are as before. Insertions are more involved, because Q is not necessarily a subset of Q_0, so new entries of $C[\cdot, \cdot]$ need to be computed. To insert a vertex u into Q, consider two cases.

- CASE 1: $u \notin Q_0$; i.e., u has degree $\leq m/q_0$ in Γ. For each $v \in Q$, we can compute $C[u, v]$ by going through each of the $O(m/q_0)$ different components γ adjacent to u in Γ and testing whether $\gamma v \in \Gamma$ in $\widetilde{O}(1)$ time via a dictionary lookup. The total time of this step is $\widetilde{O}(qm/q_0)$.
- CASE 2: $u \in Q_0$. For each $v \in Q_0$, $C[u, v]$ has already been precomputed. For each $v \in Q \setminus Q_0$, since v has degree $\leq m/q_0$ in Γ, we can compute $C[u, v]$ by going through each of the $O(m/q_0)$ different components γ adjacent to v in Γ and testing whether $\gamma u \in \Gamma$ in $\widetilde{O}(1)$ time via a dictionary lookup. The total time in this case is also $\widetilde{O}(qm/q_0)$.

We can now maintain the graph G^*, as in the proof of Lemma 4.1, by at most q edge updates to the dynamic graph connectivity structure, in additional $\widetilde{O}(q)$ time. ∎

THEOREM 5.2. *We can design a data structure to maintain a subset $S \subseteq V$ under any* semi-online *update sequence, where preprocessing takes $\widetilde{O}(m^{2\omega/(\omega+1)}) = O(m^{1.41})$ time, updates to S take $\widetilde{O}(m^{(3\omega-1)/(2\omega+2)}) = O(m^{0.91})$ amortized time, and connectivity queries on the subgraph induced by S take $\widetilde{O}(m^{1/2})$ time.*

Proof. At the beginning of each block of q updates, we set Q to contain the vertices in S with the q smallest deletion times, set $P = S \setminus Q$, and rebuild the data structure from Lemma 5.1. Each insertion in S is applied to Q with $|Q| \le 2q$ at all times, and deletions in S can occur to Q only. The amortized update cost is

$$\widetilde{O}\left(\frac{mq_0^{(\omega-1)/2} + q_0^{\omega}}{q} + \frac{mq}{q_0}\right),$$

which is asymptotically minimized by setting $q_0 = m^{2/(\omega+1)}$ and $q = m^{1/2}$. ∎

6. A fully dynamic solution. In the fully dynamic problem, we have no control on future deletions, so the subset P can no longer remain static. However, since P undergoes deletions only, we can bring in amortization techniques (see [5] for a similar, geometric example). Deletions cause splitting of components, and a standard idea is to always split the smaller set from the larger set. Remarkably, the matrix-multiplication output can be updated efficiently during this process, due to the linear dependence on m in Lemma 3.1.

LEMMA 6.1. *We can make the data structure in Lemma* 5.1 *support an additional operation: deletion in P. The total cost of ℓ such deletions is bounded by $\widetilde{O}(mq_0^{(\omega-1)/2} + \ell q_0^{\omega})$.*

Proof. The data structure is the same as in the proof of Lemma 5.1 but with one additional ingredient: To maintain the connected components of P, we store the subgraph induced by P in a decremental graph connectivity data structure [17, 31, 32]; the total maintenance cost over a sequence of deletions is $\widetilde{O}(m)$. Note that such a structure can handle auxiliary operations, such as reporting the size of a component or enumerating the vertices of a component. Insertions and deletions in Q are done exactly as in the proof of Lemma 5.1; it remains to describe how to perform deletions in P.

DELETIONS IN P: To delete a vertex w from P, we find the component γ containing w and observe how γ is split into several components $\gamma_1, \ldots, \gamma_k$ from the decremental graph connectivity structure. Without loss of generality, suppose that γ_1 has the largest size. Note that each γ_i $(i = 2, \ldots, k)$ has at most half the size of γ. Let m' be the sum of the degrees over all vertices of $\{w\} \cup \gamma_2 \cup \cdots \cup \gamma_k$ in G. (See Figure 6.1.)

Both Γ and $C[\cdot, \cdot]$ change as a result of the split.

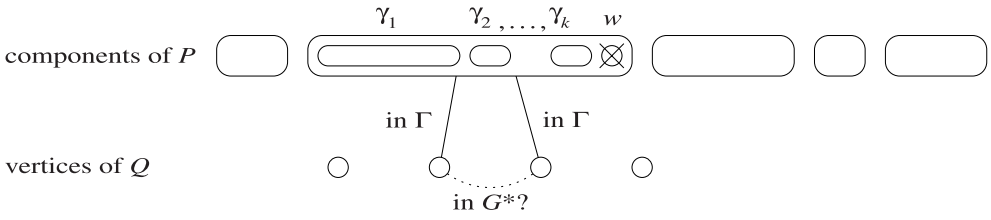


FIG. 6.1. *Making the data structure fully dynamic.*

- To update the multigraph Γ, we perform the following steps: For each edge $uv \in E$ with $u \in \gamma_i$ $(i = 2, \ldots, k)$, we insert a copy of $\gamma_i v$ and remove a copy of γv. For each $wv \in E$, we remove a copy of γv. Finally, we remove $\{w\} \cup \gamma_2 \cup \cdots \cup \gamma_k$ from γ so that γ becomes γ_1. These steps require $O(m')$ dictionary updates and take $\widetilde{O}(m')$ time.
- To update the $C[\cdot, \cdot]$ values, we perform the following steps: For each $(u, v) \in (Q_0 \times Q_0) \cup (Q \times Q)$, if $\gamma u, \gamma v \in \Gamma$ before, we decrement $C[u, v]$. For each $(u, v) \in (Q_0 \times Q_0) \cup (Q \times Q)$, if $\gamma_1 u, \gamma_1 v \in \Gamma$, we increment $C[u, v]$. These steps require $O(q^2)$ dictionary lookups and take $\widetilde{O}(q_0^2)$ time. Finally, for each $(u, v) \in (Q_0 \times Q_0) \cup (Q \times Q)$, we add to $C[u, v]$ the number of components from $\{\gamma_2, \ldots, \gamma_k\}$ adjacent to both u and v in Γ. These numbers can be computed in $O(m' q_0^{(\omega-1)/2} + q_0^\omega)$ total time by applying Lemma 3.1 to a bipartite subgraph of Γ with $O(m')$ edges (with $\gamma_2, \ldots, \gamma_k$ on one side and Q_0 or Q on the other).

We can now rebuild the graph G^* from scratch (according to the definition (4.2)) in time $O(q^2)$, which is absorbed by the other cost.

We can account for the $O(m' q_0^{(\omega-1)/2} + q_0^\omega)$ cost by charging $O(\deg(v) q_0^{(\omega-1)/2})$ units to each vertex $v \in \{w\} \cup \gamma_2 \cup \cdots \cup \gamma_k$ and charging $O(q_0^\omega)$ units to the deletion operation itself. Here $\deg(v)$ denotes the degree of v in G. Each vertex v is charged at most $O(\log n)$ times overall, since each time v is charged, the component containing v shrinks at least by a factor of two in size (and components never expand, as P undergoes only deletions). Therefore, the total cost charged to vertices is bounded by $O(\sum_{v \in V} \deg(v) q_0^{(\omega-1)/2} \log n) = \widetilde{O}(m q_0^{(\omega-1)/2})$. The total cost charged to the deletion operations is $O(\ell q_0^\omega)$. ∎

THEOREM 6.2. *We can design a data structure to maintain a subset $S \subseteq V$ under any* online *update sequence, where preprocessing takes $\widetilde{O}(m^{(5\omega+1)/(3\omega+3)}) = O(m^{1.28})$ amortized time, updates to S take $\widetilde{O}(m^{4\omega/(3\omega+3)}) = O(m^{0.94})$ amortized time, and connectivity queries on the subgraph induced by S take $\widetilde{O}(m^{1/3})$ time.*

Proof. At the beginning of each block of q updates, we set $P = S$, set $Q = \emptyset$, and rebuild the data structure from Lemma 6.1. Insertions into S are applied to Q, with $|Q| \le q$ at all times, but deletions can occur to both P and Q. The amortized update cost is

$$\widetilde{O}\left(\frac{m q_0^{(\omega-1)/2} + q q_0^\omega}{q} + \frac{mq}{q_0}\right),$$

which is asymptotically minimized by setting $q_0 = q^{4/(\omega+1)}$ and $q = m^{1/3}$. ∎

Remarks. The space requirement of the data structure is indeed $\widetilde{O}(m)$: The dynamic graph connectivity structures take $\widetilde{O}(m)$ space, the dictionary for Γ takes $O(m)$ space, and the $C[\cdot, \cdot]$ entries take $O(q_0^2)$ space, which is sublinear for the parameters chosen in the proof of Theorem 6.2.

We can obtain a query-update trade-off version of Theorem 6.2: With $\widetilde{O}(q)$ query time for any given parameter $q \le m^{1/3}$, the amortized update time is $\widetilde{O}(m/q^{(3-\omega)/(\omega+1)}) = \widetilde{O}(m/q^{0.18})$.

Edge updates are indeed less difficult than vertex updates: For example, to insert a new edge vw into E, we can create a new dummy vertex u joined to v and w and then insert u into S (i.e., Q). In the proof of Lemma 5.1, since u has degree 2 only, we can update Γ in $\widetilde{O}(1)$ time and follow Case 1 to update $C[\cdot, \cdot]$ in less than $\widetilde{O}(q_0 m/q)$ time (note that Q_0 does not change). Later, to delete the edge vw, we can simply

delete the dummy vertex u from S. Therefore, the cost of an edge insertion/deletion is at most the cost of a vertex insertion/deletion.

7. Discussions.

7.1. Is fast matrix multiplication necessary? Our algorithms have limited practical appeal because of the use of fast matrix multiplication (FMM). One may wonder whether FMM is essential to solve our problem. We suspect that the answer might be yes, in view of the following observations.

OBSERVATION 7.1.

1. *The problem of multiplying a $\sqrt{n} \times n$ Boolean matrix with an $n \times \sqrt{n}$ Boolean matrix with m nonzero entries can be reduced to offline dynamic subgraph connectivity on a graph with n vertices and m edges using $O(n)$ updates and queries.*

2. *The problem of detecting a triangle (a 3-cycle) in a directed graph with m edges can be reduced to offline dynamic subgraph connectivity using $O(m)$ updates and queries.*

3. *The problem of detecting a quadrilateral (a 4-cycle) in a directed graph with m edges can be reduced to offline dynamic subgraph connectivity using $O(m)$ updates and queries.*

Proof.

1. As noted essentially in section 3, an equivalent problem is the following: given a bipartite graph G with a set P of n vertices on one side and a set Q of $O(\sqrt{n})$ vertices on the other side, decide whether u and v are adjacent to a common vertex for every pair $u, v \in Q$. To solve this problem, we first put all vertices of P in the subset S. For each pair $u, v \in Q$, we insert u and v into S, test whether u and v are connected in the subgraph induced by S, and then delete u and v from S. The number of queries and vertex updates to S is $O(|Q|^2) = O(n)$.

2. Let $H = (V, E)$ be the given graph. Define an undirected graph G with vertex set $V \times \{(1, 2, 3)\}$, where we create edges $(u, 1)(v, 2)$ and $(u, 2)(v, 3)$ whenever $uv \in E$. Initially, we put all vertices of the form $(w, 2)$ in S. To detect a triangle in H, we go through each edge $(v, u) \in E$, insert $(u, 1)$ and $(v, 3)$ into S, test whether $(u, 1)$ and $(v, 3)$ are connected, and then delete $(u, 1)$ and $(v, 3)$ from S. The answer is yes iff one of the tests returns true.

3. Define G as before, but with the addition of a vertex s adjacent to all vertices of the form $(u, 1)$, and another vertex t adjacent to all vertices of the form $(v, 3)$. Initially, we put s and t in S, along with all vertices of the form $(w, 2)$. To detect a quadrilateral in H, we go through each vertex $z \in V$, insert all vertices into the subsets $A = \{(u, 1) \mid (z, u) \in E\}$ and $B = \{(v, 3) \mid (v, z) \in E\}$ to S, test whether s and t are connected, and then delete all vertices in A and B from S. The answer is yes iff one of the tests returns true (if s and t are connected, there is a length-2 path from A to B, yielding a quadrilateral through z). The number of vertex updates to S is $O(\sum_{z \in V} \deg(z)) = O(m)$. ∎

For the first problem, the best bound we know that does not use FMM is $O(m\sqrt{n})$ (by Lemma 3.1 with $\omega = 3$). So, it is unlikely that offline dynamic subgraph connectivity can be solved in $o(n^{1/2})$ time without some kind of FMM (though this does not rule out the possibility of a sublinear bound without FMM). For the second and third problem, the best algorithms known for sparse graphs, due to Alon, Yuster, and Zwick [4] and Yuster and Zwick [33], both require FMM and run in time $O(m^{2\omega/(\omega+1)}) = O(m^{1.41})$ and (a little less than) $O(m^{(4\omega-1)/(2\omega+1)}) = O(m^{1.48})$, respectively.

7.2. Is the graph problem necessary? The problem we start with is geometric, but the solution we give is mostly graph-theoretic. One may wonder whether this is the right approach. For $d = 3$, the answer is yes, as shown below.

OBSERVATION 7.2. *The dynamic subgraph connectivity problem can be reduced to the dynamic box connectivity problem in* $\mathbb{R}^3$.

Proof. We use only orthogonal segments (degenerate boxes) in $\mathbb{R}^3$: For each vertex i of the given graph, construct a line ℓ_i from $(i, -\infty, 0)$ to $(i, \infty, 0)$. For the kth edge ij, create a path π_k of three segments through the points $(i, k, 0), (i, k, 1), (j, k, 1), (j, k, 0)$. Then i and j are connected iff ℓ_i and ℓ_j are connected. Inserting/deleting a vertex i in S corresponds to inserting/deleting ℓ_i. Inserting/deleting an edge in E corresponds to inserting/deleting a π_k. □

Thus, by Theorem 2.2, subgraph connectivity is *equivalent* to box connectivity in three dimensions, up to polylogarithmic factors. In particular, box connectivity in any fixed dimension ≥ 3 is equally difficult, up to polylogarithmic factors. It is intriguing to contemplate whether one can exploit the geometry of the rectangular connectivity problem to get a faster algorithm for $d = 2$ (perhaps without FMM).

7.3. Global connectivity? On a final note, we have purposely defined connectivity queries as just deciding whether two points, or two vertices, are connected. One may wonder about other kinds of connectivity queries; for example, "Is the union of the boxes connected?", or "Is the subgraph induced by S connected?"

Obtaining nontrivial results for such queries appears difficult, even in the offline setting. A major obstacle appears to be the following dynamic set union problem: Given a collection $\mathcal{C}$ of n subsets of U, of total size m, maintain a subcollection $\mathcal{S} \subseteq \mathcal{C}$ under insertions and deletions of subsets to answer the following query: "Is the union of $\mathcal{S}$ equal to U?"

REFERENCES

[1] P. K. AGARWAL, N. ALON, B. ARONOV, AND S. SURI, *Can visibility graphs be represented compactly?*, Discrete Comput. Geom., 12 (1994), pp.347–365.

[2] P. K. AGARWAL AND J. ERICKSON, *Geometric range searching and its relatives*, in Advances in Discrete and Computational Geometry, B. Chazelle, J. E. Goodman, and R. Pollack, eds., AMS, Providence, RI, 1999, pp. 1–56.

[3] P. K. AGARWAL AND M. VAN KREVELD, *Polygon and connected component intersection searching*, Algorithmica, 15 (1996), pp. 626–660.

[4] N. ALON, R. YUSTER, AND U. ZWICK, *Finding and counting given length cycles*, Algorithmica, 17 (1997), pp. 209–223.

[5] T. M. CHAN, *A fully dynamic algorithm for planar width*, Discrete Comput. Geom., 30 (2003), pp. 17–24.

[6] T. M. CHAN, *Semi-online maintenance of geometric optima and measures*, SIAM J. Comput., 32 (2003), pp. 700–716.

[7] D. COPPERSMITH AND S. WINOGRAD, *Matrix multiplication via arithmetic progressions*, J. Symbolic Comput., 9 (1990), pp. 251–280.

[8] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST, AND C. STEIN, *Introduction to Algorithms*, 2nd ed., MIT Press, Cambridge, MA, 2001.

[9] C. DEMETRESCU AND G. F. ITALIANO, *Trade-offs for fully dynamic transitive closure on DAGs: Breaking through the* $O(n^2)$ *barrier*, J. ACM, 52 (2005), pp. 147–156.

[10] D. DOBKIN AND S. SURI, *Maintenance of geometric extrema*, J. Assoc. Comput. Mach., 38 (1991), pp. 275–298.

[11] H. EDELSBRUNNER AND H. A. MAURER, *On the intersection of orthogonal objects*, Inform. Process. Lett., 13 (1981), pp. 177–181.

[12] D. EPPSTEIN, *Dynamic Euclidean minimum spanning trees and extrema of binary functions*, Discrete Comput. Geom., 13 (1995), pp. 111–122.

[13] D. EPPSTEIN, Z. GALIL, G. F. ITALIANO, AND A. NISSENZWEIG, *Sparsification: A technique for speeding up dynamic graph algorithms*, J. ACM, 44 (1997), pp. 669–696.

[14] T. Feder and R. Motwani, *Clique partitions, graph compression and speeding up algorithms*, J. Comput. System Sci., 51 (1995), pp. 261–272.
[15] G. N. Frederickson, *Data structures for on-line updating of minimum spanning trees, with applications*, SIAM J. Comput., 14 (1985), pp. 781–798.
[16] D. Frigioni and G. F. Italiano, *Dynamically switching vertices in planar graphs*, Algorithmica, 28 (2000), pp. 76–103.
[17] M. R. Henzinger and V. King, *Randomized dynamic graph algorithms with polylogarithmic time per operation*, J. ACM, 46 (1999), pp. 502–516.
[18] M. R. Henzinger and V. King, *Maintaining minimum spanning forests in dynamic graphs*, SIAM J. Comput., 31 (2001), pp. 364–374.
[19] J. Hershberger and S. Suri, *Kinetic connectivity of rectangles*, in Proceedings of the Fifteenth Annual Symposium on Computational Geometry, Miami, FL, 1999, pp. 237–246.
[20] J. Hershberger and S. Suri, *Simplified kinetic connectivity for rectangles and hypercubes*, in Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, Washington, D.C., 2001, pp. 158–167.
[21] J. Holm, K. de Lichtenberg, and M. Thorup, *Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree,* 2*-edge, and biconnectivity*, J. ACM, 48 (2001), pp. 723–760.
[22] X. Huang and V. Y. Pan, *Fast rectangular matrix multiplication and applications*, J. Complexity, 14 (1998), pp. 257–299.
[23] H. Imai and T. Asano, *Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane*, J. Algorithms, 4 (1983), pp. 310–323.
[24] S. Khanna, R. Motwani, and R. H. Wilson, *On certificates and lookahead on dynamic graph problems*, Algorithmica, 21 (1998), pp. 377–394.
[25] M. A. Lopez and R. Thurimella, *On computing connected components of line segments*, IEEE Trans. Comput., 44 (1995), pp. 597–601.
[26] K. Mehlhorn, *Data Structures and Algorithms* 3*: Multi-Dimensional Searching and Computational Geometry*, Springer-Verlag, Heidelberg, 1984.
[27] K. Mulmuley, *Computational Geometry: An Introduction Through Randomized Algorithms*, Prentice–Hall, Englewood Cliffs, NJ, 1994.
[28] M. H. Overmars, *The Design of Dynamic Data Structures*, Lecture Notes in Comput. Sci. 156, Springer-Verlag, Berlin, 1983.
[29] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.
[30] L. Roditty and U. Zwick, *Improved dynamic reachability algorithms for directed graphs*, in Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, Vancouver, BC, Canada, 2002, pp. 679–689.
[31] M. Thorup, *Decremental dynamic connectivity*, J. Algorithms, 33 (1999), pp. 229–243.
[32] M. Thorup, *Near-optimal fully-dynamic graph connectivity*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, Portland, OR, 2000, pp. 343–350.
[33] R. Yuster and U. Zwick, *Detecting short directed cycles using rectangular matrix multiplication and dynamic programming*, in Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA, 2004, pp. 247–253.
[34] R. Yuster and U. Zwick, *Fast sparse matrix multiplication*, ACM Trans. Algorithms, 1 (2005), pp. 2–13.

SIAM J. COMPUT.
Vol. 36, No. 3, pp. 695–720

ON THE NUMBER OF CROSSING-FREE MATCHINGS, CYCLES, AND PARTITIONS*

MICHA SHARIR† AND EMO WELZL‡

Abstract. We show that a set of n points in the plane has at most $O(10.05^n)$ perfect matchings with crossing-free straight-line embedding. The expected number of perfect crossing-free matchings of a set of n points drawn independently and identically distributed from an arbitrary distribution in the plane is at most $O(9.24^n)$. Several related bounds are derived: (a) The number of all (not necessarily perfect) crossing-free matchings is at most $O(10.43^n)$. (b) The number of *red-blue* perfect crossing-free matchings (where the points are colored red or blue and each edge of the matching must connect a red point with a blue point) is at most $O(7.61^n)$. (c) The number of *left-right* perfect crossing-free matchings (where the points are designated as left or right endpoints of the matching edges) is at most $O(5.38^n)$. (d) The number of perfect crossing-free matchings across a line (where all the matching edges must cross a fixed halving line of the set) is at most 4^n. These bounds are employed to infer that a set of n points in the plane has at most $O(86.81^n)$ crossing-free spanning cycles (simple polygonizations) and at most $O(12.24^n)$ crossing-free partitions (these are partitions of the point set so that the convex hulls of the individual parts are pairwise disjoint). We also derive lower bounds for some of these quantities.

Key words. crossing-free geometric graphs, counting, simple polygonizations, crossing-free matchings, crossing-free partitions

AMS subject classifications. 52C10, 52C45, 52A40, 68R05

DOI. 10.1137/050636036

1. Introduction.

Let P be a set of n points in the plane. A *geometric graph* on P is a graph that has P as its vertex set and whose edges are drawn as straight segments connecting the corresponding pairs of points. The graph is *crossing-free* if no pair of its edges cross each other; i.e., any two edges are not allowed to share any points other than common endpoints. Therefore, these are planar graphs with a plane embedding given by this specific drawing. We are interested in the number of crossing-free geometric graphs on P of several special types; see, e.g., Figure 1. Specifically, we consider the numbers $\mathsf{tr}(P)$ of *triangulations* (i.e., maximal crossing-free graphs), $\mathsf{pm}(P)$ of crossing-free *perfect matchings*, $\mathsf{sc}(P)$ of crossing-free *spanning cycles*, and $\mathsf{cfp}(P)$ of *crossing-free partitions*[1] (these are partitions of P so that the convex hulls of the individual parts are pairwise disjoint). We are primarily concerned with upper bounds for the numbers listed above in terms of n.

*Received by the editors July 14, 2005; accepted for publication (in revised form) April 14, 2006; published electronically October 12, 2006. An extended abstract of this work appeared in [36].
http://www.siam.org/journals/sicomp/36-3/63603.html

†School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012 (michas@tau.ac.il). This author's work was supported by a grant from the U.S.–Israel Binational Science Foundation, by NSF grant CCR-00-98246, by a grant from the Israeli Academy of Sciences for a Center of Excellence in Geometric Computing at Tel Aviv University, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

‡Institute of Theoretical Computer Science, ETH Zurich, 8092 Zurich, Switzerland (emo@inf.ethz.ch).

[1]Our research was triggered by Marc van Kreveld asking about the number of crossing-free partitions (see [10] for a motivation from geographic information systems) and, in the same week, by Michael Hoffmann and Yoshio Okamoto asking about the number of crossing-free spanning paths of a point set (motivated by their quest for good fixed parameter algorithms for the planar Euclidean traveling salesman problem in the presence of a fixed number of inner points [14]); see also [23].

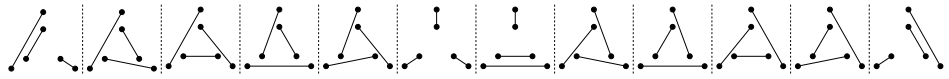


FIG. 1. 6 *points with* 12 *crossing-free perfect matchings, the maximum possible number. See* [3] *for the maximum numbers for up to* 10 *points:* 3 *for* 4 *points,* 12 *for* 6, 56 *for* 8, *and* 311 *for* 10.

History. This problem goes back to Newborn and Moser [32] in 1980 who asked for the maximal possible number of crossing-free spanning cycles in a set of n points[2]—they provide an upper bound of $2 \cdot 6^{n-2} \left\lfloor \frac{n}{2} \right\rfloor !$ but conjectured that the right bound should be of the form c^n for some constant c. This fact was established in 1982 by Ajtai et al. [4], who showed[3] that there are at most 10^{13n} crossing-free graphs on n points. For motivation they mentioned—besides [32]—a question of Avis about the maximum number of triangulations a set of n points can have.

Further developments were mainly concerned with deriving progressively better upper bounds for the number of triangulations[4] [38, 17, 35], thus far culminating in a 59^n upper bound by Santos and Seidel [34] in 2003.[5] This compares to $\Omega(8.48^n)$, the largest known number of triangulations for a set of n points, recently derived by Aichholzer et al. [1]; this improves an earlier lower bound of $8^n/\mathrm{poly}(n)$ given by García, Noy, and Tejel [21]. (We let "poly(n)" denote a polynomial factor in n.)

Since every crossing-free graph is contained in some triangulation, and a triangulation has at most $3n-6$ edges, an upper bound of c^n for the number of triangulations immediately yields an upper bound of $2^{3n-6}c^n < (8c)^n$ for the number of all crossing-free graphs on a set of n points. Thus, with $c \leq 59$, this number is at most 472^n. To the best of our knowledge, *all* upper bounds derived so far on the number of crossing-free graphs of various types are derived via a bound on the number of triangulations, albeit in more refined ways.

One such approach is to exploit the fact that graphs of certain specific types have a fixed number of edges. For example, since a perfect matching has $\frac{n}{2}$ edges, we readily obtain $\mathsf{pm}(P) \leq \binom{3n-6}{n/2} \mathsf{tr}(P) < 227.98^n$ [18]. A short historical account of bounds on $\mathsf{sc}(P)$, with references including [6, 16, 21, 22, 24, 32, 33], can be found at the web site [15] (see also [12, section 8.4, Problem 8]). The best bound published so far is $3.37^n \cdot \mathsf{tr}(P) \leq 198.83^n$, which relies on a bound of 3.37^n on the number of cycles in a planar graph [7]. (In the course of our investigations, we showed that a graph with m edges and n vertices can have at most $\left(\frac{m}{n}\right)^n$ cycles; hence, a planar graph can have at most 3^n cycles. Then Seidel provided us with an argument, based on linear algebra, that a planar graph can have at most $\sqrt{6}^n < 2.45^n$ spanning cycles.)

Crossing-free partitions fit into the picture, since every such partition can be uniquely identified with the graph of edges of the convex hulls of the individual parts—these edges form a crossing-free geometric graph of at most n edges; see Figure 2.

[2]In fact, Akl's work [6] appeared earlier, but it already referred to the manuscript by Newborn and Moser, and improved a lower bound (on the maximal number of crossing-free spanning cycles) of theirs.

[3]This paper is famous for its *crossing lemma*, proved in preparation of the singly exponential bound. The lemma gives an upper bound on the number of edges a geometric graph with a given number of crossings can have.

[4]Interest was also motivated by the obviously related question (from geometric modeling [38]) of how many bits it takes to encode a triangulation of a point set.

[5]Recently, this bound was improved to 43^n in [37]. However, the bounds on spanning cycles and crossing-free partitions we derive here via matchings are still better than the bounds obtained via this new triangulation bound.

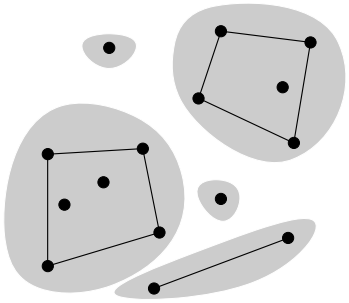


FIG. 2. *A crossing-free partition and its graph.*

The situation is better understood for special configurations, for example, for P a set of n points in convex position[6] (namely, the vertex set of a convex n-gon), where the *Catalan numbers* $C_m := \frac{1}{m+1}\binom{2m}{m} = \Theta(m^{-3/2}4^m)$, $m \in \mathbb{N}_0$, play a prominent role. In convex position $\mathsf{tr}(P) = C_{n-2}$ (the Euler–Segner problem; cf. [39, p. 212] for a discussion of its history), $\mathsf{pm}(P) = C_{n/2}$ for n even (due to [20]; cf. [39]; e.g., see Figure 3), $\mathsf{sc}(P) = 1$, and $\mathsf{cfp}(P) = C_n$ (see [9]).

Crossing-free partitions for points in convex position constitute a well-established notion because of many connections to other problems, probably starting with "planar rhyme schemes" in Becker's note [9]; cf. [39, Solution to 6.19pp]. The general case was considered by [13] (under the name of pairwise linearly separable partitions) for clustering algorithms. They show that the number of partitions into k parts is $O(n^{6k-12})$ for k constant.

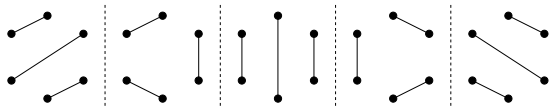


FIG. 3. *6 points in convex position with $C_3 = 5$ crossing-free perfect matchings.*

Under the assumption of *general position* (no three points on a common line) it is known [21] that the number of crossing-free perfect matchings on a set of fixed size is minimized when the set is in convex position.[7] With little surprise, the same holds for spanning cycles, but it does not hold for triangulations [2, 25, 30]. For crossing-free partitions, this is an open question.

New results. The main results of this paper are the following upper bounds for a set P of n points in the plane: $\mathsf{pm}(P) = O(10.05^n)$, $\mathsf{sc}(P) = O(86.81^n)$, and $\mathsf{cfp}(P) = O(12.24^n)$. Also, the expected number of perfect crossing-free matchings of a set of n points drawn independently and identically distributed (i.i.d.) from *any* distribution in the plane (as long as two random points coincide with probability 0) is at most $O(9.24^n)$.

The new bound on the number of crossing-free perfect matchings is derived by an inductive technique that we have adapted from the method that Santos and Seidel [34] used for triangulations (the adaption is, however, far from obvious). We then

[6] For another example, it can be shown that the number of triangulations is at most $2^{3mn-m-n}$ for an $m \times n$ grid (with $(m+1)(n+1)$ points) [5] (cf. also [26]).

[7] Recently, Aichholzer et al. [1] showed that any family of acyclic graphs has the minimal number of crossing-free embeddings on a fixed point set when the set is in convex position.

go on to derive several improved bounds on the number of crossing-free matchings of various special types. Specifically, we show the following:

(a) The number of all (not necessarily perfect) crossing-free matchings is at most $O(10.43^n)$.

(b) The number of *red-blue* perfect crossing-free matchings (where half of the points are colored red and half blue, and each edge of the matching must connect a red point with a blue point) is at most $O(7.61^n)$.

(c) The number of *left-right* perfect crossing-free matchings (where the points are designated as left or right endpoints of the matching edges) is at most $O(5.38^n)$.

(d) The number of perfect crossing-free matchings across a line (where all the matching edges must cross a fixed halving line of the set) is at most 4^n.

Finally, we derive upper bounds for the numbers of crossing-free spanning cycles and crossing-free partitions of P in terms of the number of certain types of matchings of certain point sets P' that are constructed from P. This yields the bounds $O(86.81^n)$ for the number of crossing-free cycles and $O(12.24^n)$ for the number of crossing-free partitions.

We summarize the state of affairs in Table 1, including lower bounds which we will derive in section 6, many of which use the *double-chain* configuration from [21].

TABLE 1

*Entries c in the upper bound row should be read as $O(c^n)$, and entries c in the lower bound row should be read as $\Omega(c^n/\mathrm{poly}(n))$, where $n := |P|$. "*ma*" stands for all (not necessarily perfect) crossing-free matchings, "*rbpm*" for perfect red-blue crossing-free matchings, "*lrpm*" for perfect left-right crossing-free matchings, "*alpm*" for perfect crossing-free matchings across a line, and "*rdpm*" for the expected number of perfect crossing-free matchings of a set of i.i.d. points.*

	tr	pm	sc	cfp	ma	rbpm	lrpm	alpm	rdpm
$\forall P :\leq$	59 [34]	10.05	86.81	12.24	10.43	7.61	5.38	4	9.24
$\exists P :\geq$	8.48 [1]	3 [21]	4.64 [21]	5.23	4	2.23	2	2	3

This paper shows that significantly better bounds can be derived for matchings than those known earlier for other types of graphs and, moreover, that matchings are a good basis for deriving bounds for crossing-free partitions and spanning cycles—as opposed to the situation before, where such bounds have always relied on triangulations.

2. Matchings: The setup and a recurrence. Let P be a set of n points in the plane in general position, no three on a line, no two on a vertical line. It is easy to see that this is no constraint when it comes to upper bounds on $\mathsf{pm}(P)$. A *crossing-free matching* is a collection of pairwise disjoint segments whose endpoints belong to P. Given such a matching M, each point of P is either *matched* if it is an endpoint of a segment of M, or *isolated* otherwise. The number of matched points is clearly always even. If $2m$ points are matched and s points are isolated, we call M a *crossing-free m-matching* or *(m, s)-matching*. We have $n = 2m + s$.

We denote by $\mathsf{ma}_m(P)$ the number of crossing-free matchings of P with m segments (for $m \in \mathbb{R}$—this number is clearly 0 unless $m \in \{0, 1, \ldots, \lfloor \frac{n}{2} \rfloor\}$), and by $\mathsf{ma}(P)$ the number of all crossing-free matchings of P (i.e., $\mathsf{ma}(P) = \sum_m \mathsf{ma}_m(P)$). Recall that $\mathsf{pm}(P) = \mathsf{ma}_{n/2}(P)$.

Let M be a crossing-free (m, s)-matching on a set P of $n = 2m + s$ points, as above. The *degree* $d(p)$ of a point $p \in P$ in M is defined as follows. It is 0 if p is isolated in M. Otherwise, if p is a left (resp., right) endpoint of a segment of M, $d(p)$ is equal to the number of visible left (resp., right) endpoints of other segments of M,

plus the number of visible isolated points; "*visible*" means *vertically* visible from the *relative interior* of the segment of M that has p as an endpoint. Thus p and the other endpoint of the segment are not counted in $d(p)$. See Figure 4 for an illustration.

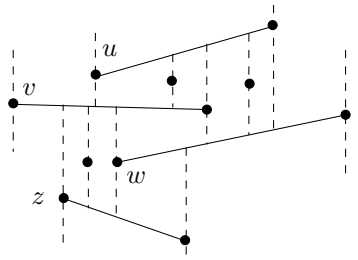


FIG. 4. *Degrees in a matching:* $d(u) = 2$, $d(v) = 5$, $d(w) = 1$, $d(z) = 2$.

Each left (resp., right) endpoint u in M can contribute at most 2 to the degrees of other points: 1 to each of the left (resp., right) endpoints of the segments lying vertically above and below u if there exist such segments. Similarly, each isolated point u can contribute at most 4 to the degrees of other points: 1 to each of the endpoints of the segments lying vertically above and below u. It follows that

$$\sum_{p\in P} d(p) \le 4m + 4s.$$

There are many segments ready for removal. The idea is to remove segments incident to points of low degree in an (m, s)-matching (points of degree at most 3 or at most 4, to be specific). We will show that there are many such points at our disposal. Then, in the next step, we will show that segments with an endpoint of low degree can be reinserted in not very many ways. These two facts will be combined to derive a recurrence for the matching count.

For each integer $i \in \mathbb{N}_0$, let $v_i = v_i(M)$ denote the number of *matched* points of P with degree i in M. Hence, $\sum_{i\ge 0} v_i = 2m$.

LEMMA 2.1. *Let $n, m, s \in \mathbb{N}_0$, with $n = 2m + s$. In every (m, s)-matching of any set of n points, we have*

$$2n \le 4v_0 + 3v_1 + 2v_2 + v_3 + 6s, \tag{1}$$

$$3n \le 5v_0 + 4v_1 + 3v_2 + 2v_3 + v_4 + 7s. \tag{2}$$

Proof. Let P be the underlying point set. We have

$$\sum_{i\ge 0} i\, v_i = \sum_{p\in P} d(p) \le 4s + 4m = 4s + \sum_{i\ge 0} 2v_i.$$

Therefore, $0 \le 4s + \sum_{i\ge 0}(2 - i)v_i$. For $\kappa \in \mathbb{R}^+$, we add κ times $n = s + \sum_{i\ge 0} v_i$ to both sides to get

$$\kappa n \le (4 + \kappa)s + \sum_{i\ge 0}(2 + \kappa - i)v_i \le (4 + \kappa)s + \sum_{0\le i<2+\kappa} (2 + \kappa - i)v_i. \tag{3}$$

We specialize[8] to $\kappa = 2$ for assertion (1) and $\kappa = 3$ for (2). □

[8] We list here explicitly the two values that lead to the best results in the further derivations, although at this point it clearly looks rather arbitrary.

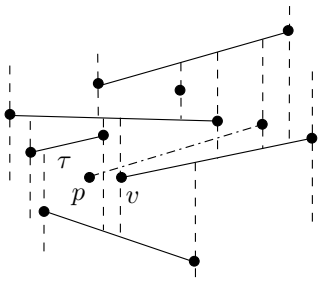


FIG. 5. *Inserting a segment at p; $d(p) = 1$ after insertion.*

There are not very many ways of inserting a segment. Fix some $p \in P$ and let M be a crossing-free matching which leaves p isolated. Now we match p with some other isolated point such that the overall matching continues to be crossing-free. For $i \in \mathbb{N}_0$, let $h_i = h_i(p, P, M)$ be the number of ways this can be done so that p has degree i after its insertion.

LEMMA 2.2.

$$4h_0 + 3h_1 + 2h_2 + h_3 \leq 24, \tag{4}$$

$$5h_0 + 4h_1 + 3h_2 + 2h_3 + h_4 \leq 48. \tag{5}$$

Proof. Let $\ell_i = \ell_i(p, P, M)$ be the number of ways we can match the point p as a left endpoint of degree i. First, we claim that $\ell_0 \in \{0, 1\}$.

To show this, form the *vertical decomposition* of M by drawing a vertical segment up and down from each (matched or isolated) point of $P \setminus \{p\}$ and extend these segments until they meet an edge of M or else, all the way to infinity; see Figure 5 for an illustration of such a decomposition. We call these vertical segments *walls* in order to distinguish them from the segments in the matching.

We obtain a decomposition of the plane into vertical trapezoids. Let τ be the trapezoid containing p (assuming general position, p lies in the interior of τ). See Figure 5.

We move from τ to the right through vertical walls to adjacent trapezoids until we reach a vertical wall that is determined by a point v that is either a left endpoint or an isolated point (if at all—we may make our way to infinity when p cannot be matched as a left endpoint to any point, in which case $\ell_i = 0$ for all i).

Note that up to that point there was always a unique choice for the next trapezoid to enter. Every crossing-free segment with p as its left endpoint will have to go through all of these trapezoids. It connects either to v (which *can* happen only if v is isolated) or crosses the vertical wall up or down from v. The former case yields a segment that gives p degree 0. In the latter case, v will contribute 1 to the degree of p. So pv, if an option, is the only possible segment that lets p have degree 0 as a left endpoint. (pv will not be an option when it crosses some segment or when v is a left endpoint.)

We will return to this setup when we consider degrees ≥ 1, in which case v acts as a *bifurcation point*. Before doing so, we first introduce a function f. It maps every nonnegative real vector $(\lambda_0, \lambda_1, \ldots, \lambda_k)$ of arbitrary length $k+1 \in \mathbb{N}$ to the maximum possible value[9] the expression

$$\lambda_0\ell_0 + \lambda_1\ell_1 + \cdots + \lambda_k\ell_k \tag{6}$$

[9] A priori, this value could be infinite.

can attain (for any isolated point in any matching of any finite point set *of any size*). We have already shown that $f(\lambda) \leq \lambda$ for $\lambda \in \mathbb{R}_0^+$. We claim that for all $(\lambda_0, \lambda_1, \ldots, \lambda_k) \in (\mathbb{R}_0^+)^{k+1}$, with $k \geq 1$, we have

$$f(\lambda_0, \lambda_1, \ldots, \lambda_k) \;\leq\; \max\{\lambda_0 + f(\lambda_1, \ldots, \lambda_k),\; 2f(\lambda_1, \ldots, \lambda_k)\}. \tag{7}$$

Assuming (7) has been established, we can conclude that $f(1) \leq 1$, $f(2,1) \leq 3$, $f(3,2,1) \leq 6$, and $f(4,3,2,1) \leq 12$; that is,[10] $4\ell_0 + 3\ell_1 + 2\ell_2 + \ell_3 \leq 12$ and the first inequality of the lemma follows, since the same bound clearly holds for the case when p is a right endpoint. The second inequality follows similarly from $f(5,4,3,2,1) \leq 24$.

So it remains to prove (7). Consider a constellation with a point p that realizes the value of $f(\lambda_0, \lambda_1, \ldots, \lambda_k)$. We return to the setup considered above, where we have traced a unique sequence of trapezoids from p to the right, till we encountered the first bifurcation point v (if v does not exist, then all ℓ_i vanish).

Case 1. *v is isolated.* We know that $\lambda_0\ell_0 \leq \lambda_0$. If we remove v from the point set, then every possible crossing-free segment emanating from p to its right has its degree decreased by 1. Therefore, $\lambda_1\ell_1 + \cdots + \lambda_k\ell_k \;\leq\; f(\lambda_1, \ldots, \lambda_k)$, so the expression (6) cannot exceed $\lambda_0 + f(\lambda_1, \ldots, \lambda_k)$ in this case.

Case 2. *v is a matched left endpoint.* Then $\lambda_0\ell_0 = 0$ (that is, we cannot connect p to v). Possible crossing-free segments with p as a left endpoint are discriminated according to whether they pass above or below v. We first concentrate on the segments that pass above v; we call them *relevant segments* (emanating from p). Let ℓ_i' be the number of relevant segments that give p degree i. We carefully remove isolated points from $P \setminus \{p\}$ and segments with their endpoints from the matching M (eventually also the segment of which v is a left endpoint), so that in the end all relevant segments are still available and each one, if inserted, makes the degree of p exactly 1 unit smaller than its original value (this deletion process may create new possibilities for segments from p). That will show $\lambda_1\ell_1' + \cdots + \lambda_k\ell_k' \;\leq\; f(\lambda_1, \ldots, \lambda_k)$. The same will apply to segments that pass below v, using a symmetric argument, which gives the bound of $2f(\lambda_1, \ldots, \lambda_k)$ for (6) in this second case.

The removal process is performed as follows. We define a relation $\prec$ on the set whose elements are the edges of M and the singleton sets formed by the isolated points of $P \setminus \{p\}$: $a \prec b$ if a point $a' \in a$ is vertically visible from a point $b' \in b$, with a' below b'. As is well known (cf. [19, Lemma 11.4]), $\prec$ is acyclic. Let $\prec^+$ denote the transitive closure of $\prec$, and let $\prec^*$ denote the transitive reflexive closure of $\prec$.

Let e be the segment with v as its left endpoint, and consider a minimal element a with $a \prec^+ e$. Such an element exists, unless e itself is a minimal element with respect to $\prec$.

a is a singleton: Thus it consists of an isolated point; with abuse of notation we also denote by a the isolated point itself. a cannot be a point to which p can connect with a relevant edge. Indeed, if this were the case, we add that edge $e' = pa$ and modify $\prec$ to include e' too; more precisely, any pair in $\prec$ that involves a is replaced by a corresponding pair involving e', and new pairs involving e' are added (clearly, the relation remains acyclic and all pairs related under $\prec^+$ continue to be so related after e' is included and replaces a). See Figure 6(a). We have $e \prec e'$ (since, by assumption, the left endpoint v of e is vertically visible below e') and $e' \prec^+ e$ (since

[10] Note that $\ell_i \leq 2^i$ for each $i \geq 0$ (which can be shown to be tight); this only yields a bound of 26 for the linear combination in question. Moreover, $\sum_{i=0}^{k} \ell_i \leq 2^k$ (which again is tight), but this only improves the bound to 15, still short of what we need.

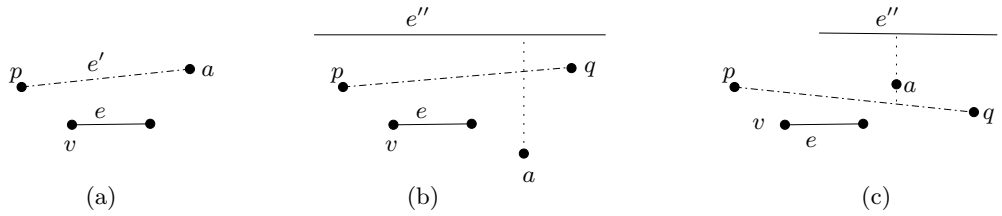


FIG. 6. (a) *The point a cannot be connected to p via a relevant edge.* (b), (c) *a cannot contribute from below (in* (b)*) or from above (in* (c)*) to the degree of p when a relevant edge pq is inserted.*

the right endpoint a of e' satisfies $a \prec^+ e$)—a contradiction. With a similar reasoning we can rule out the possibility that a contributes to the degree of p when matched via a relevant edge pq. Indeed, if this is the case, let e'' be the segment directly above a, which is the first link in the chain that gives $a \prec^+ e$; i.e., $a \prec e'' \prec^* e$ (e'' must exist since $a \prec^+ e$). After adding pq with a contributing to its degree, we have either $a \prec pq$ and $pq \prec e''$ (see Figure 6(b)) or we have $pq \prec a$ (see Figure 6(c)). In the former case, we have $a \prec pq \prec e'' \prec^* e \prec pq$, contradicting the acyclicity of $\prec$. In the latter case, we have $pq \prec a \prec^+ e \prec pq$, again a contradiction. So if we remove a, then all relevant edges from p remain in the game and the degree of each of them (i.e., the degree of p that the edge induces when inserted) does not change.

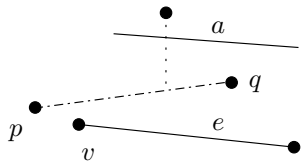


FIG. 7. *Edge a cannot obstruct a point from contributing from above to the degree of p when a relevant edge pq is inserted.*

a is an edge: It cannot obstruct any isolated point or left endpoint below it from contributing to the degree of a relevant edge pq above v (because a is minimal with respect to $\prec$). If a obstructs a contribution to a relevant edge pq from above, then we add pq; thus $pq \prec a$, which, together with $a \prec^+ e$ and $e \prec pq$, contradicts the acyclicity of $\prec$. See Figure 7. Again, we can remove a without any changes to relevant possible edges from p.

We keep successively removing elements until e is minimal with respect to $\prec$. Note that so far all the relevant edges from p are still possible, and the degree of p that any of them induces when inserted has not changed.

Now we remove e with its endpoints. This cannot clear the way for any new contribution to the degree of a relevant edge. In fact, any such degree decreases by exactly 1 because v disappears. The claim is shown, and the proof of the lemma is completed. ∎

Deriving a recurrence.

LEMMA 2.3. *Let $n, m \in \mathbb{N}_0$, such that $m \le \frac{n}{2}$ and $s := n - 2m$. For every set P of n points, we have*

$$\mathsf{ma}_m(P) \le \begin{cases} \frac{12(s+2)}{n-3s}\,\mathsf{ma}_{m-1}(P) & \text{if } s < \frac{n}{3}, \\ \frac{16(s+2)}{n-7s/3}\,\mathsf{ma}_{m-1}(P) & \text{if } s < \frac{3n}{7}. \end{cases}$$

Let us note right away that the first inequality supersedes the second for $s < \frac{n}{5}$ (i.e., $m > \frac{2n}{5}$), while the second one is superior for $s > \frac{n}{5}$.

Proof. Fix the set P, and let $\mathcal{X}$ and $\mathcal{Y}$ be the sets of all crossing-free m-matchings and $(m-1)$-matchings, respectively, in P.

Let us concentrate on the first inequality. We define an edge-labeled bipartite graph $\mathcal{G}$ on $\mathcal{X} \dot\cup \mathcal{Y}$ as follows: Given an m-matching M, if p is an endpoint of a segment $e \in M$ and $d(p) \le 3$, then we connect $M \in \mathcal{X}$ to the $(m-1)$-matching $M \setminus \{e\} \in \mathcal{Y}$ with an edge labeled $(p, d(p))$; $d(p)$ is the *degree label* of the edge. Note that M and $M \setminus \{e\}$ can be connected by two (differently labeled) edges if both endpoints of e have degree at most 3.

For $0 \le i \le 3$, let x_i denote the number of edges in $\mathcal{G}$ with degree label i. We have

$$(2n-6s)\ \underbrace{|\mathcal{X}|}_{\mathsf{ma}_m(P)}\ \le\ 4x_0+3x_1+2x_2+x_3\ \le\ 24(s+2)\ \underbrace{|\mathcal{Y}|}_{\mathsf{ma}_{m-1}(P)}\ .$$

The first inequality is a consequence of inequality (1) of Lemma 2.1. The second inequality is implied by inequality (4) in Lemma 2.2, as follows. For a fixed $(m-1)$-matching M' in P, consider an edge of $\mathcal{G}$ that is incident to M' and is labeled by (p, i) (if there is such an edge). Then p must be one of the $s+2$ isolated points of P (with respect to M'), and there is a way to connect p to another isolated point in a crossing-free manner so that p has degree i in the new matching. Hence, the contribution by p and M' to the sum $4x_0+3x_1+2x_2+x_3$ is at most 24 by inequality (4) in Lemma 2.2, and the right inequality follows. The combination of both inequalities yields the second inequality in (8).

By considering endpoints up to degree 4 (instead of 3), we get the second inequality in an analogous fashion (with the help of inequality (2) in Lemma 2.1 and inequality (5) in Lemma 2.2). ∎

For $m, n \in \mathbb{N}_0$, let $\mathsf{ma}_m(n)$ denote the maximum number of crossing-free m-matchings a set of n points can have.

LEMMA 2.4. *Let $s, m, n \in \mathbb{N}_0$, with $n = 2m + s$. We have*

$$\mathsf{ma}_0(0) = 1,$$
$$\mathsf{ma}_m(n) \le \begin{cases} \frac{n}{s}\,\mathsf{ma}_m(n-1) & \text{for } s \ge 1, \\ \frac{12(s+2)}{n-3s}\,\mathsf{ma}_{m-1}(n) & \text{for } s < \frac{n}{3}, \\ \frac{16(s+2)}{n-7s/3}\,\mathsf{ma}_{m-1}(n) & \text{for } s < \frac{3n}{7}. \end{cases} \tag{8}$$

Proof. $\mathsf{ma}_0(0) = 1$ is trivial.

The first of the three inequalities in (8) is implied by

$$s \cdot \mathsf{ma}_m(P) = \sum_{p \in P} \mathsf{ma}_m(P \setminus \{p\}) \le n \cdot \mathsf{ma}_m(n-1)$$

for any set P of n points. The second and third inequalities follow from Lemma 2.3. ∎

3. Solving a recurrence. We derive an upper bound for a function

$$G \equiv G_{\lambda,\mu} :\ \mathbb{N}_0^2 \to \mathbb{R}^+$$

for a pair of parameters $\lambda, \mu \in \mathbb{R}^+$, $\mu \geq 1$, which satisfies

$$G(0,0) = 1,$$
$$G(m,n) \leq \begin{cases} \frac{n}{s}\, G(m, n-1) & \text{for } s \geq 1, \\ \frac{\lambda(s+2)}{n-\mu s}\, G(m-1, n) & \text{for } s < \frac{n}{\mu}, \end{cases} \tag{9}$$

with the convention $s := n - 2m$.

The recurrence in (8) implies that an upper bound on $G_{12,3}(m,n)$ also serves as an upper bound for $\mathsf{ma}_m(n)$, and the same holds for $G_{16,7/3}(m,n)$. We will see how to best combine the two parameter pairs to obtain even better bounds for $\mathsf{ma}_m(n)$. Later, we will encounter other instances of this recurrence, with other values of λ and μ.

We normalize by dividing by $\lambda^m \mu^{n-m}$. Then (9) becomes

$$\frac{G(m,n)}{\lambda^m \mu^{n-m}} \leq \begin{cases} \frac{n}{\mu s}, \frac{G(m,n-1)}{\lambda^m \mu^{n-1-m}} & \text{for } s \geq 1, \\ \frac{\mu(s+2)}{n-\mu s}\, \frac{G(m-1,n)}{\lambda^{m-1} \mu^{n-m+1}} & \text{for } s < \frac{n}{\mu}. \end{cases}$$

We set $H(m,n) = H_\mu(m,n) := \frac{G(m,n)}{\lambda^m \mu^{n-m}}$. Therefore, still with the convention $s := n - 2m$ and the assumption $\mu \geq 1$, we have

$$H(0,0) = 1,$$
$$H(m,n) \leq \begin{cases} \frac{n}{\mu s}\, H(m, n-1) & \text{for } s \geq 1, \\ \frac{\mu(s+2)}{n-\mu s}\, H(m-1, n) & \text{for } s < \frac{n}{\mu}. \end{cases} \tag{10}$$

We note that this recurrence depends only on μ.

LEMMA 3.1. *Let $m, n \in \mathbb{N}_0$, with $m \leq \frac{n}{2}$. Then $H(m,n) \leq \binom{n}{m}$.*

Proof. $H(0,0) = 1 \leq \binom{0}{0}$ forms the basis of a proof by induction on n and m. For all $n \in \mathbb{N}_0$, $H(0,n) \leq \mu^{-n} \leq 1 = \binom{n}{0}$ follows, since $\mu \geq 1$.

Let $1 \leq m \leq \frac{n}{2}$. If $m \leq n - \mu s$, then $s \leq \frac{n-m}{\mu} < \frac{n}{\mu}$. Hence, the second inequality in (10) can be applied, after which the first inequality can be applied. Hence,

$$\begin{aligned} H(m,n) &\leq \frac{\mu(s+2)}{n-\mu s}\, H(m-1,n) \\ &\leq \frac{\mu(s+2)}{n-\mu s}\, \frac{n}{\mu(s+2)}\, H(m-1,n-1) \\ &\leq \frac{n}{m} \binom{n-1}{m-1} = \binom{n}{m}. \end{aligned}$$

Otherwise, $m > n - \mu s$ holds, which ensures that $\mu s > n - m \geq 0$, i.e., $s \geq 1$. We can therefore employ the first inequality of (10) and obtain

$$H(m,n) \leq \frac{n}{\mu s}\, H(m,n-1) < \frac{n}{n-m} \binom{n-1}{m} = \binom{n}{m}. \qquad \square$$

By expanding along the first inequality for a while before employing Lemma 3.1,

we get

$$\begin{aligned} H(m,n) &\le \frac{n}{\mu s}\cdots\frac{n-k+1}{\mu(s-k+1)}\,H(m,n-k) \\ &\le \frac{1}{\mu^k}\left(\prod_{i=0}^{k-1}\frac{n-i}{s-i}\right)\binom{n-k}{m} \\ &= \frac{1}{\mu^k}\frac{\binom{n}{k}}{\binom{s}{k}}\binom{n-k}{m} \qquad (11) \\ &= \frac{1}{\mu^k}\frac{\binom{2m}{m}}{\binom{n-m-k}{m}}\binom{n}{2m} \quad \text{for } \mathbb{N}_0 \ni k \le s. \qquad (12) \end{aligned}$$

When we stop this unwinding of the recurrence, we could have alternatively proceeded one more step, and upper bound $H(m,n-k)$ by $\frac{n-k}{\mu(s-k)}\binom{n-k-1}{m}$, provided that $k<s$. As long as this expression is smaller than $\binom{n-k}{m}$, we should indeed have expanded further. That is, we expand as long as

$$\begin{aligned} &\frac{n-k}{\mu(s-k)}\binom{n-k-1}{m} < \binom{n-k}{m} \\ \Leftrightarrow\ &\frac{n-k}{\mu(s-k)}(n-k-m) < n-k \\ \Leftrightarrow\ &k < \frac{\mu s+m-n}{\mu-1} = n - m\left(\frac{2\mu-1}{\mu-1}\right) = n-\frac{m}{\rho} \end{aligned}$$

for $\rho := \frac{\mu-1}{2\mu-1}$. In other words, the best choice of k in (11) is

$$k = \left\lceil n - \frac{m}{\rho}\right\rceil = n - \left\lfloor \frac{m}{\rho}\right\rfloor. \qquad (13)$$

In fact, if this suggested value of k is negative (or if $\rho = 0$), we should not expand at all. Instead, we can try to expand along the second inequality of (10), to get (note that here reducing m by 1 increases s by 2)

$$\begin{aligned} H(m,n) &\le \frac{\mu(s+2)}{n-\mu s}\cdots\frac{\mu(s+2+2(k-1))}{n-\mu(s+2(k-1))}H(m-k,n) \\ &\le \left(\prod_{i=0}^{k-1}\frac{\frac{s}{2}+1+i}{\frac{n}{2\mu}-\frac{s}{2}-i}\right)\binom{n}{m-k} \\ &= \frac{\binom{\frac{s}{2}+k}{k}}{\binom{\frac{n}{2\mu}-\frac{s}{2}}{k}}\binom{n}{m-k} \qquad (14) \end{aligned}$$

for $\mathbb{N}_0 \ni k < \frac{n}{2\mu}-\frac{s}{2}+1 = m - \frac{\mu-1}{2\mu}n + 1$; here we employ the usual generalization of binomial coefficients $\binom{a}{k}$ to $a\in\mathbb{R}$, namely, $\binom{a}{k} := \frac{a(a-1)\cdots(a-k+1)}{k!}$.

Rather than optimizing the value of k at which we stop the unwinding of the second recurrence inequality of (10), we approximate it by

$$k = \left\lceil m - \frac{\mu-1}{2\mu-1}\,n\right\rceil = m - \lfloor \rho n\rfloor \qquad (15)$$

and note that it lies in the allowed range, provided that it is positive. (With some tedious calculations, one can show that the optimal stopping value is $k = m - \lfloor \rho(n+1) \rfloor$, which is either equal to the k in (15) or is smaller than it by 1.)

When $\frac{m}{n} = \rho$, both values suggested for k in (13) and (15) are 0, which indicates that we have to content ourselves with the bound $\binom{n}{m}$ from Lemma 3.1. Otherwise, it is clear which way to expand, since

$$\frac{m}{n} < \rho \;\Rightarrow\; n - \left\lfloor \tfrac{m}{\rho} \right\rfloor \geq 0,$$
$$\frac{m}{n} > \rho \;\Rightarrow\; m - \lfloor \rho n \rfloor \geq 0.$$

We are now ready for an improved bound. For that we substitute k in (11) according to (13), and in (14) according to (15).

LEMMA 3.2. *Let $m, n \in \mathbb{N}_0$, where $2m \leq n$, and set $\rho := \frac{\mu-1}{2\mu-1}$. If $\frac{m}{n} \leq \rho$, then*

$$H_\mu(m,n) \leq \frac{1}{\mu^{n-\lfloor m/\rho \rfloor}} \frac{\binom{n}{n-\lfloor m/\rho \rfloor}}{\binom{n-2m}{n-\lfloor m/\rho \rfloor}} \binom{\lfloor m/\rho \rfloor}{m},$$

and for $\frac{m}{n} > \rho$, we have

$$H_\mu(m,n) \leq \frac{\binom{\frac{n}{2} - \lfloor \rho n \rfloor}{m - \lfloor \rho n \rfloor}}{\binom{m - \frac{n}{2}(1-\frac{1}{\mu})}{m - \lfloor \rho n \rfloor}} \binom{n}{\lfloor \rho n \rfloor}.$$

Thus, $G_{\lambda,\mu}(m,n) \leq \overline{G}_{\lambda,\mu}(m,n)$ with

$$\overline{G}_{\lambda,\mu}(m,n) := \begin{cases} \lambda^m \mu^{\lfloor m/\rho \rfloor - m} \dfrac{\binom{n}{n-\lfloor m/\rho \rfloor}}{\binom{n-2m}{n-\lfloor m/\rho \rfloor}} \binom{\lfloor m/\rho \rfloor}{m} & \text{for } \frac{m}{n} \leq \rho, \\ \lambda^m \mu^{n-m} \dfrac{\binom{\frac{n}{2} - \lfloor \rho n \rfloor}{m - \lfloor \rho n \rfloor}}{\binom{m - \frac{n}{2}(1-\frac{1}{\mu})}{m - \lfloor \rho n \rfloor}} \binom{n}{\lfloor \rho n \rfloor} & \text{for } \frac{m}{n} > \rho. \end{cases}$$

Next we work out a number of properties of the upper bound $\overline{G}_{\lambda,\mu}$.

Estimates up to a polynomial factor. In the following derivations, we sometimes use "$\approx_n$" to denote equality up to a polynomial factor in n.

We will frequently use the following estimate (implied by Stirling's formula; cf. [29, Chapter 10, Corollary 9]):

$$\binom{\alpha n}{\lceil \beta n \rceil} \approx_n \binom{\alpha n}{\lfloor \beta n \rfloor} \approx_n \left(\frac{\alpha^\alpha}{\beta^\beta (\alpha-\beta)^{\alpha-\beta}} \right)^n \qquad \text{for } \alpha, \beta \in \mathbb{R},\ \alpha \geq \beta \geq 0.$$

Big m. We note that for $\frac{m-1}{n} \geq \rho$

$$\overline{G}_{\lambda,\mu}(m,n) = \frac{\lambda(s+2)}{n - \mu s}\, \overline{G}_{\lambda,\mu}(m-1,n) \quad \text{with } s := n - 2m.$$

Since $\frac{\lambda(s+2)}{n-\mu s} < 1 \Leftrightarrow s < \frac{n-2\lambda}{\lambda+\mu} \Leftrightarrow m > \frac{(\lambda+\mu-1)n+2\lambda}{2(\lambda+\mu)}$, the function $\overline{G}_{\lambda,\mu}(m,n)$ maximizes for integers m in the range $\rho n \leq m \leq \frac{n}{2}$ at

$$m^* := \left\lfloor \frac{(\lambda+\mu-1)n + 2\lambda}{2(\lambda+\mu)} \right\rfloor = \left\lfloor \frac{n}{2} - \frac{n-2\lambda}{2(\lambda+\mu)} \right\rfloor \tag{16}$$

unless this value is not in the provided range. However, $m^* \leq \frac{n}{2}$ unless n is very small ($n < 2\lambda$). And $m^* \geq \rho n$ unless $\lambda < \mu - 1$.

Small m. With the identity indicated in (12) we have, for $\frac{m}{n} \le \rho$, that $\overline{G}$ can also be written as

$$\overline{G}_{\lambda,\mu}(m,n) \;=\; \lambda^m \mu^{\lfloor m/\rho \rfloor - m} \frac{\binom{2m}{m}}{\binom{\lfloor m/\rho \rfloor - m}{m}} \binom{n}{2m} \;\approx_m\; (4\lambda(\mu-1))^m \binom{n}{2m}. \tag{17}$$

This bound peaks (up to an additive constant) at

$$m^{**} := \left\lfloor \frac{\sqrt{\lambda(\mu-1)}}{1+2\sqrt{\lambda(\mu-1)}}\, n \right\rfloor.$$

We observe that $m^{**} \le \rho n$ for $\lambda \le \mu - 1$.

We can summarize that the function $\overline{G}_{\lambda,\mu}(m,n)$ attains its maximum—up to a poly(n)-factor—over m at

$$m = \begin{cases} m^{**} & \text{if } \lambda \le \mu - 1, \\ m^{*} & \text{otherwise.} \end{cases} \tag{18}$$

In all applications in this paper we have $\lambda > \mu - 1$; thus the peak occurs at m^*.

4. Bounds for matchings.

4.1. Perfect matchings. For perfect matchings we consider the case where n is even, $m = \frac{n}{2}$, and $s = 0$. We note that in this case $m/n = 1/2 > \rho$ for any value of μ. Hence, the second bound of Lemma 3.2 applies. We first calculate $\frac{n}{2} - \frac{n}{2}(1-\frac{1}{\mu}) = \frac{1}{2\mu}\, n$ and $\frac{n}{2} - \lfloor \rho n \rfloor = \lceil \frac{n}{2} - \frac{\mu-1}{2\mu-1}\, n \rceil = \lceil \frac{1}{2(2\mu-1)}\, n \rceil$. Hence,

$$\begin{aligned}
\overline{G}_{\lambda,\mu}\left(\frac{n}{2}, n\right) \;&=\; (\lambda\mu)^{n/2} \binom{\frac{1}{2\mu}\, n}{\left\lceil \frac{1}{2(2\mu-1)}\, n \right\rceil}^{-1} \binom{n}{\left\lfloor \frac{\mu-1}{2\mu-1}\, n \right\rfloor} \\
&\approx_n\; (\lambda\mu)^{n/2} \left(\frac{\left(\frac{1}{2(2\mu-1)}\right)^{\frac{1}{2(2\mu-1)}} \left(\frac{\mu-1}{2\mu(2\mu-1)}\right)^{\frac{\mu-1}{2\mu(2\mu-1)}}}{\left(\frac{1}{2\mu}\right)^{\frac{1}{2\mu}} \left(\frac{\mu-1}{2\mu-1}\right)^{\frac{\mu-1}{2\mu-1}} \left(\frac{\mu}{2\mu-1}\right)^{\frac{\mu}{2\mu-1}}} \right)^n \\
&=\; (\lambda\mu)^{n/2} \left(\mu^{\frac{1}{2(2\mu-1)} - \frac{\mu}{2\mu-1}} (\mu-1)^{\frac{\mu-1}{2\mu(2\mu-1)} - \frac{\mu-1}{2\mu-1}} (2\mu-1)^{-\frac{1}{2\mu}+1} \right)^n \\
&=\; (\lambda\mu)^{n/2} \left((\mu-1)^{-\frac{\mu-1}{2\mu}} \mu^{-\frac{1}{2}} (2\mu-1)^{\frac{2\mu-1}{2\mu}} \right)^n \\
&=\; \left(\lambda^{\frac{1}{2}} (\mu-1)^{-\frac{\mu-1}{2\mu}} (2\mu-1)^{\frac{2\mu-1}{2\mu}} \right)^n.
\end{aligned}$$

Substituting $(\lambda,\mu) = (12,3)$ and $(16, \frac{7}{3})$, as suggested by Lemma 2.4, we obtain the following upper bounds for the number of crossing-free perfect matchings:

$$\begin{aligned}
\overline{G}_{12,3}\left(\frac{n}{2}, n\right) &\approx_n \left(2^{\frac{2}{3}} \cdot 3^{\frac{1}{2}} \cdot 5^{\frac{5}{6}}\right)^n = O(10.5129^n) \quad \text{and} \\
\overline{G}_{16,\frac{7}{3}}\left(\frac{n}{2}, n\right) &\approx_n \left(2^{\frac{10}{7}} \cdot 3^{-\frac{1}{2}} \cdot 11^{\frac{11}{14}}\right)^n = O(10.2264^n).
\end{aligned}$$

While the second bound is obviously superior, we remember that the recurrence with $(\lambda,\mu) = (12,3)$ is better for $m > \frac{2n}{5}$ (or $s < \frac{n}{5}$). This observation leads to the

following better bound for P a set of n points and for $k = \lfloor \frac{n}{2} - \frac{2n}{5} \rfloor = \lfloor \frac{n}{10} \rfloor$, where we expand as in the first inequality of Lemma 2.3:

$$\begin{aligned}\mathsf{pm}(P) \;&\le\; \left(\prod_{i=0}^{k-1} \frac{12(2i+2)}{n-6i}\right) \mathsf{ma}_{n/2-k}(P) \le 4^k \binom{\frac{n}{6}}{k}^{-1} \overline{G}_{16,7/3}(n/2-k,n) \\ &\approx_n \left(2^{20/21}\, 3^{-2/7}\, 5^{1/21}\, 11^{11/14}\right)^n = O(10.0438^n).\end{aligned}$$

Perfect versus all matchings. Recall from Lemma 2.3 that

$$\mathsf{ma}_m(P) \le \frac{12(s+2)}{n-3s} \mathsf{ma}_{m-1}(P).$$

Note that $\frac{12(s+2)}{n-3s} < 1$ for $m > \frac{7n}{15} + \frac{4}{5}$ (and in this range the factor $\frac{12(s+2)}{n-3s}$ is smaller than the alternative offered in Lemma 2.3). That is, there are always fewer perfect matchings than there are $\lfloor \frac{7n}{15} + \frac{4}{5} \rfloor$-matchings. More specifically, for sets P with $n := |P|$ even and for $k = \frac{n}{2} - \lfloor \frac{7n}{15} + \frac{4}{5} \rfloor = \lceil \frac{n}{30} - \frac{4}{5} \rceil$, we have

$$\begin{aligned}\mathsf{pm}(P) = \mathsf{ma}_{n/2}(P) \;&\le\; \prod_{i=0}^{k-1} \frac{12(2i+2)}{n-6i}\; \mathsf{ma}_{n/2-k}(P) \\ &=\; \left(\frac{12\cdot 2}{6}\right)^k \binom{\frac{n}{6}}{k}^{-1} \mathsf{ma}_{n/2-k}(P) \\ &\approx_n\; 4^{n/30} \left(\left(\frac{1}{5}\right)^{1/5} \left(\frac{4}{5}\right)^{4/5}\right)^{n/6} \mathsf{ma}_{\lfloor 7n/15+4/5 \rfloor}(P) \\ &=\; \left(2^{1/3}\, 5^{-1/6}\right)^n \mathsf{ma}_{\lfloor 7n/15+4/5 \rfloor}(P)\,.\end{aligned}$$

This implies that $\mathsf{pm}(P) \le \left(2^{1/3}\, 5^{-1/6}\right)^n \mathsf{ma}(P)\, \mathrm{poly}(n) = O(0.9635^n)\, \mathsf{ma}(P)$. In every point set there are exponentially (in the size of the set) more crossing-free matchings than there are crossing-free perfect matchings.

4.2. All matchings. Our considerations in the derivation of the bound for perfect matchings imply the following upper bound for matchings with m segments:

$$\mathsf{ma}_m(P) \le \begin{cases} \overline{G}_{16,7/3}(m,n), & m \le \frac{2n}{5}, \\ \overline{G}_{12,3}(m,n)\; \frac{\overline{G}_{16,7/3}(\frac{2n}{5},n)}{\overline{G}_{12,3}(\frac{2n}{5},n)}, & \text{otherwise.} \end{cases} \tag{19}$$

To determine where the expression (19) maximizes, we note that $\overline{G}_{16,7/3}$ does not peak in its "small m"-range ($m \le \frac{4}{11}$) since $16 > \frac{7}{3} - 1$ (recall (18)). In the "big m"-range, it peaks at roughly $\frac{26n}{55}$ (see (16)), which exceeds $\frac{2}{5}$. Therefore, the maximum occurs when $\overline{G}_{12,3}$ comes into play, which peaks at roughly $\frac{7n}{15}$. For that value the upper bound evaluates to $\approx_n (2^{13/21}\, 3^{-2/7}\, 5^{3/14}\, 11^{11/14})^n = O(10.4244^n)$.

We summarize in the following main theorem.

THEOREM 4.1. *Let P be a set of n points in the plane. Then the following hold:*

(1) $\mathsf{pm}(P) \le \left(2^{20/21}\, 3^{-2/7}\, 5^{1/21}\, 11^{11/14}\right)^n \mathrm{poly}(n) = O(10.0438^n)$.

(2) $\mathsf{pm}(P) \le \left(2^{1/3}\, 5^{-1/6}\right)^n \mathsf{ma}(P)\, \mathrm{poly}(n) = O(0.9635^n)\, \mathsf{ma}(P)$.

(3) $\mathsf{ma}(P) \le \left(2^{13/21}\, 3^{-2/7}\, 5^{3/14}\, 11^{11/14}\right)^n \mathrm{poly}(n) = O(10.4244^n)$.

We note, by the way, that the first inequality in the theorem is a direct consequence of the other two inequalities.

4.3. Random point sets. Let P be any set of $N \in \mathbb{N}$ points in the plane, no three on a line, and let $r \in \mathbb{N}$ with $r \leq N$. If R is a subset of P chosen uniformly at random from $\binom{P}{r}$, then for $\lambda = 16$, $\mu = \frac{7}{3}$, and provided that $m \leq \frac{\mu-1}{2\mu-1}N = \frac{4}{11}N$, and that $r \geq 2m$, we have, using (17),[11]

$$\begin{aligned}
\mathbf{E}[\mathsf{ma}_m(R)] &= \left(\sum_{R\in\binom{P}{r}} \mathsf{ma}_m(R)\right) \Big/ \binom{N}{r} = \mathsf{ma}_m(P)\binom{N-2m}{r-2m}\Big/\binom{N}{r} \\
&\leq (4\lambda(\mu-1))^m \binom{N}{2m}\left(\binom{N-2m}{r-2m}\Big/\binom{N}{r}\right)\mathrm{poly}(m) \\
&\approx_m (4\lambda(\mu-1))^m \binom{r}{2m} = \left(2^8\,3^{-1}\right)^m \binom{r}{2m}.
\end{aligned}$$

We see that if we sample r points from a large enough set, then the expected number of crossing-free matchings observes for all m the upper bound derived for the range of small m.

Suppose now that, for n even, we sample n i.i.d. points from an arbitrary distribution, for which we require only that two sampled points coincide with probability 0. Then we can first sample a set P of $N > \frac{11}{8}n$ points and then choose a subset of size n uniformly at random from the family of all subsets of this size. We obtain a set R of n i.i.d. points from the given distribution. If P is in general position, by the argument above the expected number of perfect crossing-free matchings is at most $\approx_n (2^8\,3^{-1})^{n/2}$. If P exhibits collinearities, we perform a small perturbation yielding a set $\tilde{P}$ and the subset $\tilde{R}$. Now the bound applies to $\tilde{R}$ and also to R since a sufficiently small perturbation cannot decrease the number of crossing-free perfect matchings.

THEOREM 4.2. *For any distribution in the plane for which two sampled points coincide with probability* 0*, the expected number of crossing-free perfect matchings of* n *i.i.d. points is at most*

$$\left(2^4\,3^{-1/2}\right)^n \mathrm{poly}(n) = O(9.2377^n).$$

We next consider several variants of crossing-free *bipartite* matchings for which better upper bounds can be derived.

4.4. Red-blue perfect matchings. We assume that the given set P of n points is the disjoint union $R \dot\cup B$ of two subsets, and that each edge in the matching has to connect a point of R with a point of B. We refer to the points of R as red points and to those of B as blue.

We repeat the preceding analysis but modify the definition of the degree $d(p)$ of a point: If p is a matched point in R, say the left endpoint of its edge e, then $d(p)$ is equal to the number of left endpoints plus the number of *blue* isolated points that are vertically visible from (the relative interior of) e. A symmetric definition holds for right endpoints and for points $p \in B$. (Intuitively, a blue isolated point q has to contribute only to the degrees of red points because, when we insert an edge emanating from a blue point p, it cannot connect to q, and it does not matter whether

[11] There is a small subtlety in that the second identity in the derivation relies on the fact that P is in general position. For that consider 3 points on a line.

it passes above or below q; that is, q does not cause any bifurcation in the ways in which p can be connected.)

In this case we have

$$\sum_{p\in P} d(p) \le 4m + 2s$$

because each isolated point contributes to the degree of only 2 matched points. This changes the bounds in Lemma 2.1 to

$$2n \le 4v_0 + 3v_1 + 2v_2 + v_3 + 4s, \quad \text{and}$$
$$3n \le 5v_0 + 4v_1 + 3v_2 + 2v_3 + v_4 + 5s.$$

The rest of the analysis continues verbatim, except that now the recurrence (8) involves the factors $\frac{12(s+2)}{n-2s}$ and $\frac{16(s+2)}{n-5s/3}$ or, in other words, $(\lambda,\mu) = (12,2)$ (with $\rho = 1/3$) and $(16,\frac{5}{3})$ (with $\rho = 2/7$), respectively. The first factor is superior for $s < \frac{n}{3}$, i.e., $m > \frac{n}{3}$.

We thus obtain, with $k = \lfloor \frac{n}{6} \rfloor$, a bound of

$$\left(\prod_{i=0}^{k-1} \frac{12(2i+2)}{n-4i}\right) \overline{G}_{16,5/3}(n/2-k,n)$$

for the number of perfect red-blue matchings. Manipulating it, as above, yields the following theorem.

THEOREM 4.3. *Let P be a set of n points in the plane with each one colored red or blue. Then the number of red-blue perfect crossing-free matchings in P is at most*

$$\left(2^{6/5}\, 3^{-3/20}\, 7^{7/10}\right)^n \operatorname{poly}(n) = O(7.6075^n)\,.$$

4.5. Left-right perfect matchings. Here we assume that P is partitioned into two disjoint subsets L, R and consider bipartite matchings in $L \times R$ such that, for each edge of the matching, its left endpoint belongs to L and its right endpoint to R.

We modify the definition of the degrees of the points, as in the red-blue case, and have, as above,

$$\sum_{p\in P} d(p) \le 4m + 2s.$$

The analysis further improves because when we insert an edge emanating from a point $p \in L$, say, the corresponding numbers h_i must be equal to ℓ_i, since p can only be the left endpoint of the edge. A similar improvement holds for points $q \in R$. Hence, we can bound the sum $4h_0 + 3h_1 + 2h_2 + h_3$ by 12, rather than 24; similarly, we have $5h_0 + 4h_1 + 3h_2 + 2h_3 + h_4 \le 24$. That is, we have the two options $(\lambda,\mu) = (6,2)$ and $(8,\frac{5}{3})$. We thus obtain the bound

$$\left(\prod_{i=0}^{k-1} \frac{6(2i+2)}{n-4i}\right) \overline{G}_{8,5/3}(n/2-k,n) \quad \text{for } k = \lfloor \tfrac{n}{6} \rfloor,$$

which leads to the following result.

THEOREM 4.4. *Let P be a set of n points in the plane and assume that the points are classified as left endpoints or right endpoints. Then the number of left-right perfect crossing-free matchings in P that obey this classification is at most*

$$\left(2^{7/10}\, 3^{-3/20}\, 7^{7/10}\right)^n \operatorname{poly}(n) = O(5.3793^n).$$

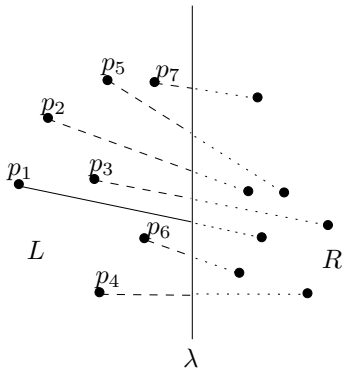


FIG. 8. *Recursively counting permutations induced on λ by left half-edges.*

4.6. Matchings across a line. Consider next the special case of crossing-free bipartite perfect matchings between two sets of $\frac{n}{2}$ points each that are separated by a line. Here we can obtain an upper bound that is smaller than the one in Theorem 4.4.

THEOREM 4.5. *Let n be an even integer. The number of crossing-free perfect bipartite matchings between two separated sets of $\frac{n}{2}$ points each in the plane is at most ${C_{n/2}}^2 < 4^n$ (recall that C_m is the mth Catalan number).*

Proof. Let L and R be the given separated sets. Without loss of generality, take the separating line λ to be the y-axis and assume that the points of L lie to the left of λ and the points of R lie to its right. Let M be a crossing-free perfect bipartite matching in $L \times R$. For each edge e of M, let e_L (resp., e_R) denote the portion of e to the left (resp., right) of λ, and refer to them as the *left half-edge* and the *right half-edge* of e, respectively. We will obtain an upper bound for the number of combinatorially different ways to draw the left half-edges of a crossing-free perfect matching in $L \times R$. The same bound will apply symmetrically to the right half-edges, and the final bound will be the square of this bound.

In more detail, we ignore R and consider collections S of $\frac{n}{2}$ pairwise disjoint segments, each connecting a point of L to some point on λ, so that each point of L is incident to exactly one segment. For each segment in S, we label its λ-endpoint by the point of L to which it is connected. The increasing y-order of the λ-endpoints of the segments thus defines a permutation of L, and our goal is to bound the number of different permutations that can be generated in this way. (In general, this is a *strict* upper bound on the quantity we seek; see below.)

We obtain this bound in the following recursive manner. Write $m := |L| = \frac{n}{2}$. Sort the points of L from left to right (we may assume that there are no ties—they can be eliminated by a slight rotation of λ), and let $p_1, p_2, \ldots, p_m$ denote the points in this order. Consider the half-edge e_1 emanating from the leftmost point p_1. Any other point p_j lies either above or below e_1. By rotating e_1 about p_1, we see that there are at most m (exactly m, if we assume general position) ways to split $\{p_2, \ldots, p_m\}$ into a subset L_1^+ of points that lie above e_1 and a complementary subset L_1^- of points that lie below e_1, where in the ith split, $|L_1^+| = i - 1$ and $|L_1^-| = m - i$. Note that, in any crossing-free perfect bipartite matching that has e_1 as a left half-edge incident to p_1, all the points of L_1^+ (resp., of L_1^-) must be incident to half-edges that terminate on λ *above* (resp., *below*) the λ-endpoint of e_1; see Figure 8.

Hence, after having fixed i, we can proceed to bound recursively and separately

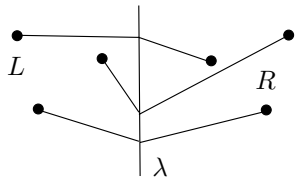


FIG. 9. *A left and a right permutation which are not compatible.*

the number of permutations induced by L_1^+ and the number of those induced by L_1^-. In other words, denoting by $\Pi(m)$ the maximum possible number of different permutations induced in this way by a set L of m points (in general position), we get the following recurrence:

$$\Pi(m) \leq \sum_{i=1}^{m} \Pi(i-1)\Pi(m-i)$$

for $m \geq 1$, where $\Pi(0) = 1$. However, this is the recurrence that (with equality) defines the Catalan numbers, so we conclude that $\Pi(m) \leq C_m$.

A (probably weak) upper bound for the number of crossing-free perfect bipartite matchings in $L \times R$ is thus ${C_m}^2$. Indeed, for any permutation π_L of L and any permutation π_R of R, there is at most one crossing-free perfect bipartite matching in $L \times R$ that induces both permutations. Namely, it is the matching that connects the jth point in π_L to the jth point in π_R for each $j = 1, \ldots, m$. See Figure 9 for an example of two such permutations that do not yield a (straight-edge) crossing-free matching.

We thus obtain the asserted upper bound ${C_m}^2 = {C_{n/2}}^2 < 4^n$. □

5. Two implications. We show how bounds for matchings can be used to infer bounds for other problems, in particular spanning cycles and crossing-free partitions.

5.1. Spanning cycles. Apart from triangulations, crossing-free spanning cycles (simple polygonizations) have received the most attention in this context (see the original question in [32], the web site [15], or the book [12, section 8.4, Problem 8]). It is easy to see the relation $\mathsf{sc}(P) \leq \mathsf{pm}(P)^2$ for sets P with an even number of points. A better bound on $\mathsf{sc}(P)$ can be obtained via left-right matchings as follows.

THEOREM 5.1. *Let P be a set of n points in the plane. Then the number of crossing-free spanning cycles satisfies*

$$\mathsf{sc}(P) \leq (2^{7/5}\, 3^{7/10}\, 7^{7/5})^n \mathrm{poly}(n) = O(86.8089^n).$$

Proof. Let P be a given set of n points. We construct a new set P' of $2n$ points by creating two copies p^+, p^- of each point $p \in P$, and by placing these copies covertically very close to the original location of p, with p^+ lying above p^-.

Let π be a cycle in P. We map π to a perfect matching in P' as follows. For each $p \in P$, let q, r be its neighbors in π. (i) If both q, r lie to the left of p, with the edge qp lying above rp, we connect p^+ to either q^+ or q^-, and connect p^- to either r^+ or r^- (the actual choices will be determined at q and r by similar rules). (ii) The same rule applies in the case where both q, r lies to the right of p. (iii) If q lies to the left of p and r lies to the right of p, then we connect p^+ to either q^+ or q^-, and connect p^- to either r^+ or r^-. It is clear that the resulting graph π^* is a crossing-free perfect

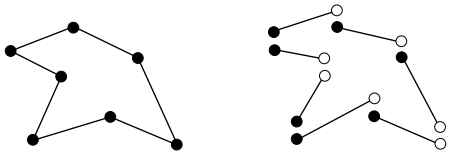


FIG. 10. *A cycle in P induces a left-right perfect matching in P'.*

matching in P', assuming general position of the points of P, if we draw each pair of points p^+, p^- sufficiently close to each other. See Figure 10 for an illustration.

We assign to each point $p \in P$ a label that depends on π. A point whose two neighbors in π lie to its left is labeled as a *right point*, a point whose two neighbors in π lie to its right is labeled as a *left point*, and a point having one neighbor in π to its right and one to its left is labeled as a *middle point*.

We assign the cycle π to the pair (π^*, λ), where π^* is the resulting perfect matching on P' and λ is the labeling of P, as just defined.

Each pair (π^*, λ) can be realized by at most one cycle π in P, by simply merging each pair p^+, p^- back into the original point p. (The resulting graph need not be a cycle; in general it is a collection of pairwise disjoint cycles.) It therefore suffices to bound the number of such pairs (π^*, λ).

A given labeling λ of P uniquely classifies each point of P' as being either a left point of an edge of the matching or a right endpoint of such an edge. Hence, the number of crossing-free perfect matchings π' on P' that respect this left-right assignment is at most $(2^{7/10}\, 3^{-3/20}\, 7^{7/10})^{2n}\mathrm{poly}(n)$. The number of labelings of P is 3^n. Hence, the number of crossing-free cycles in P is at most $(2^{7/5}\, 3^{7/10}\, 7^{7/5})^n\mathrm{poly}(n)$, as asserted. □

Clearly, it follows from the proof that the bound holds for the number of crossing-free spanning paths as well, and also for the number of cycle covers (or path covers) of P.[12]

5.2. Crossing-free partitions. We now relate crossing-free partitions of a point set P to matchings, thereby establishing an upper bound on $\mathsf{cfp}(P)$.

To this end, every crossing-free partition of P is mapped to a tuple (M, S, I^+, I^-) where (see Figure 11)

(i) M is the matching in P, whose edges connect the leftmost point to the rightmost point of each set in the partition with at least two elements (we refer to each such segment as the *spine* of its set);

(ii) S is the set of all points that form singleton sets in the partition; and

(iii) I^+ (resp., I^-) is the set of points in $P \setminus S$ that are neither the leftmost nor the rightmost in their set, and which lie *above* (resp., *below*) the spine of their set.

We observe that M is crossing-free and that the partition is uniquely determined by (M, S, I^+, I^-). Therefore, any upper bound on the number of such tuples will establish an upper bound on the number of crossing-free partitions. For every crossing-free matching M on P there are $3^{n-2|M|}$ triples (S, I^+, I^-) which form a 4-tuple with M (clearly, not all of them have to come from a crossing-free partition, so we overcount). Therefore $\sum_m 3^{n-2m}\mathsf{ma}_m(P)$ is an upper bound on the number of crossing-free partitions.

[12] A slight improvement can be obtained by noting that when a cycle has j middle points, we can derive from it 2^j distinct matchings in P', by flipping the connections to some of the pairs of P' that represent middle points.

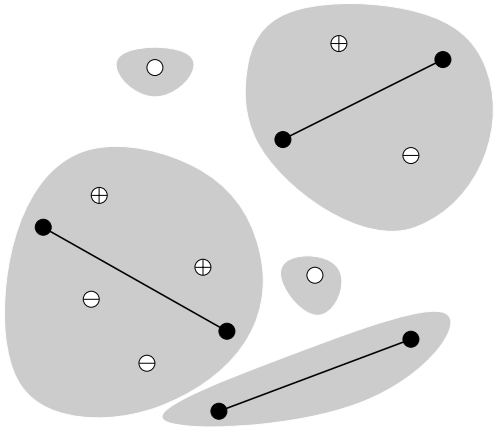


FIG. 11. *Encoding a crossing-free partition: Spines, isolated (◯), top (⊕), and bottom (⊖) points.*

Ignoring the 3^n-factor for the time being, we have to determine an upper bound on $3^{-2m}\mathsf{ma}_m(P)$, for which we employ the bound from inequality (19). We observe that $3^{-2m}\overline{G}_{\lambda,\mu}(m,n) = \overline{G}_{\lambda/9,\mu}(m,n)$, and therefore

$$3^{-2m}\mathsf{ma}_m(P) \le \begin{cases} \overline{G}_{16/9,7/3}(m,n), & m \le \frac{2n}{5}, \\ \overline{G}_{4/3,3}(m,n)\ \frac{\overline{G}_{16,7/3}(\frac{2n}{5},n)}{\overline{G}_{12,3}(\frac{2n}{5},n)}, & \text{otherwise.} \end{cases} \tag{20}$$

Since $\frac{16}{9} \ge \frac{7}{3} - 1$ (see (18)), the peak will not occur in the "small m"-range of $\overline{G}_{16/9,7/3}$. In its "big m"-range, the maximum occurs at m roughly $\frac{14n}{37}$ (see (16)) which lies in the interval $[\frac{4}{11}, \frac{2}{5}]$. Also, $G_{4/3,3}$ peaks for $m \le \frac{2n}{5}$ since $\frac{4}{3} \le 3 - 1$ (consult (18)). Therefore, the bound peaks at m roughly $\frac{14n}{37}$ with the value

$$3^n\,\overline{G}_{16/9,7/3}(\lfloor\tfrac{14n}{37}\rfloor, n) \approx_n (2^{4/7}\,3^{-1/2}\,11^{11/14}\,37^{3/14})^n\ .$$

Note that we could have estimated the number of 4-tuples by first choosing a subset Q, which is the union of S and the endpoints of M, then choosing a matching in Q, and then partitioning $P \setminus Q$ into $I^+ \cup I^-$. This leads to a bound of $\approx_n \sum_k \binom{n}{k} c^k 2^{n-k} = (c+2)^n$, where c is the constant in the bound for all matchings. This yields a bound of $O(12.43^n)$, which falls short of our bound obtained above.

THEOREM 5.2. *Let P be a set of n points in the plane. Then the number of crossing-free partitions satisfies*

$$\mathsf{cfp}(P) \le \left(2^{4/7}\,3^{-1/2}\,11^{11/14}\,37^{3/14}\right)^n \mathrm{poly}(n) = O(12.2388^n).$$

6. Lower bounds. In this section we briefly derive the lower bounds mentioned in Table 1. Most of them rely on an analysis of the so-called *double chain*, as it was first considered by García, Noy, and Tejel [21] in the context of crossing-free graphs. For matchings across a line (and left-right matchings) we use a different configuration.

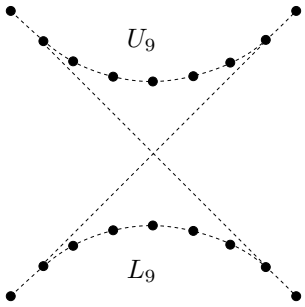


FIG. 12. *The double chain* D_{18}.

6.1. The double chain. Given $m \in \mathbb{N}$, the double chain D_{2m} consists of $n := 2m$ points; see Figure 12. There is an upper half U_m of m points on the parabola $y = \frac{x^2+1}{2}$ with their x-coordinates in $[-1, +1]$, and there is a lower half L_m of m points on the parabola $y = -\frac{x^2+1}{2}$ in the same x-range. The important property is that U_m and L_m are in convex position, and the relative interior of each segment connecting a point from U_m with a point from L_m is disjoint from the convex hulls of U_m and of L_m and thus cannot cross any segment connecting points within these sets.

García, Noy, and Tejel [21] show, among other things, that $\mathsf{sc}(D_{2m}) = \Omega(4.64^n)$ and that

$$\mathsf{pm}(D_{2m}) = \sum_{k=0}^{\lfloor m/2 \rfloor} \binom{m}{2k}^2 {C_k}^2 \approx_n 3^n. \tag{21}$$

We wish to recapitulate the argument for the latter bound. A crossing-free perfect matching with k inner edges within U_m leaves $m - 2k$ points in U_m to be matched to the same number of points in L_m. So within L_m, we also have k inner edges. If we choose the $2k$ endpoints in U_m for the inner edges ($\binom{m}{2k}$ choices), then we have C_k possibilities to connect them in a perfect crossing-free matching; the same bound applies to L_m. The remaining points from U_m and L_m allow exactly one crossing-free perfect matching from the upper set to the lower set. This gives the bound in (21). (The estimate for the sum builds on the observation that $\sum_{i=0}^{N} {a_i}^2 \approx_N \left(\sum_{i=0}^{N} a_i\right)^2$ for nonnegative real numbers a_i.)

In a similar fashion we can argue now for

$$\mathsf{ma}(D_{2m}) = \sum_{k=0}^{m} \binom{m}{k}^2 {M_k}^2 \approx_n 4^n,$$

where $M_k = \sum_i \binom{k}{2i} C_i = \Theta(k^{-3/2} 3^k)$ is the kth Motzkin number that counts the number of all matchings of k points in convex position [31].

Crossing-free partitions. Along similar lines we easily get a lower bound of

$$\mathsf{cfp}(D_{2m}) \geq \sum_{k=0}^{m} \binom{m}{k}^2 {C_k}^2 \approx_n 5^n.$$

This bound for crossing-free partitions counts only a restricted class of such partitions, namely, those composed of a matching between $m - k$ points in U_m and $m - k$ points

in L_m, together with crossing-free partitions among the remaining k points in U_m and among the remaining k points in L_m.

Let us perform an exhaustive counting of crossing-free partitions of the double chain. Here are the ingredients.

Recall first that for $N \in \mathbb{N}_0$, $i \in \mathbb{N}$, the number N can be written as an ordered sum of i nonnegative integers in $\binom{N+i-1}{i-1}$ ways, and as an ordered sum of i positive integers in $\binom{N-1}{i-1}$ ways.

Now we "prepare" the upper half U_m for a crossing-free partition as follows. We specify the number k of parts that extend to the lower half, and we also specify which k contiguous nonempty subsequences of points of U_m form the upper portions of these extended parts; we refer to these sequences as *docking places*. If the overall size of these docking places is $k+\ell$, we have to specify numbers $a_i \in \mathbb{N}_0$, $0 \le i \le k$, which are the sizes of intermediate nondocking parts, and numbers $b_i \in \mathbb{N}$, $1 \le i \le k$, which are the sizes of docking parts, so that $m = a_0+b_1+a_1+\cdots b_k+a_k$, with $\sum a_i = m-k-\ell$ (and so $\sum b_i = k+\ell$).

There are $\binom{m-k-\ell+(k+1)-1}{(k+1)-1} = \binom{m-\ell}{k}$ ways to choose the a_i's, and $\binom{k+\ell-1}{k-1}$ ways to choose the b_i's. That is, the number of configurations with k docking places (with the nondocking points already forming a crossing-free partition within U_m) is exactly

$$\sum_{\ell=0}^{m-k} \binom{m-\ell}{k}\binom{k+\ell-1}{k-1} C_{m-k-\ell}.$$

Hence, repeating the same analysis for the lower half L_m and observing that, as in the case of matchings, there is a unique way to connect the upper and lower docking places in a noncrossing manner, we obtain

$$\mathsf{cfp}(D_{2m}) = {C_m}^2 + \sum_{k=1}^{m} \left(\sum_{\ell=0}^{m-k} \binom{m-\ell}{k}\binom{k+\ell-1}{k-1} C_{m-k-\ell} \right)^2.$$

So for an estimate up to a polynomial factor in m, it remains to find k and ℓ so that $f(m,\ell,k) := \binom{m-\ell}{k}\binom{k+\ell-1}{k-1}C_{m-k-\ell}$ is large. We have

$$f(m, \lfloor 0.05m \rfloor, \lfloor 0.22m \rfloor) > 5.23^m \mathrm{poly}(m),$$

which gives $\mathsf{cfp}(D_{2m}) > (5.23^m \mathrm{poly}(m))^2 = 5.23^n \mathrm{poly}(n)$. (The coefficients 0.05 and 0.22 were chosen via a numerical experimentation.)

Red-blue matchings. It is worthwhile to notice that if we color n points in convex position, with n even, alternately red and blue along the boundary of their convex hull, then all perfect matchings on this set are compatible with this coloring. That is, we have a colored set of n points with $C_{n/2} \approx_n 2^n$ crossing-free perfect red-blue matchings. Again, we will employ the double chain for a better lower bound.

Assume m to be even, consider D_{2m}, and color the points in U_m alternately red and blue, starting with red at the leftmost point. Then color L_m alternately blue and red, starting with blue at the leftmost point. Given that coloring, we generate perfect red-blue matchings as follows:

(i) Choose some k, $0 \le k \le \frac{m}{2}$.

(ii) Select k red points in U_m ($\binom{m/2}{k}$ possibilities).

(iii) Select k blue points in L_m ($\binom{m/2}{k}$ possibilities).

(iv) Match the selected red points and their next (to the right) blue neighbors in U_m with the selected blue points and their next (to the right) red neighbors in L_m. This can be done in a *unique* crossing-free manner, which is also red-blue compatible.

(v) Match the remaining $m-2k$ points in U_m. By the way points were selected, the remaining points are still alternately red and blue and thus allow $C_{m/2-k}$ red-blue matchings, and the same holds for the lower chain L_m.

This gives

$$\sum_{k=0}^{m/2} \binom{m/2}{k}^2 {C_{m/2-k}}^2 \approx_m \sum_{k=0}^{m/2} \binom{m/2}{k}^2 \left(4^{m/2-k}\right)^2 \approx_m 5^m = \sqrt{5}^{\,n} = \Omega(2.23^n)$$

perfect crossing-free red-blue matchings as claimed in Table 1. The above procedure does not catch all possible perfect crossing-free red-blue matchings—a more accurate analysis might lead to a better bound.

Perfect matchings in random sets. Finally, let us describe a distribution in the plane such that the expected number of crossing-free perfect matchings of n i.i.d. points, for n even, is at least $3^n/\mathrm{poly}(n)$. We draw a random point p by first choosing an x uniformly at random in $[-1,+1]$ and then letting $p = (x, \frac{x^2+1}{2})$ or $p = (x, -\frac{x^2+1}{2})$, each of the two possibilities with probability $\frac{1}{2}$. A set P of n i.i.d. points from this distribution is of the form $U_k \cup L_{n-k}$ with probability $\frac{1}{2^n}\binom{n}{k}$. Therefore,

$$\mathbf{E}[\mathsf{pm}(P)] = \frac{1}{2^n}\sum_{k=0}^{n}\binom{n}{k}\mathsf{pm}(U_k \cup L_{n-k}) \geq \frac{1}{2^n}\binom{n}{n/2}\mathsf{pm}(\underbrace{U_{n/2} \cup L_{n/2}}_{D_n}) \approx_n 3^n.$$

6.2. Matchings across a line. We present a simple construction with about 2^n different crossing-free perfect bipartite matchings across a line.

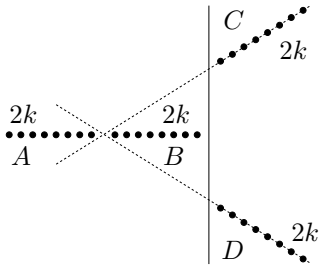


FIG. 13. *The lower bound construction for crossing-free perfect matchings across a line.*

Assume that $n = 8k$, and refer to Figure 13. Take two disjoint horizontal segments that lie on the x-axis to the left of the y-axis, and place on each of them $2k$ points. Denote by A (resp., B) the set of points on the left (resp., right) segment. The set L is $A \cup B$. To form R, draw two lines that separate A and B, one with positive slope and one with negative slope. Place $2k$ points on each of these lines to the right of the y-axis, and denote the set on the line with positive (resp., negative) slope by C (resp., D). The set R is $C \cup D$.

In order to specify a crossing-free perfect bipartite matching, we proceed as follows: Split A into two sets A_C and A_D of size k each, split B into two sets B_C and B_D of size k each, split C into two sets C_A and C_B of size k each, and split D into two sets D_A and D_B of size k each. The total number of choices is $\binom{2k}{k}^4 \approx_k 2^{8k} = 2^n$.

Now we match A_C with C_A, A_D with D_A, B_C with C_B, and B_D with D_B, which can always be done in a unique way that is noncrossing; see Figure 13.

We have thus shown that the following theorem holds.

THEOREM 6.1. *The maximum number of crossing-free perfect bipartite matchings between two separated sets, each of $\frac{n}{2}$ points, is at least $\binom{2\lfloor n/8\rfloor}{\lfloor n/8\rfloor}^4 \approx_n 2^n$.*

Clearly, this also serves as a lower bound for the more general case of perfect left-right matchings, for which we were not able to improve over the 2^n bound.

7. Discussion, open problems.

Relating the basis-constants. For $n \in \mathbb{N}$, let $\mathsf{pm}(n) := \max_{|P|=n} \mathsf{pm}(P)$ and $c_{\mathsf{pm}} := \limsup_{n\to\infty} \sqrt[n]{\mathsf{pm}(n)}$. (In fact, there is a unique limit for n over the even integers.) In an analogous fashion, define the constants c_{ma}, c_{sc}, c_{cfp}, and c_{lrpm} for the corresponding matching bounds. Also, define

$$\mathsf{rdpm}(n) := \sup_{\mu} \mathbf{E}_{\mu,n}[\mathsf{pm}(P)] \quad \text{and} \quad c_{\mathsf{rdpm}} := \limsup_{n\to\infty} \sqrt[n]{\mathsf{rdpm}(n)},$$

where μ ranges over all distributions of the plane which let two points coincide with probability 0, and $\mathbf{E}_{\mu,n}$ denotes the expectation for P a set of n i.i.d. points from distribution μ.

Apart from the absolute bounds that we derived for these constants, we have shown a number of relations among them, e.g.,

$$\begin{aligned}
&c_{\mathsf{pm}} \le 2^{1/3}\,5^{-1/6}\,c_{\mathsf{ma}} \qquad \text{(note also that } c_{\mathsf{ma}} \le c_{\mathsf{pm}} + 1\text{)},\\
&c_{\mathsf{sc}} \le 3\,{c_{\mathsf{lrpm}}}^2 \qquad \text{(also } c_{\mathsf{sc}} \le {c_{\mathsf{pm}}}^2\text{), and}\\
&c_{\mathsf{cfp}} \le c_{\mathsf{ma}} + 2 \qquad \text{(see remark preceding Theorem 5.2; note } c_{\mathsf{ma}} \le c_{\mathsf{cfp}}\text{)}.
\end{aligned}$$

We also derived a better upper bound on c_{rdpm} than on c_{pm} (while these constants still share the same lower bound of 3). It would be interesting to know whether that is an artifact of our proof. We believe not, supported by the following observation: If we consider four points, then in nonconvex position they have three crossing-free perfect matchings. If, however, we choose four i.i.d. points from any distribution, then they are in nonconvex position with probability less than $\frac{5}{8}$ [28], and therefore the expected number of crossing-free perfect matchings has to be less than $\frac{5}{8}\cdot 3 + \frac{3}{8}\cdot 2 = 2.625$.

Conjecture 1. $c_{\mathsf{rdpm}} < c_{\mathsf{pm}}$.

Also, can the bound for i.i.d. points be improved for specific distributions, uniform distribution in a disk, say?

Counting and enumeration. As far as we know, the algorithmic complexity of computing the number $\mathsf{pm}(P)$ of crossing-free perfect matchings for a set P of points is open—neither a polynomial algorithm is known, nor any lower bounds, $\#\mathcal{P}$-complete, say. The same is true for the numbers $\mathsf{tr}(P)$, $\mathsf{sc}(P)$, or $\mathsf{cfp}(P)$.

The situation looks somewhat more promising for enumeration. For triangulations and crossing-free spanning trees of a point set, Avis and Fukuda [8] show how to enumerate these objects in time poly(n) times the size of the output (see [27] for an application for enumeration of crossing-free graphs on a point set).

Nothing of the kind is known for perfect crossing-free matchings and spanning cycles. We mention on the side that *maximal* crossing-free matchings can be enumerated efficiently, due to a general result of that kind for maximal cliques in graphs [11]. To see this, define a graph for an n point set as follows. Let the vertices be the $\binom{n}{2}$ segments connecting pairs of points. Two such segments are connected by an edge if

they are disjoint, i.e., they neither cross nor share an endpoint. Now cliques in this graphs correspond to crossing-free matchings of the point set.

For perfect crossing-free matchings, we would need maximum cliques in the constructed graph. For these, no efficient enumeration algorithms exist (and are unlikely to exist at all), but it is still feasible that the special geometric structure allows such an algorithm for our problem.

Acknowledgment. We thank Andreas Razen for reading a draft of the paper and for several helpful comments.

REFERENCES

[1] O. Aichholzer, T. Hackl, B. Vogtenhuber, C. Huemer, F. Hurtado, and H. Krasser, *On the number of plane graphs*, in Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, Miami, FL, 2006, pp. 504–513.

[2] O. Aichholzer, F. Hurtado, and M. Noy, *A lower bound on the number of triangulations of planar point sets*, Comput. Geom., 29 (2004), pp. 135–145.

[3] O. Aichholzer and H. Krasser, *The point-set order-type database: A collection of applications and results*, in Proceedings of the 13th Canadian Conference on Computer Geometry, 2001, pp. 17–20.

[4] M. Ajtai, V. Chvátal, M. M. Newborn, and E. Szemerédi, *Crossing-free subgraphs*, Ann. Discrete Math., 12 (1982), pp. 9–12.

[5] E. E. Anclin, *An upper bound for the number of planar lattice triangulations*, J. Combin. Theory Ser. A, 103 (2003), pp. 383–386.

[6] S. G. Akl, *A lower bound on the maximum number of crossing-free Hamiltonian cycles in a rectilinear drawing of K_n*, Ars Combin., 7 (1979), pp. 7–18.

[7] H. Alt, U. Fuchs, and K. Kriegel, *On the number of simple cycles in planar graphs*, Combin. Probab. Comput., 8 (1999), pp. 397–405.

[8] D. Avis and K. Fukuda, *Reverse search for enumeration*, Discrete Appl. Math., 65 (1996), pp. 21–46.

[9] H. W. Becker, *Planar rhyme schemes*, Math. Mag., 22 (1948-49), pp. 23–26.

[10] M. Benkert, I. Reinbacher, M. van Kreveld, J. S. B. Mitchell, J. Snoeyink, and A. Wolff, *Delineating boundaries for imprecise regions*, Algorithmica, to appear.

[11] I. M. Bomze, M. Budinich, P. M. Pradalos, and M. Pelillo, *The maximum clique problem*, in Handbook of Combinatorial Optimization, Vol. 4, D.-Z. Du and P. M. Pardalos, eds., Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999, pp. 1–74.

[12] P. Brass, W. Moser, and J. Pach, *Research Problems in Discrete Geometry*, Springer-Verlag, New York, 2005.

[13] V. Capoyleas, G. Rote, and G. J. Woeginger, *Geometric clustering*, J. Algorithms, 12 (1991), pp. 341–356.

[14] V. G. Deineko, M. Hoffmann, Y. Okamoto, and G. J. Woeginger, *The traveling salesman problem with few inner points*, Oper. Res. Lett., 34 (2006), pp. 106–110.

[15] E. Demaine, *Simple Polygonizations*, http://theory.lcs.mit.edu/~edemaine/polygonization/ (version January 9, 2005).

[16] L. Deneen and G. Shute, *Polygonizations of point sets in the plane*, Discrete Comput. Geom., 3 (1988), pp. 77–87.

[17] M. O. Denny and C. A. Sohler, *Encoding a triangulation as a permutation of its point set*, in Proceedings of the 9th Canadian Conference on Computer Geometry, 1997, pp. 39–43.

[18] A. Dumitrescu, *On two lower bound constructions*, in Proceedings of the 11th Canadian Conference on Computer Geometry, 1999.

[19] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Monogr. Theoret. Comput. Sci. EATCS Ser. 10, Springer-Verlag, Berlin, 1987.

[20] A. Errera, Mém. Acad. Roy. Belgique Coll. 8°, 11 (1931), p. 26.

[21] A. García, M. Noy, and J. Tejel, *Lower bounds on the number of crossing-free subgraphs of K_N*, Comput. Geom., 16 (2000), pp. 211–221.

[22] A. García and J. Tejel, *A lower bound for the number of polygonizations of N points in the plane*, Ars Combin., 49 (1998), pp. 3–19.

[23] M. Grantson, C. Borgelt, and C. Levcopoulos, *Minimum weight triangulation by cutting out triangles*, in Proceedings of the 16th Annual International Symposium on Algorithms and Computation, Lecture Notes in Comput. Sci. 3827, Springer-Verlag, Berlin, 2006, pp. 984–994.

[24] R. B. Hayward, *A lower bound for the optimal crossing-free Hamiltonian cycle problem*, Discrete Comput. Geom., 2 (1987), pp. 327–343.
[25] F. Hurtado and M. Noy, *Counting triangulations of almost-convex polygons*, Ars Combin., 45 (1997), pp. 169–179.
[26] V. Kaibel and G. Ziegler, *Counting lattice triangulations*, in British Combinatorial Surveys, C. D. Wensley, ed., Cambridge University Press, Cambridge, UK, 2003, pp. 277–307.
[27] A. Kawamoto, M. P. Bendsøe, and O. Sigmund, *Planar articulated mechanism design by graph theoretical enumeration*, Struct. Multidiscip. Optim., 27 (2004), pp. 295–299.
[28] L. Lovász, K. Vesztergombi, U. Wagner, and E. Welzl, *Convex quadrilaterals and k-sets*, in Towards a Theory of Geometric Graphs, Contemp. Math. 342, J. Pach, ed., AMS, Providence, RI, 2004, pp. 139–148.
[29] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North–Holland Math. Library 16, North–Holland, Amsterdam, 1977.
[30] P. McCabe and R. Seidel, *New lower bounds for the number of straight-edge triangulations of a planar point set*, in Proceedings of the 20th European Workshop on Computer Geometry, 2004.
[31] T. S. Motzkin, *Relations between hypersurface cross ratios, and a combinatorial formula for partitions of a polygon, for permanent preponderance, and for non-associative products*, Bull. Amer. Math. Soc., 54 (1948), pp. 352–360.
[32] M. Newborn and W. O. J. Moser, *Optimal crossing-free Hamiltonian circuit drawings of the K_n*, J. Combin. Theory Ser. B, 29 (1980), pp. 13–26.
[33] J. Pach and G. Tóth, *Graphs drawn with few crossings per edge*, Combinatorica, 17 (1997), pp. 427–439.
[34] F. Santos and R. Seidel, *A better upper bound on the number of triangulations of a planar point set*, J. Combin. Theory Ser. A, 102 (2003), pp. 186–193.
[35] R. Seidel, *On the number of triangulations of planar point sets*, Combinatorica, 18 (1998), pp. 297–299.
[36] M. Sharir and E. Welzl, *On the number of crossing-free matchings, (cycles and partitions)*, in Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, Miami, FL, 2006, pp. 860–869.
[37] M. Sharir and E. Welzl, *Random triangulations of planar point sets*, in Proceedings of the 22nd Annual ACM Symposium on Computer Geometry, Sedona, AZ, 2006, pp. 273–281.
[38] W. S. Smith, *Studies in Computational Geometry Motivated by Mesh Generation*, Ph. D. thesis, Princeton University, Princeton, NJ, 1989.
[39] R. P. Stanley, *Enumerative Combinatorics*, Vol. 2, Cambridge University Press, Cambridge, UK, 1999.

SIAM J. COMPUT.
Vol. 36, No. 3, pp. 721–739

COUNTING AND ENUMERATING POINTED PSEUDOTRIANGULATIONS WITH THE GREEDY FLIP ALGORITHM*

HERVÉ BRÖNNIMANN†, LUTZ KETTNER‡, MICHEL POCCHIOLA§, AND JACK SNOEYINK¶

Abstract. We present an algorithm to enumerate the pointed pseudotriangulations of a given point set, based on the greedy flip algorithm of Pocchiola and Vegter [*Discrete Comput. Geom.* 16 (1996), pp. 419–453]. Our two independent implementations agree and allow us to experimentally verify or disprove conjectures on the numbers of pointed pseudotriangulations and triangulations of a given point set. (For example, we establish that the number of triangulations is bounded by the number of pointed pseudotriangulations for all sets of up to 10 points.)

Key words. algorithm, pseudotriangulation, triangulation, combinatorics, enumeration

AMS subject classifications. 52C25, 52C45, 65D18, 68U05, 68W05

DOI. 10.1137/050631008

1. Introduction.

Algorithms that perform computations on sets of points in the plane frequently benefit from using the points to decompose the plane into simpler regions: triangulations, Voronoi diagrams, visibility maps, and Delaunay tessellations are good examples [18]. Decompositions called pseudotriangulations or geodesic triangulations have been studied for convex sets and for simple polygons in the plane because of their applications to visibility [39, 40], ray shooting [17, 22], covering and separation [42], and stretchability of pseudolines arrangements [44]. They have been used in a number of kinetic data structures (KDSs) for collision detection among moving objects in the plane [1, 30, 31] because they can be maintained by edge flips and can form a partition of the free space, whose size is related to minimum link separators of the objects. Streinu used them in an elegant proof, where it is possible to unfold any chain in the plane without self-intersection [50].

Pseudotriangulations possess versatility and uniformity properties that make them worthy of study. For instance, there is an edge-flip operation that applies to any internal edge in a pointed pseudotriangulation, unlike the edge-flip operation in triangulations. In section 2 we recall the definition of the edge-flip operation on pointed pseudotriangulations and some results on the graph of pointed pseudotriangulations, in which two pseudotriangulations are adjacent if they differ by a single edge flip.

The main result of this paper is an algorithm that uses edge flips to enumerate all pointed pseudotriangulations of a given point set. There have been many interesting results on counting and enumerating triangulations for a given set of points in the

*Received by the editors April 24, 2005; accepted for publication (in revised form) April 14, 2006; published electronically October 16, 2006. A preliminary version of this paper appeared in *Proc. ALENEX*, 2005.

http://www.siam.org/journals/sicomp/36-3/63100.html

†CIS, Polytechnic University, Six Metrotech, Brooklyn NY 11201 (hbr@poly.edu). This author's research was partially supported by NSF Career grant 0133599.

‡Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany (kettner@mpi-sb.mpg.de).

§Ecole Normale Supérieure Paris, 45 Rue d'Ulm, 75230 Paris Cedex 05, France (pocchiola@di.ens.fr).

¶Department of Computer Science, Sitterson Hall, University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-3175 (snoeyink@cs.unc.edu).

plane. In a series of upper bounds on the number of triangulations of a given n-point set, a bound of $59^{n+o(n)}$ by Santos and Seidel [48] recently replaced the previous best result of $2^{8.12n+O(\log n)}$ by Denny and Sohler [20]. There are examples of point sets with as many as $72^{0.5n-\Theta(\log n)} = \Omega(8.48^n)$ triangulations [8], and it is even known that for *any* n-point set in general position in the plane, its number of triangulations is $\Omega(2.33^n)$ [9]. Aichholzer [3] has a counting algorithm (that can be executed from his Web page for small point sets [2]) and Bespamyatnikh [14] and Ray and Seidel [46] present enumeration algorithms. There remain elementary open questions, such as what point sets have the most and the fewest triangulations. (Aichholzer [2] maintains a list of the leading examples for up to 20 points.)

Less is known about the number of pointed pseudotriangulations of a given point set. Even the following conjecture is open.

CONJECTURE 1 (see [19]). *For any finite set of points in general position in the plane, its number of triangulations is bounded by its number of pointed pseudotriangulations with equality iff the points are in convex position.*

Randall et al. [45] have established that the number of pointed pseudotriangulations of any point set with i points inside the convex hull is upper bounded by 3^i times its number of triangulations. When combined with the best known upper bound on the number of triangulations, this gives that the number of pointed pseudotriangulations of an n-point set is bounded by $177.3^{n+o(n)}$. Bespamyatnikh has extended his enumeration algorithm [14] to pointed pseudotriangulations, but has yet to implement it. Also, his algorithm cannot efficiently enumerate only the pointed pseudotriangulations containing a given set of edges, which our algorithm can do. This in turn implies a strong connectivity property that is useful when studying the graph of pointed pseudotriangulations (see section 2.2, see also [47]).

Our algorithm, presented in section 3, is based on the greedy flip algorithm of Pocchiola and Vegter for computing the visibility complex of a scene of n convex objects in the plane [40], and on its extension developed in [11]. In section 4, we provide some implementation details; we have produced two independent implementations, which may be obtained from http://geometry.poly.edu/pstoolkit/ and www.cs.unc.edu/Research/compgeom/pseudoT/. In section 5, we present the results of experiments that explore basic conjectures on the number of pointed pseudotriangulations and triangulations. Both implementations agree in these experiments.

Note that Tutte [51] and others have studied the number of topological embeddings of triangulations and rooted triangulations when the locations of vertices are not specified. Li and Nakano [35] enumerate topologically distinct triangulations with a prescribed number of points on their boundaries. We focus strictly on the geometric question of when the vertex set must be a given set of points in the plane.

2. Minimum, or pointed, pseudotriangulations.

In this section, we recall the definition and basic properties of *pointed* pseudotriangulations. These are also called *minimum* by Streinu [50] because pointed pseudotriangulations are precisely the pseudotriangulations with the minimum number of edges. Since these are the only kind of pseudotriangulations we will consider in this paper, we will omit the words "pointed" and "minimum" and simply refer to pseudotriangulations.

2.1. Definitions.

Pseudotriangulations were defined by Pocchiola and Vegter [41] for the case of two-dimensional convex sets. In this paper, we are solely concerned with pseudotriangulations of point sets. One could replace each point p by a disk with center p and radius ϵ, for some small $\epsilon > 0$, and work within this framework. There are many subtleties involved, however, which warrant a study of pseudotriangulations

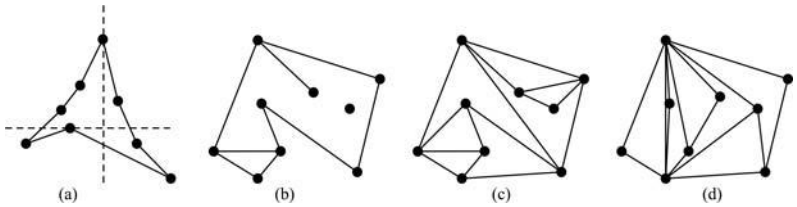


FIG. 1. (a) *A pseudotriangle and its horizontal and vertical tangents.* (b) *An acyclic planar set of edges.* (c) *A maximal acyclic planar set of edges or, put differently, a pseudotriangulation.* (d) *The canonical sorted pseudotriangulation.*

of point sets in their own right. For instance, two disks have four bitangents, only three of which can be nonintersecting; if we collapse the disks to points, all bitangents collapse to a single edge. To ease the reader's task and prevent circular dependencies, we do not make reference to the case of convex sets except in the proof of the flip property (Theorem 3).

Pseudotriangles. A *pseudotriangle* is a simple polygon in the plane that has exactly three vertices, called *corners*, with internal angle less than π. The three corners of a pseudotriangle decompose its boundary into three concave chains. The notions of *pseudoquadrangle* and *pseudo-n-gon* are defined similarly by allowing four or n corners.

Let T be a pseudotriangle. A *tangent* to T is a line l in the plane that goes either through a corner p of T and separates the two edges of T incident upon p, or through a noncorner p of T and does not separate the two edges of T incident upon p. A pseudotriangle has exactly one tangent line parallel to any given line. (See Figure 1(a).)

Pseudotriangulations. Let P be a set of n points in general position (i.e., no three collinear points) and let E be the set of $n(n-1)/2$ undirected line segments with endpoints in P (edges for short). For the purpose of this paper, we assume that there are no two parallel edges and no edge parallel to the x-axis.[1]

A subset H of E is called *planar* if its edges are pairwise interior disjoint, and it is called *acyclic* if for any endpoint p of an edge of H there is a line through p that leaves the edges of H incident upon p all on the same side. (See Figure 1(b).)

Following Streinu [50] we define a *pseudotriangulation* of P to be a maximal (for the inclusion relation), acyclic, and planar subset of E; note that the set of edges of the convex hull of P is included in every pseudotriangulation. (See Figures 1(c) and (d) for an illustration.) The following lemma recalls the connection between the acyclicity and the number of edges. It is stated and proved in [50, Theorem 3.1], and an easy proof can be adapted from [41, Lemma 2].

LEMMA 2. *The bounded faces of the subdivision of the plane induced by a pseudotriangulation of P are pseudotriangles. Furthermore the number of pseudotriangles of the subdivision is $n-2$ and its number of edges is $2n-3$.*

For points in convex position, the set of pseudotriangulations is exactly the set of triangulations; the external angle of each vertex on the convex hull is greater than π, so triangulations are acyclic. We define a canonical *sorted pseudotriangulation*, as in Figure 1(d), for any set of n points in general position by the following construction: Sort the points lexicographically by (x, y) coordinates and form a triangle with the first three points; then for each subsequent point in order, add one pseudotriangle by

[1] These two restrictions can be lifted, and indeed should be, in a good implementation. In section 5, we apply the algorithm to the point sets from the database of Aichholzer, Aurenhammer, and Krasser [5] which obey these restrictions.

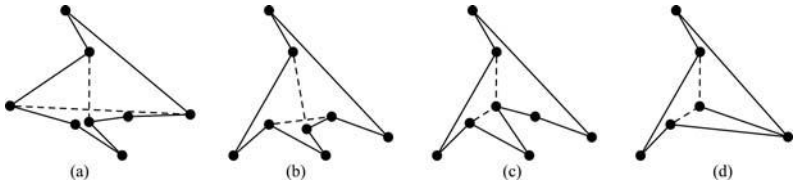


FIG. 2. *Four pseudoquadrangles. The last one,* (d)*, uses both sides of one segment. Each pseudoquadrangle has two diagonals (dashed segments), one on each shortest path that joins opposite corners. Each diagonal forms a pair of pseudotriangles, and an edge flip replaces one diagonal with the other.*

creating two tangents to the convex hull. This pseudotriangulation, called incremental in [1], has links to rigidity (where it is called a Henneberg construction) and has been used in collision detection.

Edge flips in pseudotriangulations. In a triangulation, an *edge flip* replaces any edge whose adjacent triangles form a convex quadrilateral by the opposite diagonal of that quadrilateral. Edge flips, sometimes known as Lawson flips, are useful tools for studying the properties of triangulations and for generating triangulations algorithmically [25, 33, 37].

The situation is even nicer in pseudotriangulations of point sets, since any edge that is inside the convex hull can be flipped, as described below. An almost identical flip operation is defined for pseudotriangulations of disks by Pocchiola and Vegter [40]. For completeness, we give the definition below. In section 3.2, we consider algorithms for computing flips.

Consider an edge e that is adjacent to two neighboring pseudotriangles. Each endpoint of e is a corner in at least one of the neighboring pseudotriangles, since each vertex has at most one angle that is greater than π; see Figure 2 for examples. Removing edge e merges the two neighboring pseudotriangles into a pseudoquadrangle. Indeed, at each endpoint of e either two corners merge into one corner or one corner merges with an angle greater than π. Thus, the six corners of the original two pseudotriangles become the four corners of a pseudoquadrangle.

A *diagonal* for a pseudoquadrangle is defined by connecting opposite corners with a shortest path through the interior. Such a shortest path cannot go through one of the other two corners, so it must separate the pseudoquadrangle into exactly two regions, each of which contains one of the other two corners. Hence, both regions must be pseudotriangles sharing exactly one edge e, which is the sought-after diagonal. Since there are two pairs of opposite corners, a pseudoquadrangle has two diagonals.

Now, any nonhull edge e is one diagonal of the pseudoquadrangle formed by removing e. We define the *edge flip* for e as the operation that removes e and replaces it with the other diagonal e'. Figure 2 shows four examples. It should be clear that flipping the new edge e' restores the original pseudotriangulation.

2.2. Graph and polytopes of pseudotriangulations. Formally, the *graph of pseudotriangulations* for a given set of points contains a node for each possible pseudotriangulation. An arc connects two nodes if the two corresponding pseudotriangulations differ by a single edge flip. The graph is undirected since flips are reversible.

In a previous version, we had shown, using the canonical sorted pseudotriangulation, that the graph of pseudotriangulations is connected and given a bound of $O(n^2)$ on its diameter. In 2003, Aichholzer et al. [7] improved the diameter bound to

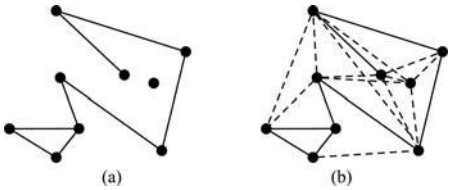


FIG. 3. (a) *An acyclic and planar subset K of E.* (b) *The set $E_K = \{e \mid K \cup \{e\}$ is acyclic and planar$\}$.*

$O(n \log^2 n)$, Bereg [13] to $O(n \log n)$, and more recently, Aichholzer et al. [4] improved it to $O(n \log k)$, where k is the number of convex layers of the point set.

The graph of pseudotriangulations has an even stronger connectivity property. If we consider only the pseudotriangulations that contain a chosen (acyclic and planar) set of edges, then we get a subgraph induced by the nodes corresponding to those pseudotriangulations, whose arcs join nodes that correspond to flips of the edges not in the chosen set of edges. This subgraph is connected. As we will see, this is a simple consequence of our enumeration algorithm. To put it in a different terminology [36], one can say that the poset, ordered by reverse inclusion, of planar acyclic sets of edges that contain the edges of the convex hull is an abstract polytope whose 1-skeleton is the graph of pseudotriangulations. Rote, Streinu, and Santos have shown that this abstract polytope is actually geometrically realizable, using a beautiful relation to minimally rigid graphs [47].

3. Enumerating pseudotriangulations. Our goal is to enumerate the set of pseudotriangulations over a given set of points. To this end we are going to define a total order $\prec$ on the set of edges and a binary tree of pseudotriangulations whose leaves considered as increasing sequences of edges are the pseudotriangulations ordered lexicographically; furthermore two adjacent pseudotriangulations in the tree are either identical or related by a flip operation. Our enumeration algorithm is a traversal of this tree guided by the aforementioned total order $\prec$. Our technique can also be applied to enumerate the pseudotriangulations that contain a given set of edges.

3.1. The flip property. We introduce some definitions and prove a flip property that is essential to proving the correctness of the enumeration algorithm. For each edge $e \in E$, define $\Theta(e)$ as the angle in $[0, \pi)$ that the edge e directed upward makes with the positive horizontal direction Ox.

Given an acyclic planar subset of edges $K \subseteq E$, we denote by E_K the set of edges $e \in E$ that can be used to complete K to a pseudotriangulation, i.e., such that $K \cup \{e\}$ is acyclic and planar. See Figure 3 for an illustration.

We define a partial order $\prec_K$ on E_K as follows: $e \prec_K e'$ if there exists a sequence of edges $e_1 = e, e_2, \ldots, e_k = e'$ such that e_i and e_{i+1} share a common endpoint and angles $\Theta(e_i) < \Theta(e_{i+1})$. According to the general position assumption, two edges sharing an endpoint have different angles and therefore are comparable. It follows that the edges of a pseudotriangle are pairwise comparable and are encountered in increasing order when traversing the boundary of the pseudotriangle counterclockwise, starting from its point of horizontal tangency. Following [40, Lemma 7] we observe that two crossing edges in E_K are the diagonals of some pseudoquadrangle (with edges in E_K) and consequently are comparable with respect to $\prec_K$.

A *filter* for a poset $(X, \prec)$ is a subset I of elements such that for any $x \prec y$, if $x \in I$ then $y \in I$. In particular, to each angle θ corresponds a filter $I_{K,\theta}$ of the poset

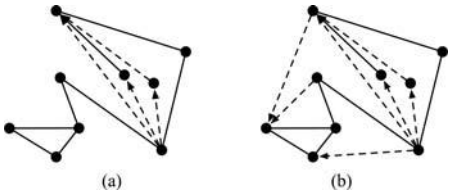


FIG. 4. *The set $G(I_{\pi/2})$ is constructed by adding edges from the filter $I_{\pi/2}$, i.e., edges with increasing angles greater than $\pi/2$, until a pseudotriangulation is formed.* (a) *illustrates that it is not sufficient to consider angles in $[\pi/2, \pi)$ only, but by "wrapping around"* (b) *we complete a pseudotriangulation.*

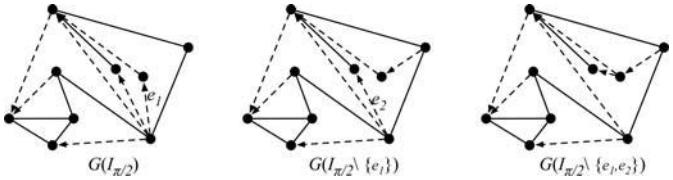


FIG. 5. *Illustration of the flip property for points.*

$(E_K, \prec_K)$ whose elements are the edges e' of E_K whose angle $\Theta(e')$ is greater than θ. Note that $I_{K,0} = E_K$, and $I_{\emptyset,0} = E$.

For any filter I of the poset $(E_K, \prec_K)$, we define a maximal planar acyclic set of edges $G(I) = \{e_1, e_2, \ldots, e_k\}$ recursively: edge e_1 is minimal in I, and, for $i \geq 1$, edge e_{i+1} is minimal in the set of edges $e \in I \setminus \{e_1, \ldots, e_i\}$ such that $K \cup \{e_1, \ldots, e_i, e\}$ is acyclic and planar. Since two edges that cross or that share an endpoint are comparable, $e_1, e_2, \ldots, e_k$ is independent on the choice of the minimal element at each step, or in other words, the set $G(I)$ is well defined.

We would like for $G(I)$ to be a pseudotriangulation, and for this it suffices to make sure that $G(I)$ has $2n - 3$ edges. This is not always possible, however, due to the fact that we chose the principal determination of Θ in $[0, \pi)$, therefore introducing a discontinuity in the comparisons, and forcing the algorithm to stop perhaps too early. (See Figure 4.) To circumvent this, we replace, in the previous definitions, the set of edges E_K with its infinite cover $\mathbb{E}_K = E_K \times \mathbb{Z}$: elements of $\mathbb{E}_K$ are still called edges, and the angle $\Theta(v)$ of an edge $v = (e, k)$ of $\mathbb{E}_K$ is defined to be the real $\Theta(e) + k\pi$. The operator on $\mathbb{E}_K$ that increases the angle of an edge by π is denoted ι. It is not hard to see that if I is a filter of $(\mathbb{E}_K, \prec_K)$, then its projection on the first factor E_K is onto, from which we deduce that $G(I)$ is a pseudotriangulation. In this context we redefine the flip operation as follows: To flip e in $G(I)$ means to replace e with $\iota(e)$ if $e \in \mathbb{K} := K \times \mathbb{Z}$ or if e is a hull-edge, otherwise to perform an edge flip on e and assign the angle of the diagonal by adding a multiple of π to fall into the range $(\Theta(e), \Theta(e) + \pi)$.

The pseudotriangulation $G(I_{\emptyset,0})$ is called the *horizontal greedy pseudotriangulation* and plays a particular role in our enumeration algorithm. Below we explain how to efficiently compute this pseudotriangulation.

We are now in a position to state the *flip property*. This property states that flipping a minimal edge in a pseudotriangulation of the form $G(I)$ results in a pseudotriangulation of the form $G(J)$, and this will be crucial in our enumeration algorithm. See Figure 5 for an illustration.

THEOREM 3 (flip property for points). *Let I be a filter of the poset $(\mathbb{E}_K, \prec_K)$ and let e be minimal in I. Then $G(I \setminus e)$ is obtained from $G(I)$ by flipping e.*

The proof relies on the theory of pseudotriangulations developed for bounded two-dimensional convex sets in [11, 40]. The heart of the proof is to replace each point $p \in P$ by the disk with center p and radius ϵ, for some small $\epsilon > 0$, and to derive the flip property for points from the "flip property for disks" (cf. [40, Theorem 12] and [11, Theorem 5]). There are many subtleties involved, however. For one thing, disks have four bitangents, only three of which can be nonintersecting, and they all map to a single edge of the point set. Nevertheless, with a bit of care, it is possible to carry the flip property from the case of disks to the case of points.

We briefly recall the terminology and the results of [11, 40] needed for our purpose. Let $O_1, O_2, \ldots, O_n$ be a collection of n pairwise disjoint bounded closed convex subsets of the plane with nonempty interiors and regular boundaries (obstacles or disks for short). We assume that there is no line tangent to three disks. A *bitangent* is a closed undirected line segment whose supporting line is tangent to two disks at its endpoints. A *free* bitangent is a bitangent whose interior lies in free space, defined as the complement of the disks. In the following considerations all bitangents are free. A set of (free) bitangents is called *planar* if its elements are pairwise disjoint. A *pseudotriangulation* (of the O_i's) is a planar set of bitangents that is maximal for the inclusion relation. It is known that a pseudotriangulation contains $3n-3$ bitangents that decompose the convex hull of the disks into $2n-2$ *pseudotriangles* where, in this context, a pseudotriangle is a simply connected region of the plane whose boundary consists of three convex curves that share a tangent at their common endpoint and which is included in the triangle formed by the three endpoints of these convex curves.

We denote by B the set of (free) bitangents and we introduce its infinite cover $\mathbb{B} = B \times \mathbb{Z}$: elements of $\mathbb{B}$ are still called bitangents; the angle $\Theta(v)$ of the bitangent $v = (b, k) \in \mathbb{B}$ is defined to be the real $\Theta(b) + k\pi$, where $\Theta(b)$ is the angle in $[0, \pi)$ that b, oriented upward, makes with the horizontal positive direction Ox, and the direction of $v \in \mathbb{B}$ is the unit vector $(\cos\Theta(v), \sin\Theta(v)) \in \mathbb{S}^1$; the operator that increases the angle of a bitangent by π is denoted ι.

Given a planar subset H of B, we introduce the set B_H of bitangents of B that cross no bitangent of H properly (thus $H \subseteq B_H$). The set $\mathbb{B}_H$ is endowed with a partial order $\prec_H$ defined as follows: $b \prec_H b'$ if there exists a sequence of bitangents $b_1 = b, b_2, \ldots, b_k = b'$ in $\mathbb{B}_H$ such that b_i and b_{i+1} touch the same oriented disk[2] and such that $\Theta(b_i) < \Theta(b_{i+1})$. Each (proper) filter I of $(\mathbb{B}_H, \prec_H)$ is associated with its so-called greedy pseudotriangulation

$$G(I) = \{b_1, b_2, \ldots, b_{3n-3}\} \subset I$$

defined as follows: (1) b_1 is minimal in I, and (2) b_{i+1} is minimal in the subset of bitangents of I crossing none of the bitangents $b_1, b_2, \ldots, b_i$. Since crossing bitangents are comparable (cf. Lemma [40, Lemma 7]), $G(I)$ is well defined and is a superset of the set of minimal elements of I. To flip b in $G(I)$ means to replace b with $\iota(b)$ if $b \in \mathbb{H} := H \times \mathbb{Z}$ or if b is a hull bitangent, otherwise to replace b with the second diagonal (with the appropriated angle) of the pseudoquadrangle obtained by merging the two pseudotriangles incident upon b in $G(I)$.

[2] An oriented disk is a disk with a "direction," "sense," or "orientation" assigned to its boundary. An oriented disk and a bitangent touch each other, or are tangent to each other, if their directions at the point of touching are the same.

THEOREM 4 (flip property for disks [11, 40]). *Let b be minimal in the filter I of the poset $(\mathbb{B}_H, \prec_H)$. Then $G(I \setminus b)$ is obtained from $G(I)$ by flipping b.*

According to the flip property, the mapping that associates with the bitangent $b \in \mathbb{B}_H$ the bitangent $b' \in \mathbb{B}_H$ defined by $\{b'\} = G(I \setminus b) \setminus G(I)$, where b is minimal in I, is well defined (because it is independent of I), one-to-one, and onto; the bitangent b' is denoted $\phi(b; H)$.

We turn now to the proof of the flip property for points. We split the proof into several lemmas. The key idea of the proof is to define an epimorphism of posets to carry the flip property from the case of disks to the case of points. Before defining this epimorphism, we reformulate the greedy procedure in terms more suitable for our subsequent analysis.

LEMMA 5. *Let I be a filter of $\mathbb{B}_H$ and let F be* initial *in I; i.e., $I \setminus F$ is a filter. Then $G(I) = G(I(F))$, where $I(F)$ is the filter of the poset $\mathbb{B}_{H \cup G(F)}$ defined by $I(F) = I \cap \mathbb{B}_{H \cup G(F)}$. A similar result holds when taking K and $\mathbb{E}_K$ for H and $\mathbb{B}_H$, respectively.*

Proof. Since F is initial, $G(F)$ is a subset of $G(I)$. The lemma follows easily. □

Now we turn to the construction of the epimorphism. For $\epsilon > 0$ let $O_i(\epsilon)$ be the disk with center p_i and radius ϵ. Since the points are in general position, there exists a real $\epsilon_0 > 0$ such that for all $\epsilon < \epsilon_0$ no line pierces three of the disks $O_i(\epsilon)$ and no horizontal line pierces two disks. (Remember that we assumed that no two points have the same ordinate.) We choose such an ϵ, introduce the set B of $2n(n-1)$ (free) bitangents of the $O_i(\epsilon)$s, and denote by η the four-to-one mapping that associates with a bitangent b in B tangent to the disks O_i and O_j the edge $\eta(b)$ of E with endpoints p_i and p_j. Our first lemma provides a characterization of acyclicity in terms of the crossing predicate.

LEMMA 6. *Let K be a planar subset of E. Then K is acyclic iff for all maximal (for the inclusion relation) planar subsets H of $\eta^{-1}(K)$ one has $\eta(H) = K$.*

Proof. The "if part" is easy. To prove the "only if part" we show that if there exists an edge $e \in K$ and a maximal planar subset H of $\eta^{-1}(K)$ such that $e \notin \eta(H)$, then K is not acyclic. Let p and q be the endpoints of e and let p^+ (resp., p^-) be the set of edges $e' \in K$ such that (i) p is the endpoint of e', i.e., $e' = [p, r]$ for some point r, and (ii) the triplet of points (p, r, q) is oriented counterclockwise (resp., clockwise).

The assumption that $e \notin \eta(H)$ is equivalent to saying that for all $b \in \eta^{-1}(e)$ there exists a bitangent b' of H such that b and b' are crossing. A simple case analysis shows that this is equivalent to saying that the sets $p^+ \cup q^+$, $p^+ \cup q^-$, $p^- \cup q^+$, and $p^- \cup q^-$ are nonempty; from this, we deduce that p^+ and p^- (or q^+ and q^-) are nonempty and consequently that K is not acyclic. □

Our next lemma is the key to the construction of an epimorphism from some $B_H \subseteq \eta^{-1}(E_K)$ onto E_K.

LEMMA 7. *Let K be an acyclic and planar subset of E and let H be a maximal (for the inclusion relation) planar subset of $\eta^{-1}(K)$. Then*

1. *$\eta(\mathbb{B}_H) = \mathbb{E}_K$, and*
2. *if $b \prec_H b'$, then $\eta(b) \preceq_K \eta(b')$.*

Proof. Claim (1) is a consequence of Lemma 6. Indeed, let $K' = K \cup \{e\}$ be planar and acyclic and let $H' \supseteq H$ be a maximal planar subset of $\eta^{-1}(K')$ that contains H. According to Lemma 6 applied to the pair K', H', one has $\eta(H') = K'$ and consequently some bitangent of $\eta^{-1}(e) \in B_H$. Claim (2) is a simple consequence of Claim (1) and the observation that if b and b' touch the same oriented disk with

$\Theta(b) < \Theta(b')$, then $\eta(b)$ and $\eta(b')$ share a common endpoint and $\Theta(\eta(b)) \leq \Theta(\eta(b'))$ with equality iff $\eta(b) = \eta(b')$. □

In other words, the restriction η_H of η to $\mathbb{B}_H$ is a mapping onto $\mathbb{E}_K$ that preserves the orderings. We show that η_H preserves also the greedy pseudotriangulations.

LEMMA 8. *Let K be an acyclic and planar subset of E, let H be a maximal planar subset of $\eta^{-1}(K)$, and let I be a filter of $(\mathbb{E}_K, \prec_K)$. Then*

1. *$\eta_H^{-1}(I)$ is a filter of $(\mathbb{B}_H, \prec_H)$, and*
2. *$\eta(G(\eta_H^{-1}(I))) = G(I)$.*

Proof. Claim (1) is a simple consequence of Lemma 7. Claim (2) is proved by induction on the set of planar acyclic subsets K of E ordered by reverse inclusion. Let $J = \eta_H^{-1}(I)$. This is clearly valid if K is maximal since in that case $\mathbb{E}_K = \mathbb{K}$, $\mathbb{B}_H = \mathbb{H}$, and consequently $G(I) = I \setminus \iota(I)$ and $G(J) = J \setminus \iota(J)$ from which we deduce that $\eta(G(J)) = G(I)$. Assume now that K is not maximal and let e be minimal in I. If $e \notin K$, we set $K' = K \cup \{e\}$ and $H' = H \cup G(\eta_H^{-1}(\{e\})$. According to Lemma 5, $G(I) = G(I')$ and $G(J) = G(J')$, where $I' = I(\eta_H^{-1}\{e\})$ and $J' = J(\{e\})$. One can check that $I' = \eta_{H'}^{-1}(J')$, from which the result follows by induction since $K \subset K'$. If $e \in K$, we replace e with an initial segment of J that contains exactly one element not in K and proceed similarly. □

We are now ready to carry the flip property from the case of disks to the case of points.

Proof of Theorem 3. Let $J = \eta_H^{-1}(I)$ and let $F = \eta_H^{-1}(e)$. Note that F is initial in J since $J \setminus F = \eta_H^{-1}(I \setminus e)$ (cf. Lemma 8, claim 1). Thanks to the flip property for disks, the set $G(J \setminus F)$ is obtained from $G(I)$ by successively flipping the bitangents of F. Therefore one has

$$G(J \setminus F) = (G(J) \setminus G(F)) \cup \phi(F; H) \setminus F$$

and consequently, according to Lemma 8,

$$G(I \setminus e) = (G(I) \setminus e) \cup \eta(\phi(F, H) \setminus F).$$

Since $G(I \setminus e)$ and $G(I)$ have the same cardinality, namely $2n - 3$, it follows that $G(I \setminus e) \setminus G(I)$ is reduced to a single edge, which is one of the edges of $\eta(\phi(F; H) \setminus F)$. □

3.2. Algorithms for edge flip. In this section we sketch two implementations of the edge-flip operation, assuming that the pseudotriangulation is stored in a data structure that allows us to access its edges in order around a given face. Standard structures for planar subdivisions, such as doubly connected edge lists or quadedge [18, 23], provide this.

Rotational sweep for edge flip. We can determine the new diagonal obtained by flipping edge e using a rotational sweep somewhat similar to the rotating caliper [18]. The algorithm proceeds by rotating parallel tangents simultaneously along the interiors of the two pseudotriangles adjacent to e. Starting from the edge e that we want to flip, the two tangents initially coincide but have opposite orientations. If we sweep through the angles, the two tangents immediately separate and meet again only when they reach the new diagonal. We can discretize this sweep because the tangents rotate around vertices until they are collinear with the next edge of a pseudotriangle. Then they advance to the next vertex. At corners, the tangent changes its orientation with respect to the edge orientation. The sweep terminates when the tangents again coincide. See Figure 8 for an implementation.

A matroidal flip algorithm. The predicate in the rotational sweep is a test ordering two vectors. One can also give an implementation that only uses the orientation predicate `left_turn`(p, q, r), which returns true iff the point sequence p, q, r forms a left turn. We can call such an algorithm "matroidal," in that it uses only information about the order type of the points [15, 32]. Such algorithms are usually better in that they have fewer degenerate configurations, lower arithmetic complexity, and generalize to other matroids.

The idea behind the algorithm is to identify the flip as the only diagonal edge on the shortest path connecting the opposite corners. The funnel algorithm of Lee and Preparata [34] can be modified [24] to compute shortest paths in linear time and to return the unique edge not on the boundary of the pseudoquadrangle. Alternatively, one can compute common tangents for the pairs of chains in a pseudoquadrangle to identify the diagonals. Tangents for two separated chains can be found in $O(\log n)$ time [29, 38]. When computing the visibility graph of a set of convex obstacles, Angelier and Pocchiola [11] use a clever amortization scheme to compute tangents in $O(1)$ time apiece.

3.3. The enumeration algorithm. In this section, we consider the total order $<$ on E and $\mathbb{E}$ induced by Θ. This order is compatible with, and linearly extends, $(E, \prec)$. Although we assume general position, the case of parallel edges could be handled by considering a linear extension of $(E, \prec)$.

In the algorithm, we speak of edges in E as colored red or blue. The red edges are fixed and will not be flipped; the blue edges can be flipped. We now describe the following binary tree $\mathcal{T} = \mathcal{T}(P)$ of {red,blue}-colored pseudotriangulations of P. Each node of the tree corresponds to a colored pseudotriangulation G, and we identify the node of the tree with its pseudotriangulation G. The tree is defined as follows:

1. The root of $\mathcal{T}$ is the horizontal greedy pseudotriangulation $G(I_{\emptyset,0})$; all its edges are blue.
2. Let G be a node of $\mathcal{T}$: If either (i) a blue edge of G has an angle $\geq \pi$, or (ii) all the edges of G are red, then G is a leaf of the tree. Otherwise, let e be a minimal blue edge, e.g., the blue edge with minimum angle $\Theta(e)$. The right child of G is obtained from G by flipping e, and its left child is obtained by changing the color of e to red.

A leaf G satisfying 2(i) is called a blue leaf, and a leaf satisfying 2(ii) is called a red leaf. Blue leaves stop the algorithm from enumerating a pseudotriangulation G several times; without stopping for blue leaves, the tree would be infinite, and each pseudotriangulation would be reported for every value of Θ with the same remainder modulo π.

The algorithm simply explores the tree $\mathcal{T}$ by a depth-first traversal, visiting the left child before the right child, and reporting the red leaves in the order in which they are discovered.

The algorithm is fully described once we explain how to find the minimal blue edge e. In the most direct implementation, the blue edges are stored in a priority queue, ordered by Θ. The edge e at the top of the queue is removed upon descending to either child, and edge e' is enqueued when descending to a right child iff its angle $\Theta(e') < \pi$ (otherwise the right child is a blue leaf and the recursion stops).

It is not necessary to store the priority queue in the recursion stack if we simply add edge e back to the queue when returning to the parent after visiting the right subtree. Thus, the stack grows by $O(1)$ at each recursive call.

THEOREM 9. *The set of red leaves of $\mathcal{T}(P)$ ordered from left to right (in the order they are reported by the algorithm) is the set of pseudotriangulations of P ordered lexicographically by Θ.*

Proof. Let G be a pseudotriangulation with edges in E and let $G_0, G_1, \dots G_i, \dots G_k$ be the path in the tree defined inductively: G_0 is the root of the tree and G_{i+1} is the left child of G_i if the minimal blue edge of G_i belongs to G; otherwise G_{i+1} is its right child. Let K_i be the set of red edges of G_i, edge e_i be the minimal blue edge of G_i, and filter $I_i = I_{K_i,\Theta(e_i)}$ of E_{K_i}. We claim that

1. $K_i \subseteq G$,
2. $G \setminus K_i \subset I_i$,
3. $G(I_i) = (G_i \setminus K_i) \cup \iota(K_i)$,

from which we deduce that G_k is a red leaf and $G = G_k$.

Claims (1), (2) and (3) are easily proven by induction on i using the flip property of the previous section. The proof is finished by noting that the red leaves of the tree are pseudotriangulations with edges in E. □

Note that the theorem is also valid for any total order $<$ that is a linear extension of $\prec$, yielding a well-defined tree $\mathcal{T}_<(P)$. Since Θ induces such a total order and is easy to compute, thanks to the geometry, it is convenient to use it. See remark 1 below.

THEOREM 10. *The height of the tree $\mathcal{T}(P)$ is at most $n(n-1)/2$.*

Remarks. 1. In this formulation, the algorithm depends on the order Θ of the edges, which is not implied by the orientation of all triplets of points. For this reason, the algorithm as described here is not matroidal. It is, however, possible to give a matroidal algorithm, by selecting for e *any* blue edge that is minimal for the partial order $\prec$. In this case the tree $\mathcal{T}(P)$ isn't uniquely defined, and finding such an edge is more difficult, necessitating the maintenance of the antichain ($\hat{I}$ in the notation of [11]) associated with the current filter I while traversing the tree.

2. Maintaining the *dual* pseudotriangulation of G (in the terminology of [11]) while traversing the tree, where an edge and its dual have the same color, allows us to retrieve e when coming back from the right subtree. Hence instead of storing a recursive stack to remember e on the way up the tree, the algorithm can maintain only G and its dual. This changes the space complexity of the algorithm from quadratic to linear.

3. If K is an acyclic and planar set of edges, then by coloring the edges of K red, and replacing the root of the tree with the horizontal greedy pseudotriangulation $G(I_{K,0}) \subseteq E_K$, the algorithm enumerates the set of pseudotriangulations that contain K. Observe that this proves that the subgraph of pseudotriangulations that contain K is connected.

3.4. The horizontal greedy triangulation. We now explain how to compute $G(I_{\emptyset,0})$. In fact, the algorithm can be adapted by a simple rotation to compute $G(I_\theta)$ for any $\theta \in [0, \pi)$. It is convenient for the exposition (and for the algorithm) to order the points of P by lexicographical order, i.e., $p_1 <_{yx} p_2 <_{yx} \cdots p_n$.

The construction uses *lower* and *upper horizon trees*, defined here as follows. For all point p_i with $1 \le i < n$, denote by $\ell(p)$ the point p_j, for $j > i$, which minimizes the angle $\Theta(p\vec{p}_j) \in [0, \pi)$. Define $\ell(p_n) = p_n$. Since $\ell^n(p) = p_n$, the set of edges of the form $p\ell(p)$ is a tree whose root is p_n (it is connected because every p_i has a path to p_n, and it has n vertices and $n-1$ edges). We call that tree the *lower horizon tree* and denote it by $T_\ell(P)$.

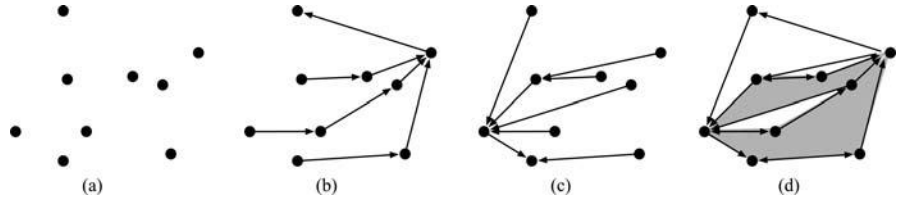


FIG. 6. (a) *shows point set,* (b) *its lower horizon tree and* (c) *its upper horizon tree;* (d) *shows that the superimposition of the horizon trees yields pseudoquadrangles (shaded areas) and pseudotriangles.*

COMPUTELOWERHORIZONTREE(P)	COMPUTEUPPERHORIZONTREE(P)
Effects: computes $\ell(p_i)$ for every $p_i \in P$	**Effects:** computes $u(p_i)$ for every $p_i \in P$
1: $\ell(p_n) \leftarrow p_n$	1: $u(p_1) \leftarrow p_1$
2: **for** $i \leftarrow n-1$ **downto** 1 **do**	2: **for** $i \leftarrow 2$ **to** n **do**
3: $j \leftarrow i+1$	3: $j \leftarrow i-1$
4: **while** `right_turn`$(p_i, p_j, \ell(p_j))$ **do**	4: **while** `right_turn`$(p_i, p_j, u(p_j))$ **do**
5: $j \leftarrow j+1$	5: $j \leftarrow j-1$
6: $\ell(p_i) \leftarrow p_j$	6: $u(p_i) \leftarrow p_j$

FIG. 7. *Computing the lower and upper horizon trees.*

Likewise, for point p_i with $1 < i \leq n$, denote by $u(p)$ the point p_j, $j < i$, which minimizes the angle $\Theta(\vec{pp}_j) \in [\pi, 2\pi)$ (define $u(p_1) = p_1$). The set of edges of the form $pu(p)$ is also a tree, of root p_1, which we call the *upper horizon tree* and denote by $T_u(P)$.

The following lemma, first observed in [43], forms the basis of the algorithm. See Figure 6 for an illustration.

LEMMA 11. *Let $K = T_\ell(P) \cup T_u(P)$ be the set of edges belonging to the horizon trees.*

1. *K contains all the edges of the convex hull of P.*

2. *K decomposes the convex hull of P into regions, each of which is either a pseudotriangle or a pseudoquadrangle.*

3. *$K \subseteq G(I_{\emptyset,0})$.*

With this lemma, the algorithm is straightforward. The pseudocode is given in Figure 7. It computes $\ell(p)$ for each $p \in P$ by Andrew's variant of Graham's convex hull algorithm [10]. We need the predicate `right_turn`(p, q, r) which returns true if the point sequence p, q, r forms a right turn. (In particular, the inner **while** loop will stop at $j = n$ since `right_turn`(p_i, p_n, p_n) is always false.)

Note that the algorithm still produces the correct tree if two edges are parallel, or even if two points have the same ordinate (thanks to the lexicographical order).

Computing $u(p)$ is performed by a similar algorithm. The initial sorting takes $O(n \log n)$ time. Once the horizon trees have been computed in $O(n)$ time, the subdivision they induce can be constructed in linear time, and each region can be visited to determine if it is a pseudotriangle or pseudoquadrangle. A pseudoquadrangle can be split in time linearly proportional to its number of edges by computing its two diagonals and inserting the one with smaller Θ. Thus, once the points are sorted lexicographically, the algorithm computes $G(I_{\emptyset,0})$ in linear time.

3.5. Complexity analysis. The algorithm spends $O(n)$ time for a flip or a priority queue operation in the worst case, and hence time $O(n)$ per edge of the tree.

Since the number of edges is the same as the number of internal nodes, which is also the number of leaves minus one, the algorithm spends amortized time $O(n)$ per pseudotriangulation.

Using a heap for the priority queue reduces the cost of the priority queue operations to $O(\log n)$. Moreover, using binary search can reduce the complexity of the flip algorithm to $O(\log n)$ as well, at the cost of maintaining the corners of pseudotriangles (which can be done in $O(1)$ time after a flip) and maintaining the boundary of the pseudotriangles as splittable queues as in [40].

Unfortunately, this is the time spent per leaf, counting both the n_r red *and* the n_b blue leaves. The following ratio is therefore important for analyzing the complexity of the algorithm: $\rho = (n_b + n_r)/n_r$. We initially conjectured a bound of 2 on this ratio, which was disproved by experiments (see next section). Currently, the best upper bound we have is the number of edges of a pseudotriangulation not on the convex hull, i.e., $2n - 3 - h$.

To conclude, the algorithm is set up in time $O(n \log n)$ to compute the horizontal greedy triangulation $G(I_{\emptyset,0})$ and insert its edges into the priority queue. The running time of the algorithm per red leaf of the tree (i.e., pseudotriangulation of P) is upper bounded by $O(\rho n) = O(n^2)$ and can be lowered with more complicated algorithmic machinery to $O(\rho \log n) = O(n \log n)$. All of this is in the worst case.

Note that the average complexity of a pseudotriangle is $O(1)$; thus on the average the flip will be performed in constant time. We expect that ρ is much smaller than n, although not constant ($\rho = O(\log n)$ seems a tempting conjecture, but we do not have a shred of evidence to support it). Thus, in practice we expect that the amortized cost per pseudotriangulation is much lower than $O(n \log n)$, perhaps $O(\rho)$. In order to state such a result, however, we lack what is needed, namely, an amortized bound for the flip (in the spirit of the analysis developed in [11]) and an upper bound for ρ.

Note finally that the number of pseudotriangulations grows exponentially fast (it is at least the Catalan number C_{n-2} for convex configurations [6]), thus limiting the domain of practicality of our algorithm in the low tens (resp., twenties). Thus, all the asymptotic complexities should be taken with a grain of salt. A good implementation will settle for low-complexity algorithms as well as simplicity of the code.

4. Implementation issues. Two independent implementations based on the above algorithm have been developed in order to ensure the correctness of the experimental validation of the conjecture.

4.1. Half-edge data structure. In both implementations, we chose to represent pseudotriangulation by a half-edge data structure (HDS), a.k.a. a doubly connected edge list (DCEL). One implementation is based on CGAL and described in [26, 21], and the other is based on an independent half-edge data structure described in [16].

An HDS is an edge-based data structure capable of storing a pseudotriangulation, or more generally, any connected planar set of edges. Each edge is split into two half-edges with opposite orientations. By convention, the half-edges incident to a face are oriented counterclockwise around the face. An `opposite` pointer links a half-edge to its opposite half-edge, and `next` and `prev` pointers links it to the next half-edge in counterclockwise orientation along the incident face. The incident vertices of a half-edge are named the `source` and `target`, as in [16].

4.2. Flip algorithm. Since the algorithm needs to examine the flip edge and decide whether to actually perform the flip or not, it is advantageous to implement

```
FindPseudoFlip(h)
Returns: a pair (h', g') such that the edge joining source(h') and source(g') is
the flip of the edge supporting h.
 1: g ← opposite(h)
 2: reverse_h←is_corner(h), reverse_g←is_corner(g)
 3: while true do
 4:   {decide which of g or h is the next tangent to jump to the next vertex}
 5:   if rotate_ccw_less(source(h), target(h), source(g), target(g))
                  is the same as (reverse_h≠reverse_g) then
 6:     {test if advancing g crosses over h and thus is the solution}
 7:     if left_turn(source(h), source(g), target(g))≠reverse_g then
 8:       return (h, g)
 9:     {not a solution yet, advance g}
10:     g ← next(g)
11:     if is_corner(g) then
12:       reverse_g←negate(reverse_g)
13:   else
14:     {test if advancing h crosses over g and thus is the solution}
15:     if left_turn(source(h), source(g), target(g))≠reverse_h then
16:       return (h, g)
17:     {not a solution yet, advance h}
18:     g ← next(h)
19:     if is_corner(h) then
20:       reverse_h←negate(reverse_h)
```

FIG. 8. *An implementation of the rotational flip method.*

the rotational sweep method. Moreover, in this case, neither the flip algorithm nor the enumeration algorithm needs the reverse `prev` pointers, thus saving space and execution time. For simplicity, we present below an implementation that uses `prev` pointers, namely, in the `is_corner` function. Eliminating `prev` pointers is possible (see the implementation in http://geometry.poly.edu/pstoolkit/) but complicates the pseudocode.

The function FindPseudoFlip returns two half-edge handles whose source vertices form the endpoints of the flipped diagonal rotated counterclockwise from the old diagonal. The function does not actually flip the diagonal. Note that the result could include the old diagonal, which needs to be considered before removing the old diagonal. In the function, h and g are the half-edges whose source vertices are in contact with the two rotating tangents. The two flags `reverse_h` and `reverse_g` indicate the relative orientation of the tangents to the half-edge h and g, respectively.

The pseudocode is presented in Figure 8. This function needs two geometric predicates: `left_turn`(p, q, r) returns true if the point sequence p, q, r forms a left turn, while `rotate_ccw_less`(p, q, r, s) returns true if the angle from the oriented segment pq to the oriented segment rs is less than π, which is equivalent to `left_turn`$(p, q, p + (s - r))$. As a convenience, `is_corner`(h) returns `left_turn(source(prev(`h`)), source(`h`), target(`h`))`.

We note that it is possible for two pseudotriangles to share more than the original edge h (but then it is easy to see that they cannot share more than two). In this

case, the reader can check that the algorithm does not miss the flip due to such (unavoidable) singularities.

An optimization we could have tried for the flip is to see if the two adjacent regions are triangles, which gives the diagonal without any geometric tests. (Note that because of the minimality of pseudotriangulations, the union of these two triangles must be a convex quadrilateral.) There is no guarantee, however, that even a single edge is adjacent to two triangles (consider, for instance, $p_1 = (0, -1)$, $p_2 = (0, 1)$, $p_k = (k, 0)$ for $k = 3, \ldots, n$). Nevertheless, if the point set has h edges on the boundary of its convex hull, there are at least $h - 2$ triangles in the pseudotriangulation (with equality iff all the pseudotriangles are at most quadrangular).

4.3. Enumeration algorithm. Using the FindPseudoFlip function, it is easy to implement the recursive variant of the enumeration algorithm. As noted, the only variable that we need to store in the recursion stack is the minimal edge e at the current node.

Since the number of pseudotriangulations of n points grows exponentially with n, we will not be able to run the algorithm for values of n larger than, say, 20. In fact, $n = 10$, with up to 234,160 pseudotriangulations, is already a challenge and takes on the order of a second. This dictates a few implementation choices.

First, the priority queue can be a simple vector of edges, sorted by Θ values, although using a binary heap is not a penalty and improves the performance slightly. Second, finding the flip without performing it saves a constant time. Third, a nonrecursive version of the algorithm eliminates the function call overhead, which contributes a (small) constant factor overall. Fourth, storing the old diagonal e on the stack when a diagonal is flipped avoids searching for this edge by reverse flipping the edge e' in order to restore the original pseudotriangulation. The second and fourth optimizations combined save 36% in runtime.

5. Experimental results. We started this investigation to support or find a counterexample to Conjecture 1 in section 1. The conjecture is not known to be true even for small values of n. We ran our enumeration algorithm on Aichholzer, Aurenhammer, and Krasser's comprehensive database of point sets with cardinality $n \leq 10$ [5] to obtain the number of pointed pseudotriangulations for every point set. The corresponding numbers triangulations are available on Aichholzer's Web page [2]. Our result is that for the over 14 million point sets S in the database up to $n \leq 10$, the number of triangulations of S is bounded by its number of pointed pseudotriangulations. Moreover, we also have computed the maximum number of pseudotriangulations (Table 5.1), which enriches Aichholzer's compendium. Finally, we have packaged our software into a pseudotriangulation workbench with which we can interactively examine pseudotriangulations, flip edges, and perform various algorithms (Figure 9). This was extremely useful in exploring other conjectures, including those concerning bounded-degree pseudotriangulations.

In order to elicit confidence in our implementations, we have independently devised two implementations of the enumeration algorithm and have checked that they agree on every point set in the database. The case $n = 10$ took about a month to compute on a cluster of 26 Sun workstations, and on another eight Pentium processors at 1 Ghz, and the independent computation by a coauthor took about 200 days. Luckily, both results agreed!

Interestingly, it was observed by Aichholzer that, for $n = 8$, the maximum number of pointed pseudotriangulations was achieved for the *same two* point sets as for the maximum number of triangulations, and he conjectured that the same would be true

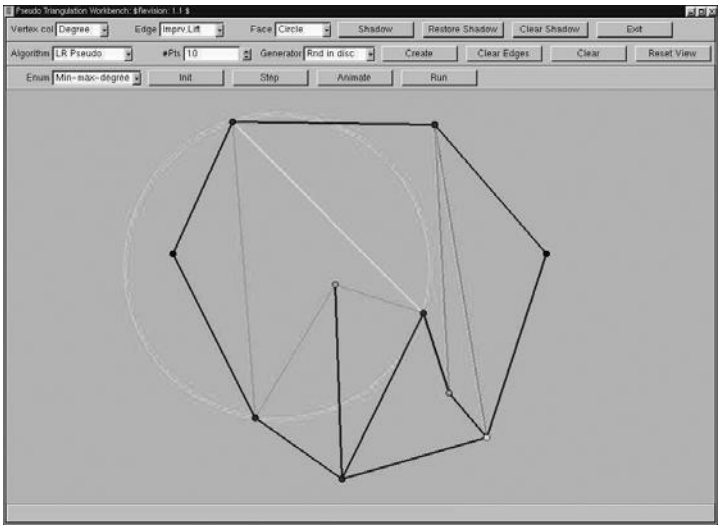


FIG. 9. *Screen shot of using the pseudotriangulation workbench to test a conjecture.*

TABLE 5.1

Number of pointed pseudotriangulations found among all the order types. In parentheses is the number of triangulations for the order type S maximizing its number of pointed pseudotriangulations, followed by the index of S in the database.

# points	# order types	Lower bound	Upper bound		Runtime
3	1	1	1	(1, #1)	< 1 sec
4	2	2	3	(1, #2)	< 1 sec
5	3	5	13	(2, #3)	< 1 sec
6	16	14	76	(8, #15)	< 1 sec
7	135	42	485	(30, #125)	1 sec
8	3 315	132	3 555	(150, #2991, and #3199)	3 min = 0.054 sec/order type
9	158 817	429	27 874	(774, #151 721)	990 min = 0.374 sec/order type
10	14 309 547	1430	234 160	(4550, #13 413 894 and #13 812 360)	About 200 days

for $n = 10$. Indeed, this is now verified. But this is not true for $n < 8$ or for $n = 9$. At this time, it seems far-fetched to conjecture that, for all even $n \geq 8$, the number of pointed pseudotriangulations attains its maximum exactly for the point sets which also maximize the number of triangulations. Nevertheless, we can state the following.

EXPERIMENTALLY PROVEN FACT 12. *For any n-point set with $n \leq 10$, its number of triangulations is less than its number of pointed pseudotriangulations, with equality iff the point set is in convex position. In that case, the number of pointed pseudotriangulations is minimal.*

Table 5.1 shows the minimum and maximum values of the number of pointed pseudotriangulations for every value of n and indicates the running time of our algorithms (both implementations were comparable). The point set with minimum number of pseudotriangulations is the point set in convex position for $n \leq 10$. This contrasts sharply with the situation on triangulations [2]. In a previous version of this paper, we conjectured that the number of pseudotriangulations is minimized for point sets in convex position for any value of n. In fact, this has been proven recently [6].

For $n = 11$, the point set database has recently been assembled by Aichholzer, Aurenhammer, and Krasser [5] and consists of 2,334,512,907 point sets. It is thus

TABLE 5.2

Maximum ratio ρ of (blue and red) leaves to red leaves during the enumeration. The second column is (a 5-digit approximation of) the exact number when available. The third column is a lower bound, obtained by trying random point sets with various distributions, while the fourth column is the best known upper bound.

# points	Exact	Lower bound	Upper bound
3	1		
4	2		
5	2.76923		4
6	3.49254		6
7	4.26786		8
8	4.89121		10
9	5.74258		12
10	6.28663		14
11	N/A	$\geq$ 6.11959	16
12	N/A	$\geq$ 5.709	18

infeasible to compute the exact lower and upper bounds using this algorithm. Nevertheless, we can still compute the number of pseudotriangulations for particular configurations (this is a further test of correctness of our algorithm when the result is know mathematically) and on random point sets.

We conjectured that the ratio ρ of (blue and red) leaves to red leaves was bounded by 2. Experiments showed that this is simply not true. In Table 5.2, we display some lower bounds on ρ, as well as the best known upper bound.

6. Conclusion. We have presented and implemented a new algorithm to enumerate all the pseudotriangulations of a point set. This algorithm uses the theory of pseudotriangulations that was developed for convex obstacles; in particular, it makes use of the greedy flip algorithm.

Using the polytopal construction of [47], one could obtain another algorithm via the reverse-search paradigm [12]. Our algorithm is more general, however, since with the proper flip algorithm it also applies to matroids (in the dual, it applies to arrangements of pseudolines).

The running time per pseudotriangulation is $O(n^2)$, although it should be possible not only to lower that upper bound by using amortization of the flip algorithm but also to obtain better upper bounds on the ratio of leaves over red leaves. Also, the algorithm can be improved in theory using fancier data structures, but since it is unlikely to be applied to point sets larger than 20, this is more of a theoretical exercise.

We independently developed two implementations of the algorithm, which agree on all point sets for $n \leq 10$. Using these implementations, we verified that Conjecture 1 is true for $n \leq 10$. The mathematical proof (even for such small values of n) is still waiting to be discovered.

Acknowledgments. This work was begun at a Bellairs workshop on pseudotriangulations organized by Ileana Streinu and partially supported by the NSF. The published results by the participants include the numbers of pseudotriangulations of special point configurations [45], the existence of pseudotriangulations with bounded degree [28, 27], and an analysis of the graph of pointed pseudotriangulations [47]. This latter work, especially quotes some of the results on flipping contained in this paper. We thank Ileana Streinu and the other participants of the NSF-supported Bellairs workshop on pseudotriangulations for many enjoyable discussions.

REFERENCES

[1] P. K. Agarwal, J. Basch, L. J. Guibas, J. Hershberger, and L. Zhang, *Deformable free space tiling for kinetic collision detection*, in New Directions in Algorithmic and Computational Robotics, B. Donald, K. Lynch, and D. Rus, eds., A. K. Peters, Wellesley, MA, 2001, pp. 83–96.

[2] O. Aichholzer, *Counting Triangulations—Olympics*, available online at http://www.cis.tugraz.at/igi/oaich/triangulations/counting/counting.html (2003).

[3] O. Aichholzer, *The path of a triangulation*, in Proc. 13th European Workshop on Computational Geometry, University of Würzburg, Germany, 1997, pp. 1–3.

[4] O. Aichholzer, F. Aurenhammer, C. Huemer, and H. Krasser, *Transforming spanning trees and pseudo-triangulations*, Inform. Process. Lett., 97 (2006), pp. 19–22.

[5] O. Aichholzer, F. Aurenhammer, and H. Krasser, *Enumerating order types for small point sets with applications*, Order, 19 (2002), pp. 265–281.

[6] O. Aichholzer, F. Aurenhammer, H. Krasser, and B. Speckmann, *Convexity minimizes pseudo-triangulations*, Comput. Geom. Theory Appl., 28 (2004), pp. 3–10.

[7] O. Aichholzer, F. Aurenhammer, H. Krasser, and P. Brass, *Pseudotriangulations from surfaces and a novel type of edge flip*, SIAM J. Comput., 32 (2003), pp. 1621–1653.

[8] O. Aichholzer, Th. Hackl, C. Huemer, F. Hurtado, H. Krasser, and B. Vogtenhuber, *On the number of plane graphs*, in Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, Philadelphia, 2006, pp. 504–513.

[9] O. Aichholzer, F. Hurtado, and M. Noy, *A lower bound on the number of triangulations of planar point sets*, Comput. Geom. Theory Appl., 29 (2004), pp. 135–145.

[10] A. M. Andrew, *Another efficient algorithm for convex hulls in two dimensions*, Inform. Process. Lett., 9 (1979), pp. 216–219.

[11] P. Angelier and M. Pocchiola, *A sum of squares theorem for visibility complexes and applications*, in Discrete and Computational Geometry—The Goodman–Pollack Festschrift, Algorithms Combin. 25, B. Aronov, S. Basu, J. Pach, and M. Sharir, eds., Springer-Verlag, Berlin, 2003, pp. 79–139.

[12] D. Avis and K. Fukuda, *Reverse search for enumeration*, Discrete Appl. Math., 65 (1996), pp. 21–46.

[13] S. Bereg, *Transforming pseudo-triangulation*, Inform. Process. Lett., 90 (2004), pp. 141–145.

[14] S. Bespamyatnikh, *An efficient algorithm for enumeration of triangulations*, Comput. Geom. Theory Appl., 23 (2002), pp. 271–279.

[15] A. Björner, M. Las Vergnas, N. White, B. Sturmfels, and G. M. Ziegler, *Oriented Matroids*, Cambridge University Press, Cambridge, UK, 1993.

[16] H. Brönnimann, *Designing and implementing a general purpose halfedge data structure*, in Proc. 5th International Workshop on Algorithm Engineering (WAE), Lecture Notes Comput. Sci. 2141, Springer, Berlin, 2001, pp. 51–66.

[17] B. Chazelle, H. Edelsbrunner, M. Grigni, L. J. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink, *Ray shooting in polygons using geodesic triangulations*, Algorithmica, 12 (1994), pp. 54–68.

[18] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, 2nd ed., Springer, Berlin, 2000.

[19] E. D. Demaine, J. S. B. Mitchell, and J. O'Rourke, eds, *Problem* 40: *The Number of Pointed Pseudotriangulations*, available online at http://maven.smith.edu/~orourke/TOPP/P40.html.

[20] M. Denny and C. Sohler, *Encoding a triangulation as a permutation of its point set*, in Proc. 9th Canadian Conference on Computataional Geometry, Kingston, Ontario, Canada, 1997, pp. 39–43.

[21] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr, *On the design of CGAL, a computational geometry algorithms library*, Softw.–Pract. Exp., 30 (2000), pp. 1167–1202.

[22] M. T. Goodrich and R. Tamassia, *Dynamic ray shooting and shortest paths in planar subdivisions via balanced geodesic triangulations*, J. Algorithms, 23 (1997), pp. 51–73.

[23] L. J. Guibas and J. Stolfi, *Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams*, ACM Trans. Graph., 4 (1985), pp. 74–123.

[24] J. Hershberger and J. Snoeyink, *Computing minimum length paths of a given homotopy class*, Comput. Geom. Theory Appl., 4 (1994), pp. 63–98.

[25] F. Hurtado, M. Noy, and J. Urrutia, *Flipping edges in triangulations*, Discrete Comput. Geom., 22 (1999), pp. 333–346.

[26] L. Kettner, *Using generic programming for designing a data structure for polyhedral surfaces*, Comput. Geom. Theory Appl., 13 (1999), pp. 65–90.

[27] L. Kettner, D. Kirkpatrick, A. Mantler, J. Snoeyink, B. Speckmann, and F. Takeuchi, *Tight degree bounds for pseudotriangulations of points*, Comput. Geom. Theory Appl., 25 (2003), pp. 1–12.

[28] L. Kettner, D. Kirkpatrick, and B. Speckmann, *Tight degree bounds for pseudo-triangulations of points*, in Proc. 13th Canadian Conference on Computational Geometry, Waterloo, Ontario, Canada, 2001, pp. 117–120.

[29] D. Kirkpatrick and J. Snoeyink, *Computing common tangents without a separating line*, in Proc. 4th Workshop on Algorithms and Data Structures (WADS), Lecture Notes Comput. Sci. 955, Springer, New York, 1995, pp. 183–193.

[30] D. Kirkpatrick, J. Snoeyink, and B. Speckmann, *Kinetic collision detection for simple polygons*, in Proc. 16th Annual ACM Symposium on Computing Geometry, ACM, New York, 2000, pp. 322–330.

[31] D. Kirkpatrick and B. Speckmann, *Separation sensitive kinetic separation for convex polygons*, in Proc. Japan Conference on Discrete and Computational Geometry (JCDCG 2000), Lecture Notes Comput. Sci. 2098, Springer, Berlin, 2001, pp. 244–251.

[32] D. E. Knuth, *Axioms and Hulls*, Lecture Notes Comput. Sci. 606, Springer, New York, 1992.

[33] C. L. Lawson, *Transforming triangulations*, Discrete Math., 3 (1972), pp. 365–372.

[34] D. T. Lee and F. P. Preparata, *Euclidean shortest paths in the presence of rectilinear barriers*, Networks, 14 (1984), pp. 393–410.

[35] Z. Li and S. Nakano, *Efficient generation of plane triangulations without repetition*, in Proc. 28th International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes Comput. Sci. 2076, Springer, Berlin, 2001, pp. 433–443.

[36] P. McMullen, *Modern developments in regular polytopes*, in Polytopes: Abstract, Convex and Computational, T. Bisztriczky, P. McMullen, R. Schneider, and A. Ivić Weiss, eds., NATO ASI Ser. 440, Kluwer Academic Publishers, Dordrecht, 1994, pp. 97–124.

[37] S. Negami, *Diagonal flips of triangulations on surfaces*, Yokohama Math. J., 47 (1999), pp. 1–44.

[38] M. H. Overmars and J. van Leeuwen, *Dynamically maintaining configurations in the plane*, in Proc. 12th Annual ACM Symposium on Theory of Computing, ACM, New York, 1980, pp. 135–145.

[39] M. Pocchiola and G. Vegter, *Minimal tangent visibility graphs*, Comput. Geom. Theory Appl., 6 (1996), pp. 303–314.

[40] M. Pocchiola and G. Vegter, *Topologically sweeping visibility complexes via pseudo-triangulations*, Discrete Comput. Geom., 16 (1996), pp. 419–453.

[41] M. Pocchiola and G. Vegter, *The visibility complex*, Internat. J. Comput. Geom. Appl., 6 (1996), pp. 279–308.

[42] M. Pocchiola and G. Vegter, *On polygonal covers*, in Advances in Discrete and Computational Geometry, B. Chazelle, J. Goodman, and R. Pollack, eds., Contemp. Math. 223, AMS, Providence, RI, 1999, pp. 257–268.

[43] M. Pocchiola, *Horizon trees versus pseudo-triangulations*, in Proc. 13th European Workshop on Computational Geometry, University of Würzburg, Germany, 1997, p. 12.

[44] M. Pocchiola and G. Vegter, *Pseudo-triangulations: Theory and applications*, in Proc. 12th Annual ACM Symposium on Computational Geometry, ACM, New York, 1996, pp. 291–300.

[45] D. Randall, G. Rote, F. Santos, and J. Snoeyink, *Counting triangulations and pseudo-triangulations of wheels*, in Proc. 13th Canadian Conference on Computational Geometry, Waterloo, Ontario, Canada, 2001, pp. 149–152.

[46] S. Ray and R. Seidel, *A Simple and Less Slow Method for Counting Triangulations and for Related Problems*, in Proc. European Workshop on Computational Geometry, Seville University, Spain, 2004, pp. 177–180.

[47] G. Rote, I. Streinu, and F. Santos, *Expansive motions and the polytope of pointed pseudo-triangulations*, in Discrete and Computational Geometry—The Goodman–Pollack Festschrift, Algorithms Combin. 25, B. Aronov, S. Basu, J. Pach, and M. Sharir, eds., Springer, Berlin, 2003, pp. 699–736.

[48] F. Santos and R. Seidel, *A better upper bound on the number of triangulations of a planar point set*, J. Combin. Theory Ser. A, 102 (2003), pp. 186–193.

[49] D. D. Sleator, R. E. Tarjan, and W. P. Thurston, *Rotation distance, triangulations, and hyperbolic geometry*, J. Amer. Math. Soc., 1 (1988), pp. 647–682.

[50] I. Streinu, *A combinatorial approach to planar non-colliding robot arm motion planning*, in Proc. 41st Annual IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 2000, pp. 443–453.

[51] W. T. Tutte, *A census of planar triangulation*, Canad. J. Math., 14 (1962), pp. 21–38.

SIAM J. COMPUT.
Vol. 36, No. 3, pp. 740–762

RANDOM k-SAT: TWO MOMENTS SUFFICE TO CROSS A SHARP THRESHOLD*

DIMITRIS ACHLIOPTAS† AND CRISTOPHER MOORE‡

Abstract. Many NP-complete constraint satisfaction problems appear to undergo a "phase transition" from solubility to insolubility when the constraint density passes through a critical threshold. In all such cases it is easy to derive upper bounds on the location of the threshold by showing that above a certain density the first moment (expectation) of the number of solutions tends to zero. We show that in the case of certain symmetric constraints, considering the second moment of the number of solutions yields nearly matching lower bounds for the location of the threshold. Specifically, we prove that the threshold for both random hypergraph 2-colorability (Property B) and random Not-All-Equal k-SAT is $2^{k-1}\ln 2 - O(1)$. As a corollary, we establish that the threshold for random k-SAT is of order $\Theta(2^k)$, resolving a long-standing open problem.

Key words. satisfiability, random formulas, phase transitions

AMS subject classifications. Primary, 68R99, 82B26; Secondary, 05C80

DOI. 10.1137/S0097539703434231

1. Introduction.

In the early 1900s, Bernstein [15] asked the following question: Given a collection of subsets of a set V, is there a partition of V into V_1, V_2 such that no subset is contained in either V_1 or V_2? If we think of the elements of V as vertices and of each subset as a hyperedge, the question can be rephrased as whether a given hypergraph can be 2-colored so that no hyperedge is monochromatic. Of particular interest is the setting where all hyperedges contain k vertices, i.e., k-uniform hypergraphs. This question was popularized by Erdős—who dubbed it "Property B" in honor of Bernstein—and has motivated some of the deepest advances in probabilistic combinatorics. Indeed, determining the smallest number of hyperedges in a non–2-colorable k-uniform hypergraph remains one of the most important problems in extremal graph theory, perhaps second only to the Ramsey problem [13].

A more modern problem, with a somewhat similar flavor, is Boolean Satisfiability: Given a Conjunctive Normal Form (CNF) formula F, is it possible to assign truth values to the variables of F so that it evaluates to true? Satisfiability has been the central problem of computational complexity since Cook [22] proved that it is complete for the class NP. The case where all clauses have the same size k is known as k-SAT and is NP-complete for all $k \geq 3$.

Random formulas and random hypergraphs have been studied extensively in probabilistic combinatorics in the last three decades. While there are a number of slightly different models for generating such structures "uniformly at random," we will see that results transfer readily between them. For the sake of concreteness, let $F_k(n,m)$ denote a formula chosen uniformly from among all $\binom{2^k\binom{n}{k}}{m}$ k-CNF formulas on n variables

*Received by the editors September 11, 2003; accepted for publication (in revised form) November 15, 2005; published electronically October 24, 2006.
http://www.siam.org/journals/sicomp/36-3/43423.html

†Department of Computer Science, University of California Santa Cruz, Santa Cruz, CA 95064 (optas@cs.ucsc.edu). This author's work was supported in part by the National Science Foundation CAREER award CCF-0546900. Part of this work was done while the author was with Microsoft Research.

‡Department of Computer Science, University of New Mexico, Albuquerque, NM 87131, and the Santa Fe Institute, Santa Fe, NM 87501 (moore@cs.unm.edu). This author's work was supported by the National Science Foundation under grants PHY-0200909, EIA-0218563, and CCR-0220070.

with m clauses. Similarly, let $H_k(n,m)$ denote a hypergraph chosen uniformly from among all $\binom{\binom{n}{k}}{m}$ k-uniform hypergraphs with n vertices and m hyperedges. We will say that a sequence of events $\mathcal{E}_n$ occurs *with high probability* (w.h.p.) if $\lim_{n\to\infty} \Pr[\mathcal{E}_n] = 1$ and *with uniformly positive probability* (w.u.p.p.) if $\liminf_{n\to\infty} \Pr[\mathcal{E}_n] > 0$. Throughout the paper, k will be arbitrarily large but fixed.

In recent years, random instances of both problems have been understood to undergo a "phase transition" as the ratio of constraints to variables passes through a critical threshold. That is, for a given number of vertices (variables), the probability that a random instance has a solution drops rapidly from 1 to 0 around a critical number of hyperedges (clauses). This sharp threshold phenomenon was discovered in the early 1990s, when several researchers [19, 49] performed computational experiments on $F_3(n, m = rn)$ and found that while for $r < 4.1$ almost all formulas are satisfiable, for $r > 4.3$ almost all are unsatisfiable. Moreover, as n increases, this transition narrows around $r \approx 4.2$. Along with similar results for other fixed $k \geq 3$ this has led to the following popular conjecture.

Satisfiability threshold conjecture. For each $k \geq 3$, there exists a constant r_k such that

$$\lim_{n\to\infty} \Pr[F_k(n, rn) \text{ is satisfiable}] = \begin{cases} 1 & \text{if } r < r_k, \\ 0 & \text{if } r > r_k. \end{cases}$$

In the last ten years, this conjecture has become an active area of interdisciplinary research, receiving attention in theoretical computer science, artificial intelligence, combinatorics, and, more recently, statistical physics. Much of the work on random k-SAT has focused on proving upper and lower bounds for r_k, both for the smallest computationally hard case $k = 3$ and for general k. At this point the existence of r_k has not been established for any $k \geq 3$. Nevertheless, we will take the liberty of writing $r_k \geq r^*$ to denote that for all $r < r^*$, $F_k(n, rn)$ is w.h.p. satisfiable; analogously, we will write $r_k \leq r^*$ to denote that for all $r > r^*$, $F_k(n, rn)$ is w.h.p. unsatisfiable.

As we will see, an elementary counting argument yields $r_k \leq 2^k \ln 2$ for all k. Lower bounds, on the other hand, have been exclusively algorithmic: To establish $r_k \geq r^*$ one shows that for $r < r^*$ some specific algorithm finds a satisfying assignment with probability that tends to 1. We will see that an extremely simple algorithm [20] already yields $r_k = \Omega(2^k/k)$. We will also see that while more sophisticated algorithms improve this bound slightly, to date no algorithm is known to find a satisfying truth assignment (even w.u.p.p.) when $r = \omega(k) \times 2^k/k$ for any $\omega(k) \to \infty$.

The threshold picture for hypergraph 2-colorability is completely analogous: For each $k \geq 3$, it is conjectured that there exists a constant c_k such that

$$\lim_{n\to\infty} \Pr[H_k(n, cn) \text{ is 2-colorable}] = \begin{cases} 1 & \text{if } c < c_k, \\ 0 & \text{if } c > c_k. \end{cases}$$

The same counting argument here implies $c_k < 2^{k-1} \ln 2$, while another simple algorithm yields $c_k = \Omega(2^k/k)$. Again, no algorithm is known to improve this bound asymptotically, leaving a multiplicative gap of order $\Theta(k)$ between the upper and lower bounds for this problem as well.

In this paper, we use the *second moment method* to show that random k-CNF formulas are satisfiable and random k-uniform hypergraphs are 2-colorable for density up to $2^{k-1} \ln 2 - O(1)$. Thus, we determine the threshold for random k-SAT within a factor of two and the threshold for Property B within a small additive constant.

Recall that $F_k(n, rn)$ is w.h.p. unsatisfiable if $r > 2^k \ln 2$. Our first main result is the following theorem.

THEOREM 1. *For all $k \geq 3$, $F_k(n, m = rn)$ is w.h.p. satisfiable if*

$$r \leq 2^{k-1} \ln 2 - 2.$$

Our second main result determines the Property B threshold within an additive $1/2 + o(1)$.

THEOREM 2. *For all $k \geq 3$, $H_k(n, m = cn)$ is w.h.p. non–2-colorable if*

$$c > 2^{k-1} \ln 2 - \frac{\ln 2}{2}. \tag{1}$$

There exists a sequence $t_k \to 0$ such that for all $k \geq 3$, $H_k(n, m = cn)$ is w.h.p. 2-colorable if

$$c < 2^{k-1} \ln 2 - \frac{\ln 2}{2} - \frac{1 + t_k}{2}. \tag{2}$$

The bound in (1) corresponds to the density for which the expected number of 2-colorings of $H_k(n, cn)$ is $o(1)$. Our main contribution is inequality (2), which we prove using the second moment method. In fact, our approach yields explicit lower bounds for the hypergraph 2-colorability threshold for each value of k (although these bounds lack an attractive closed form). We give the first few of these bounds in Table 1. We see that the gap between our upper and lower bounds converges to its limiting value of $1/2$ rather rapidly.

TABLE 1
Bounds for the 2-colorability threshold of random k-uniform hypergraphs.

k	3	4	5	6	7	8	9	10
Upper bound	2.410	5.191	10.741	21.833	44.014	88.376	177.099	354.545
Lower bound	1.5	4.083	9.973	21.190	43.432	87.827	176.570	354.027

Unlike the bounds for random k-SAT and hypergraph 2-colorability provided by analyzing algorithms, our arguments are nonconstructive: We establish that w.h.p. solutions exist for certain densities but do not offer any hint on how to find them. We believe that abandoning the algorithmic approach for proving such lower bounds is natural and, perhaps, necessary. At a minimum, the algorithmic approach is limited to the small set of rather naive algorithms whose analysis is tractable using current techniques. Perhaps more gravely, it could be that *no* polynomial-time algorithm can overcome the $\Theta(2^k/k)$ barrier. Determining whether this is true even for certain limited classes of algorithms, e.g., random walk algorithms, is a very interesting open problem.

In addition, by not seeking out some specific truth assignment, as algorithms do, the second moment method gives some first glimpses of the "geometry" of the set of solutions. Deciphering these first glimpses, getting clearer ones, and exploring potential interactions between the geometry of the set of solutions and computational hardness are great challenges that lie ahead.

We note that recently, and independently, Frieze and Wormald [34] applied the second moment method to random k-SAT in the case where k is a moderately growing

function of n. Specifically, they proved that when $k - \log_2 n \to \infty$, $F_k(n, m)$ is w.h.p. satisfiable if $m < (1 - \epsilon)m^*$ but is w.h.p. unsatisfiable if $m > (1 + \epsilon)m^*$, where $m^* = (2^k \ln 2 - O(1))\, n$ and $\epsilon = \epsilon(n) > 0$ is such that $\epsilon n \to \infty$. Their result follows by a direct application of the second moment method to the number of satisfying assignments of $F_k(n, m)$. As we will see shortly, while this approach gives a very sharp bound when $k - \log_2 n \to \infty$, it fails for any fixed k and indeed for any $k = o(\log n)$.

We also note that since this work first appeared [4, 5], the line of attack we put forward has had several other successful applications. Specifically, in [7], the lower bound for the random k-SAT threshold was improved to $2^k \ln 2 - O(k)$ by building on the insights presented here. In [8], the method was successfully extended to random Max k-SAT, while in [9, 10] it was applied to random graph coloring. We discuss these subsequent developments in the conclusions.

1.1. The second moment method and the role of symmetry. The version of the second moment method we will use is given by Lemma 1 and follows from a direct application of the Cauchy–Schwarz inequality (see Remark 3.1 in [38]).

LEMMA 1. *For any nonnegative random variable X,*

$$\Pr[X > 0] \geq \frac{\mathbf{E}[X]^2}{\mathbf{E}[X^2]}. \tag{3}$$

It is natural to try to apply Lemma 1 to random k-SAT by letting X be the number of satisfying truth assignments of $F_k(n, m)$. Unfortunately, as we will see, this "naive" application of the second moment method fails rather dramatically: For all $k \geq 1$ and every $r > 0$, $\mathbf{E}[X^2] > (1 + \beta)^n\, \mathbf{E}[X]^2$ for some $\beta = \beta(k, r) > 0$. As a result, the second moment method gives only an exponentially small lower bound on the probability of satisfiability.

The key step in overcoming this failure lies in realizing that we are free to apply the second moment method to any random variable X such that $X > 0$ implies that the formula is satisfiable. In particular, we can let X be the size of any *subset* of the set of satisfying assignments. By choosing this subset carefully, we can hope to significantly reduce the variance of X relative to its expectation and use Lemma 1 to prove that the subset is frequently nonempty. Indeed, we will establish the satisfiability of random k-CNF by focusing on those satisfying truth assignments *whose complement is also satisfying*. In section 3 we will give some intuition for why the number of such assignments has much smaller variance than the number of all satisfying assignments. For now, we observe that considering only such satisfying assignments is equivalent to interpreting the random k-CNF formula $F_k(n, m)$ as an instance of Not-All-Equal (NAE) k-SAT, where a truth assignment σ is a solution if and only if under σ every clause contains at least one satisfied literal *and* at least one unsatisfied literal. In other words, our lower bound for the k-SAT threshold in Theorem 1 is, in fact, a lower bound for the NAE k-SAT threshold.

Indeed, for both random NAE k-SAT and random hypergraph 2-colorability we will apply Lemma 1 naively, i.e., by letting X be the number of solutions. This will give Theorem 2 and the values in Table 1 for hypergraph 2-colorability and, as we will see, exactly the same bounds for random NAE k-SAT. (The proof of Theorem 2 is a slight generalization of the proof for random NAE k-SAT.) We will see that this success of the naive second moment is due to the symmetry inherent in both problems, i.e., to the fact that the complement of a solution is also a solution. We feel that highlighting this role of symmetry—and showing how it can be exploited even in asymmetric problems like k-SAT—is our main conceptual contribution. Exploiting

these ideas in other constraint satisfaction problems that have a permutation group acting on the variables' domain is an interesting area for further research.

1.2. Organization of the paper. In section 2 we give some background on random k-SAT and random hypergraph 2-colorability. In section 3 we explain why the second moment method fails when applied to k-SAT directly, and give some intuition for why counting only the NAE-satisfying assignments rectifies the problem. We also point out some connections to methods of statistical physics. In section 4 we lay the groundwork for bounding the second moment for both NAE k-SAT and hypergraph 2-colorability by dealing with some probabilistic preliminaries, introducing a "Laplace method" lemma for bounding certain sums, and outlining our strategy. The actual bounding occurs in sections 5 to 7. Specifically, in sections 5 and 6 we use the Laplace lemma to reduce the second moment calculations for both random NAE k-SAT and random hypergraph 2-colorability to the maximization of a certain function g on the unit interval, where g is independent of n. We maximize g in section 7 and prove the Laplace lemma in section 8. We conclude in section 9 by discussing some recent extensions of this work and proposing several open questions.

2. Related work.

2.1. Random k-SAT. The mathematical investigation of random k-SAT began with the work of Franco and Paull [31], who, among other results, observed that $F_k(n, m = rn)$ is w.h.p. unsatisfiable if $r \geq 2^k \ln 2$. To see this, let $C_k = 2^k \binom{n}{k}$ be the number of all possible k-clauses and let $S_k = (2^k - 1)\binom{n}{k}$ be the number of k-clauses consistent with a given truth assignment. Since any fixed truth assignment is satisfying with probability $\binom{S_k}{m}/\binom{C_k}{m} < (1-2^{-k})^m$, the expected number of satisfying truth assignments of $F_k(n, m = rn)$ is at most $[2(1-2^{-k})^r]^n = o(1)$ for $r \geq 2^k \ln 2$.

Shortly afterwards, Chao and Franco [18] complemented this result by proving that for all $k \geq 3$, if $r < 2^k/k$, then the following linear-time algorithm, called Unit Clause (uc), finds a satisfying truth assignment w.u.p.p.: If there exist unit clauses, pick one randomly and satisfy it; else pick a random unset variable and give it a random value. Note that since uc succeeds only w.u.p.p. (rather than w.h.p.) this does not imply a lower bound for r_k.

The satisfiability threshold conjecture gained a great deal of popularity in the early 1990s and has received an increasing amount of attention since then. The polynomial-time solvable case $k = 2$ was settled early on: Independently, Chvátal and Reed [20], Fernandez de la Vega [29], and Goerdt [35] proved that $r_2 = 1$. Chvátal and Reed [20], in addition to proving $r_2 = 1$, gave the first lower bound for r_k, strengthening the positive probability result of Chao and Franco [18] by analyzing the following refinement of uc, called Short Clause (sc): If there exist unit clauses, pick one randomly and satisfy it; else if there exist binary clauses, pick one randomly and satisfy a random literal in it; else pick a random unset variable and give it a random value. In [20], the authors showed that for all $k \geq 3$, sc finds a satisfying truth assignment w.h.p. for $r < (3/8)\, 2^k/k$ and raised the question of whether this lower bound for r_k can be improved asymptotically.

A large portion of the work on the satisfiability threshold conjecture since then has been devoted to the first computationally hard case, $k = 3$, and a long series of results [16, 17, 33, 1, 11, 36, 41, 25, 42, 39, 24, 44, 40, 26, 31] has narrowed the potential range of r_3. Currently this is pinned between 3.52 by Kaporis, Kirousis, and Lalas [41] and Hajiaghayi and Sorkin [36] and 4.506 by Dubois, Boufkhad, and Mandler [25]. Upper bounds for r_3 come from probabilistic counting arguments, refining the above

calculation of the expected number of satisfying assignments. Lower bounds, on the other hand, have come from analyzing progressively more sophisticated algorithms. Unfortunately, neither of these approaches helps narrow the asymptotic gap between the upper and lower bounds for r_k. The upper bounds improve $r_k \le 2^k \ln 2$ by only a small additive constant; the best algorithmic lower bound, due to Frieze and Suen [33], is $r_k \ge a_k 2^k/k$, where $\lim_{k\to\infty} a_k = 1.817\ldots$.

Two more results stand out in the study of random k-CNF formulas. In a breakthrough paper, Friedgut [32] proved the existence of a *nonuniform* satisfiability threshold, i.e., of a sequence $r_k(n)$ around which the probability of satisfiability goes from 1 to 0.

THEOREM 3 ([32]). *For each $k \ge 2$, there exists a sequence $r_k(n)$ such that for every $\epsilon > 0$,*

$$\lim_{n\to\infty} \Pr[F_k(n, rn) \text{ is satisfiable}] = \begin{cases} 1 & \text{if } r = (1-\epsilon)\, r_k(n), \\ 0 & \text{if } r = (1+\epsilon)\, r_k(n). \end{cases}$$

In [21], Chvátal and Szemerédi established a seminal result in proof complexity, by extending the work of Haken [37] and Urquhart [53] to random formulas. Specifically, they proved that for all $k \ge 3$, if $r \ge 2^k \ln 2$, then w.h.p. $f_k rn$ is unsatisfiable, but every resolution proof of its unsatisfiability contains at least $(1+\epsilon)^n$ clauses for some $\epsilon = \epsilon(k, r) > 0$. In [2], Achlioptas, Beame, and Molloy extended the main result of [21] to random CNF formulas that also contain 2-clauses, as this is relevant for the behavior of Davis–Putnam–Logemann–Loveland (DPLL) algorithms on random k-CNF. (DPLL algorithms proceed by setting variables sequentially, according to some heuristic, and backtracking whenever a contradiction is reached.) By combining the results in the present paper with the results in [2], it was recently shown [3] that a number of DPLL algorithms require exponential time *significantly below* the satisfiability threshold, i.e., for provably satisfiable random k-CNF formulas.

Finally, we note that if one chooses to live unencumbered by the burden of mathematical proof, powerful nonrigorous techniques of statistical physics, such as the "replica method," become available. Indeed, several claims based on the replica method have been subsequently established rigorously; thus it is frequently (but definitely not always) correct. Using this technique, Monasson and Zecchina [50] predicted $r_k \simeq 2^k \ln 2$. Like most arguments based on the replica method, their argument is mathematically sophisticated but far from rigorous. In particular, they argue that as k grows large, the so-called *annealed approximation* should apply. This creates an analogy with the second moment method which we discuss in section 3.4.

2.2. Random hypergraph 2-colorability. While Bernstein originally raised the 2-colorability question for certain classes of infinite set families [15], Erdős popularized the finite version of the problem [14, 27, 28, 43, 45, 51, 52] and the hypergraph representation. Recall that a 2-uniform hypergraph, i.e., a graph, is 2-colorable if and only if it has no odd cycle. In a random graph with cn edges this occurs with constant probability if and only if $c < 1/2$ (see [30] for more on the evolution of cycles in random graphs).

For all $k \ge 3$, on the other hand, hypergraph 2-colorability is NP-complete [46], and determining the 2-colorability threshold c_k for k-uniform hypergraphs $H_k(n, cn)$ remains open. Analogously to random k-SAT, we will take the liberty of writing $c_k \ge c^*$ if $H_k(n, cn)$ is w.h.p. 2-colorable for all $c < c^*$, and $c_k \le c^*$ if $H_k(n, cn)$ is w.h.p. non–2-colorable for all $c > c^*$.

Alon and Spencer [12] were the first to give bounds on the potential value of c_k. Specifically, they observed that, analogously to random k-SAT, the expected number of 2-colorings of $H_k(n, cn)$ is at most $[2(1-2^{1-k})^c]^n$ and concluded that $H_k(n, cn)$ is w.h.p. non–k-colorable if $c \geq 2^{k-1} \ln 2$. More importantly, by employing the Lovász local lemma, they proved that $H_k(n, cn)$ is w.h.p. 2-colorable if $c = O(2^k/k^2)$. Regarding the upper bound, it is easy to see that, in fact, $2(1-2^{1-k})^c < 1$ if $c = 2^{k-1} \ln 2 - (\ln 2)/2$, and this yields the upper bound of Theorem 2. Moreover, the techniques of [44, 24] can be used to improve this bound further to $2^{k-1} \ln 2 - (\ln 2)/2 - 1/4 + t_k$, where $t_k \to 0$.

The lower bound of [12] was improved by Achlioptas et al. [6] motivated by the analogies drawn in [12] between hypergraph 2-colorability and earlier work [18, 20] for random k-SAT. Specifically, it was shown in [6] that a simple, linear-time algorithm w.h.p. finds a 2-coloring of $H_k(n, cn)$ for $c = O(2^k/k)$, implying $c_k = \Omega(2^k/k)$. These were the best bounds for c_k prior to Theorem 2 of the present paper.

Finally, we note that Friedgut's result [32] applies to hypergraph 2-colorability as well, as presented in the following theorem.

THEOREM 4 ([32]). *For each $k \geq 3$, there exists a sequence $c_k(n)$ such that for every $\epsilon > 0$,*

$$\lim_{n\to\infty} \Pr[H_k(n, cn) \text{ is 2-colorable}] = \begin{cases} 1 & \text{if } c = (1-\epsilon)\, c_k(n), \\ 0 & \text{if } c = (1+\epsilon)\, c_k(n). \end{cases}$$

3. The second moment method: First look. In the rest of the paper it will be convenient to work with a model of random formulas that differs slightly from $F_k(n, m)$. Specifically, to generate a random k-CNF formula on n variables with m clauses we simply generate a string of km independent random literals, each such literal being drawn uniformly from among all $2n$ possible ones. Note that this is equivalent to selecting, with replacement, m clauses from among all possible $2^k n^k$ ordered k-clauses. This choice of distribution for k-CNF formulas will simplify our calculations significantly. As we will see in section 4.1, the derived results can be easily transferred to all other standard models for random k-CNF formulas.

3.1. Random *k*-SAT. For any formula F, given truth assignments $\sigma_1, \sigma_2, \ldots \in \{0,1\}^n$, we will write $\sigma_1, \sigma_2, \ldots \models F$ to denote that *all* of $\sigma_1, \sigma_2, \ldots$ satisfy F. Let $X = X(F)$ denote the number of satisfying assignments of a formula F. Then, for a k-CNF formula with random clauses $c_1, c_2, \ldots, c_m$ we have

(4)
$$\mathbf{E}[X] = \mathbf{E}\left[\sum_{\sigma} \mathbf{1}_{\sigma\models F}\right] = \sum_{\sigma} \mathbf{E}\left[\prod_{c_i} \mathbf{1}_{\sigma\models c_i}\right] = \sum_{\sigma}\prod_{c_i} \mathbf{E}[\mathbf{1}_{\sigma\models c_i}] = 2^n(1-2^{-k})^m,$$

since clauses are drawn independently and the probability that σ satisfies the ith random clause, i.e., $\mathbf{E}[\mathbf{1}_{\sigma\models c_i}]$, is $1-2^{-k}$ for every σ and i. Similarly, for $\mathbf{E}[X^2]$ we have

$$\mathbf{E}[X^2] = \mathbf{E}\left[\left(\sum_{\sigma} \mathbf{1}_{\sigma\models F}\right)^2\right] = \mathbf{E}\left[\sum_{\sigma,\tau} \mathbf{1}_{\sigma,\tau\models F}\right] = \sum_{\sigma,\tau}\prod_{c_i} \mathbf{E}[\mathbf{1}_{\sigma,\tau\models c_i}]. \tag{5}$$

We claim that $\mathbf{E}[\mathbf{1}_{\sigma,\tau\models c_i}]$, i.e., the probability that a fixed pair of truth assignments σ, τ satisfy the ith random clause, depends only on the number of variables z to which

σ and τ assign the same value. Specifically, if the overlap is $z = \alpha n$, we claim that this probability is

$$f_S(\alpha) = 1 - 2^{1-k} + 2^{-k}\alpha^k. \tag{6}$$

Our claim follows by inclusion-exclusion and observing that if c_i is not satisfied by σ, the only way for it to also not be satisfied by τ is for all k variables in c_i to lie in the overlap of σ and τ. Thus, f_S quantifies the correlation between the events that σ and τ are satisfying as a function of their overlap. In particular, observe that truth assignments with overlap $n/2$ are uncorrelated since $f_S(1/2) = (1 - 2^{-k})^2 = \Pr[\sigma \text{ is satisfying}]^2$.

Since the number of ordered pairs of assignments with overlap z is $2^n \binom{n}{z}$, we thus have

$$\mathbf{E}[X^2] = 2^n \sum_{z=0}^{n} \binom{n}{z} f_S(z/n)^m. \tag{7}$$

Writing $z = \alpha n$ and using the approximation $\binom{n}{z} = (\alpha^\alpha (1-\alpha)^{1-\alpha})^{-n} \times \text{poly}(n)$, we see that

$$\begin{aligned}\mathbf{E}[X^2] &= 2^n \left(\max_{0 \le \alpha \le 1} \left[\frac{f_S(\alpha)^r}{\alpha^\alpha (1-\alpha)^{1-\alpha}} \right] \right)^n \times \text{poly}(n) \\ &\equiv \left(\max_{0 \le \alpha \le 1} \Lambda_S(\alpha) \right)^n \times \text{poly}(n).\end{aligned}$$

At the same time observe that $\mathbf{E}[X]^2 = \left(2^n (1 - 2^{-k})^{rn}\right)^2 = (4 f_S(1/2)^r)^n = \Lambda_S(1/2)^n$. Therefore, if there exists some $\alpha \in [0,1]$ such that $\Lambda_S(\alpha) > \Lambda_S(1/2)$, then the second moment is exponentially greater than the square of the expectation and we get only an exponentially small lower bound for $\Pr[X > 0]$. Put differently, unless the dominant contribution to $\mathbf{E}[X^2]$ comes from "uncorrelated" pairs of satisfying assignments, i.e., pairs with overlap $n/2$, the second moment method fails.

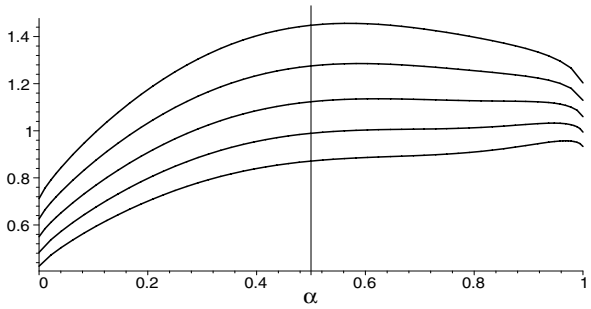
With these observations in mind, in Figure 1 we plot $\Lambda_S(\alpha)$ for $k = 5$ and different values of r. We see that, unfortunately, for all values of r shown, Λ_S is maximized at some $\alpha > 1/2$. If we look closely into the two factors comprising Λ_S, the reason for the failure of the second moment method becomes apparent: While the entropic factor $\left(\alpha^\alpha (1-\alpha)^{1-\alpha}\right)^{-1}$ is symmetric around $1/2$, the correlation function f_S is strictly increasing in $[0,1]$. Therefore, the derivative of Λ_S is never 0 at $1/2$, instead becoming 0 at some $\alpha > 1/2$ where the benefit of positive correlation balances with the cost of decreased entropy. (Indeed, this is true for all $k = o(\log n)$ and constant $r > 0$.)

3.2. Random NAE k-SAT. Let us now repeat the above analysis but with $X = X(F)$ being the number of NAE-satisfying truth assignments of a formula F. Recall that σ is an NAE-satisfying assignment if and only if under σ every clause has at least one satisfied literal *and* at least one unsatisfied literal. Thus, for a k-CNF formula with random clauses $c_1, c_2, \ldots, c_m$, proceeding as in (4), we get

$$\mathbf{E}[X] = 2^n (1 - 2^{1-k})^m, \tag{8}$$

since the probability that σ NAE-satisfies the ith random clause is $1 - 2^{1-k}$ for every σ and i.

Regarding the second moment, proceeding exactly as in (5), we write $\mathbf{E}[X^2]$ as a sum over the 4^n ordered pairs of assignments of the probability that both assignments



$k = 5$, $r = 16, 18, 20, 22, 24$ (top to bottom)

FIG. 1. *The nth root of the expected number of pairs of satisfying assignments at distance αn.*

are NAE-satisfying. As for k-SAT, for any fixed pair this probability depends only on the overlap. The only change is that if σ, τ agree on $z = \alpha n$ variables, then the probability that they both NAE-satisfy a random clause c_i is

$$\begin{aligned} \Pr[\sigma \text{ and } \tau \text{ NAE-satisfy } c_i] &= 1 - 2^{2-k} + 2^{1-k}\left(\alpha^k + (1-\alpha)^k\right) \\ &\equiv f_N(\alpha). \end{aligned} \tag{9}$$

Again, this claim follows from inclusion-exclusion and observing that for both σ, τ to NAE-violate c_i, the variables of c_i must either all be in the overlap of σ and τ or all be in their nonoverlap.

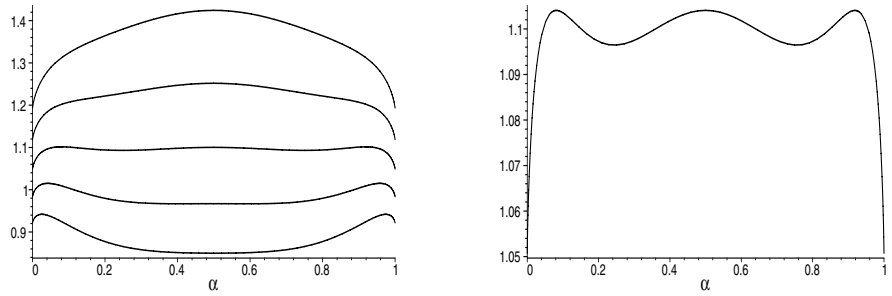
Applying Stirling's approximation for the factorial again and observing that the sum defining $\mathbf{E}[X^2]$ has only a polynomial number of terms, we now get (analogously to Λ_S in random k-SAT)

$$\begin{aligned} \mathbf{E}[X^2] &= 2^n \left(\max_{0 \le \alpha \le 1} \left[\frac{f_N(\alpha)^r}{\alpha^\alpha (1-\alpha)^{1-\alpha}} \right] \right)^n \times \text{poly}(n) \\ &\equiv \left(\max_{0 \le \alpha \le 1} \Lambda_N(\alpha) \right)^n \times \text{poly}(n). \end{aligned} \tag{10}$$

As before, it is easy to see that $\mathbf{E}[X]^2 = \Lambda_N(1/2)^n$. Therefore, if $\Lambda_N(1/2) > \Lambda_N(\alpha)$ for every $\alpha \neq 1/2$, then (10) implies that the ratio between $\mathbf{E}[X^2]$ and $\mathbf{E}[X]^2$ is at most polynomial in n. Indeed, with a more careful analysis of the interplay between the summation and Stirling's approximation, we will later show that whenever $\Lambda_N(1/2)$ is a global maximum, the ratio $\mathbf{E}[X^2]/\mathbf{E}[X]^2$ is bounded by a constant, implying that NAE-satisfiability holds w.u.p.p. So, all in all, again we hope that the dominant contribution to $\mathbf{E}[X^2]$ comes from pairs of assignments with overlap $n/2$.

The crucial difference is that now the correlation function f_N is *symmetric* around $1/2$ and, hence, so is Λ_N. As a result, the entropy-correlation product Λ_N always has a local extremum at $1/2$. Moreover, since the entropic term is always maximized at $\alpha = 1/2$ and is independent of r, for sufficiently small r this extremum is a global maximum. With these considerations in mind, in Figure 2 we plot $\Lambda_N(\alpha)$ for $k = 5$ and various values of r.

Let us start with the picture on the left, where r increases from 8 to 12 as we go from top to bottom. For $r = 8, 9$ we see that indeed Λ_N has a global maximum at $1/2$ and the second moment method succeeds. For the cases $r = 11, 12$, on the other hand, we see that $\Lambda_N(1/2)$ is actually a global minimum. In fact, we see that $\Lambda_N(1/2) < 1$,



$k = 5$, $r = 8, 9, 10, 11, 12$ (top to bottom) $k = 5$, $r = 9.973$

FIG. 2. *The nth root of the expected number of pairs of NAE-assignments at distance αn.*

implying that $\mathbf{E}[X]^2 = \Lambda_N(1/2)^n = o(1)$ and so w.h.p. there are no NAE-satisfying assignments for such r. It is worth noting that for $r = 11$, even though $X = 0$ w.h.p., the second moment is exponentially large (since $\Lambda_N > 1$ near 0 and 1).

The most interesting case is $r = 10$. Here $\Lambda(1/2) = 1.0023\ldots$ is a local maximum and greater than 1, but the two global maxima occur at $\alpha = 0.08\ldots$ and $\alpha = 0.92\ldots$, where the function equals $1.0145\ldots$. As a result, again, the second moment method gives only an exponentially small lower bound on $\Pr[X > 0]$. Note that this is in spite of the fact that $\mathbf{E}[X]$ is now exponentially large. Indeed, the largest value for which the second moment succeeds for $k = 5$ is $r = 9.973\ldots$ when the two side peaks reach the same height as the peak at $1/2$ (see the plot on the right in Figure 2).

So, the situation can be summarized as follows. By requiring that we count only NAE-satisfying truth assignments, we make it roughly twice as hard to satisfy each clause. This manifests itself in the additional factor of 2 in the middle term of f_N compared to f_S. On the other hand, now, the third term of f, capturing "joint" behavior, is symmetric around $1/2$, making Λ itself symmetric around $1/2$. This enables the second moment method which, indeed, breaks down only when the density gets within an additive constant of the upper bound for the NAE k-SAT threshold.

3.3. How symmetry reduces variance. Given a truth assignment σ and an arbitrary CNF formula F, let $Q = Q(\sigma, F)$ denote the total number of literal *occurrences* in F satisfied by σ. With this definition at hand, a potential explanation of how symmetry reduces the variance is suggested by considering the following trivial refinement of our generative model: First (i) draw km literals uniformly and independently just as before and then (ii) partition the drawn literals randomly into k-clauses (rather than assuming that the first k literals form the first clause, the next k the second, etc.).

In particular, imagine that we have just finished performing the first generative step above and we are about to perform the second. Observe that at this point the value of Q has already been determined for every $\sigma \in \{0,1\}^n$. Moreover, for each fixed σ the conditional probability of yielding a satisfying assignment corresponds to a balls-in-bins experiment: Distribute $Q(\sigma)$ balls in m bins, each with capacity k, so that every bin receives at least one ball. It is clear that those truth assignments for which Q is large at the end of the first step have a big advantage in the second.

To get an idea of what Q typically looks like on $\{0,1\}^n$ we begin by observing that the number of occurrences of a fixed literal ℓ, B_ℓ, is distributed as $\mathrm{Bin}(km, 1/(2n))$.

Thus, $\mathbf{E}[B_\ell] = O(1)$ and, moreover, the random variables B_ℓ are very weakly correlated. In particular, Q takes its maximum value on the subcube of truth assignments where every variable is assigned its majority value and, typically, decreases gradually away from there. Thus, at the end of the first step the "more promising" truth assignments are highly correlated: In satisfying many literal occurrences (thus increasing their odds for the second step), they tend to overlap with each other (and the majority assignment) at more than half the variables.

In contrast, if we focus on NAE-satisfying assignments, at the end of the first step the most promising assignments σ are those for which $Q(\sigma)$ is very close to its average value $km/2$. So, when the problem is symmetric, the typical case becomes the most favorable case and the clustering around truth assignments that satisfy many literal occurrences disappears.

If indeed "populism," i.e., the tendency of each variable to assume its majority value in the formula, is the main source of correlations in random k-SAT, then the second moment method is a good candidate for k-CNF models which do not encourage this tendency.[1] For example, one such model is *regular* random k-SAT, in which every literal occurs exactly the same number of times. Such formulas can be analyzed using a model analogous to the configuration model of random graphs, i.e., by taking precisely d copies of each literal and partitioning the resulting $2dn$ copies into clauses randomly (exactly as in the second step of our two-step model for random k-SAT).

3.4. Geometry and connections to statistical physics. A key quantity in statistical physics is the *overlap distribution* between configurations of minimum energy, known as ground states. When a constraint satisfaction problem is satisfiable, ground states correspond to solutions, such as satisfying assignments, valid colorings, and so on. In the case of random k-SAT, the overlap distribution is the probability $P(\alpha)$ that a random pair of satisfying assignments have overlap αn. Our calculation of the expected number of pairs of solutions at each possible distance is thus a weighted average of $P(\alpha)$ over all formulas, whereby formulas with more solutions contribute more heavily. Physicists call this weighted average the "annealed approximation" of $P(\alpha)$ and denote it $P_{\mathrm{ann}}(\alpha)$. It is worth pointing out that, while the annealed approximation clearly overemphasizes formulas with more satisfying assignments, Monasson and Zecchina conjectured in [50], based on the replica method, that it becomes asymptotically tight as $k \to \infty$.

On a more rigorous footing, it is easy to see that as long as Λ has a global maximum at $1/2$, $P_{\mathrm{ann}}(\alpha)$ is tightly concentrated around $1/2$, since $\Lambda(\alpha)^n$ is exponentially smaller than $\Lambda(1/2)^n$ for all other α. Our results establish that Λ is maximized at $1/2$ for densities up to $2^{k-1}\ln 2 - O(1)$. In other words, for densities almost all the way to the threshold, in the annealed approximation, almost all pairs of solutions have distance $n/2 + \Theta(\sqrt{n})$, just as if solutions were scattered uniformly at random throughout the hypercube.

Note that even if $P(\alpha)$ is concentrated around $1/2$ (rather than just $P_{\mathrm{ann}}(\alpha)$) this still allows for a typical geometry where there are exponentially many exponentially large clusters, each centered at a random assignment. Indeed, this is precisely the picture suggested by some very recent groundbreaking work of Mézard, Parisi, and Zecchina [47] and Meźard and Zecchina [48], based on nonrigorous techniques of statistical physics. If this is indeed the true picture, establishing it rigorously would require considerations much more refined than the second moment of the number of

[1] We describe recent developments on this point in the conclusions.

solutions. More generally, getting a better understanding of the typical geometry and its potential implications for algorithms appears to us a very challenging and very important open problem.

4. Groundwork.

4.1. Generative models. Given a set V of n Boolean variables, let $C_k = C_k(V)$ denote the set of all proper k-*clauses* on V, i.e., the set of all $2^k\binom{n}{k}$ disjunctions of k literals involving distinct variables. Similarly, given a set V of n vertices, let $E_k = E_k(V)$ be the set of all $\binom{n}{k}$ k-subsets of V. As we saw, a random k-CNF formula $F_k(n,m)$ is formed by selecting uniformly a random m-subset of C_k, while a random k-uniform hypergraph $H_k(n,m)$ is formed by selecting uniformly a random m-subset of E_k.

While $F_k(n,m)$ and $H_k(n,m)$ are perhaps the most natural models for generating random k-CNF formulas and random k-uniform hypergraphs, respectively, there are a number of slight variations of each model. Those are largely motivated by amenability to certain calculations. To simplify the discussion we focus on models for random formulas in the rest of this subsection. All our comments transfer readily to models for random hypergraphs.

For example, it is fairly common to consider the clauses as ordered k-tuples (rather than as k-sets) and/or to allow replacement in sampling the set C_k. Clearly, for properties such as satisfiability the issue of ordering is irrelevant. Moreover, as long as $m = O(n)$, essentially the same is true for the issue of replacement. To see that, observe that w.h.p. the number of repeated clauses is $q = o(n)$ and the subset of $m-q$ distinct clauses is uniformly random. Thus, if a monotone decreasing property (such as satisfiability) holds with probability p for a given $m = r^*n$ when replacement is allowed, it holds with probability $p - o(1)$ for all $r < r^*$ when replacement is not allowed.

The issue of selecting the literals of each clause with replacement (which might result in some "improper" clauses) is completely analogous. That is, the probability that a variable appears more than once in a given clause is at most $k^2/n = O(1/n)$ and hence w.h.p. there are $o(n)$ improper clauses. Finally, we note that by standard techniques our results also transfer to the $F_k(n,p)$ model where every clause appears independently of all others with probability p, for any p such that the expected number of k-clauses is $r^*n - n^\beta$ for some $\beta > 1/2$ (see [33]).

4.2. Strategy and tools. Our plan is to consider random k-CNF formulas formed by generating km independently and identically distributed random literals, where $m = rn$, and proving that if $X = X(F)$ is the number of NAE-satisfying assignments, then the following lemma holds.

LEMMA 2. *For all $\epsilon > 0$, $k \geq k_0(\epsilon)$, and $r < 2^{k-1}\ln 2 - (1+\ln 2)/2 - \epsilon$, there exists some constant $C = C(k,r) > 0$ such that for all sufficiently large n,*

$$\mathbf{E}[X^2] < C \times \mathbf{E}[X]^2.$$

By Lemma 1 and our discussion in section 4.1, this implies that $F_k(n, rn - o(n))$ is NAE-satisfiable, and thus satisfiable, w.u.p.p. Therefore, for all r as in Lemma 2, $F_k(n,rn)$ is satisfiable w.u.p.p. To boost this to a high probability result, thus establishing Theorem 1, we employ the following immediate corollary of Theorem 3.

COROLLARY 1. *If $F_k(n, r^*n)$ is satisfiable w.u.p.p., then for all $r < r^*$, $F_k(n,rn)$ is satisfiable w.h.p.*

Friedgut's arguments [32] also apply to NAE k-SAT, implying that $F_k(n, rn)$ is w.h.p. NAE-satisfiable for r as in Lemma 2. Thus, Lemma 2 readily yields (12) below, while (11) comes from noting that the expected number of NAE-satisfying assignments is $[2(1-2^{1-k})^r]^n$. (Similarly to hypergraphs, the techniques of [44, 24] can be used to improve the bound in (11) to $2^{k-1}\ln 2 - (\ln 2)/2 - 1/4 + t_k$, where $t_k \to 0$.) Indeed, we will see that the proof of Theorem 5 will yield Theorem 2 for random hypergraphs with little additional effort.

THEOREM 5. *For all $k \geq 3$, $F_k(n, m = rn)$ is w.h.p. non–NAE-satisfiable if*

$$r > 2^{k-1}\ln 2 - \frac{\ln 2}{2}. \tag{11}$$

There exists a sequence $t_k \to 0$ such that for all $k \geq 3$, $F_k(n, m = rn)$ is w.h.p. NAE-satisfiable if

$$r < 2^{k-1}\ln 2 - \frac{\ln 2}{2} - \frac{1+t_k}{2}. \tag{12}$$

As we saw in section 3.2, the second moment of the number of NAE-satisfying assignments is

$$2^n \sum_{z=0}^{n} \binom{n}{z} f_N (z/n)^{rn}.$$

A slightly more complicated sum will occur when we bound the second moment of the number of 2-colorings. To bound both sums we will use the following lemma which we prove in section 8.

LEMMA 3 (Laplace lemma). *Let ϕ be a positive, twice-differentiable function on $[0,1]$ and let $q \geq 1$ be a fixed integer. Let $t = n/q$ and let*

$$S_n = \sum_{z=0}^{t} \binom{t}{z}^q \phi(z/t)^n.$$

Letting $0^0 \equiv 1$, define g on $[0,1]$ as

$$g(\alpha) = \frac{\phi(\alpha)}{\alpha^\alpha (1-\alpha)^{1-\alpha}}.$$

If there exists $\alpha_{\max} \in (0,1)$ such that $g(\alpha_{\max}) \equiv g_{\max} > g(\alpha)$ for all $\alpha \neq \alpha_{\max}$ and $g''(\alpha_{\max}) < 0$, then there exists a constant $C = C(q, g_{\max}, g''(\alpha_{\max}), \alpha_{\max}) > 0$ such that for all sufficiently large n,

$$S_n < C\, n^{-(q-1)/2}\, g_{\max}^n.$$

5. Bounding the second moment for NAE k-SAT. Recall that if X is the number of NAE-assignments, then

$$\mathbf{E}[X] = 2^n (1-2^{1-k})^{rn}$$

and

$$\mathbf{E}[X^2] = 2^n \sum_{z=0}^{n} \binom{n}{z} f_N (z/n)^{rn}, \tag{13}$$

where

$$f_N(\alpha) = 1 - 2^{2-k} + 2^{1-k}\left(\alpha^k + (1-\alpha)^k\right).$$

To bound the sum in (13) we apply Lemma 3 with $q = 1$ and $\phi(\alpha) = f_N(\alpha)^r$. Thus, $g = g_r$, where

$$g_r(\alpha) = \frac{f_N(\alpha)^r}{\alpha^\alpha (1-\alpha)^{1-\alpha}}. \tag{14}$$

To show that Lemma 3 applies, we will prove in section 7 that the following lemma holds.

LEMMA 4. *For every $\epsilon > 0$, there exists $k_0 = k_0(\epsilon)$ such that for all $k \geq k_0$, if*

$$r < 2^{k-1}\ln 2 - \frac{\ln 2}{2} - \frac{1}{2} - \epsilon,$$

then $g_r(\alpha) < g_r(1/2)$ for all $\alpha \neq 1/2$, and $g_r''(1/2) < 0$.

Therefore, for all r, k, and ϵ as in Lemma 4, there exists a constant $C = C(k,r) > 0$ such that

$$\mathbf{E}[X^2] < C \times 2^n g_r(1/2)^n.$$

Since $\mathbf{E}[X]^2 = 2^n g_r(1/2)^n$, we get that for all r, k, ϵ as in Lemma 4

$$\mathbf{E}[X^2] < C \times \mathbf{E}[X]^2.$$

6. Bounding the second moment for hypergraph 2-colorability. Just as for NAE k-SAT, it will be easier to work with the model in which generating a random hypergraph corresponds to choosing km vertices uniformly at random with replacement and letting the first k vertices form the first hyperedge, the second k vertices form the second hyperedge, etc.

In [5] we proved (2) of Theorem 2 by letting X be the set of all 2-colorings and using a convexity argument to show that $\mathbf{E}[X^2]$ is dominated by the contribution of *balanced* colorings, i.e., colorings with an equal number of black and white vertices. Here we follow a simpler approach suggested by Karger; namely, we *define* X to be the number of balanced 2-colorings. We emphasize that, while technically convenient, the restriction to balanced 2-colorings is not essential for the second moment method to succeed on hypergraph 2-colorability; i.e., one has $\mathbf{E}[X^2] = O(\mathbf{E}[X]^2)$ even if X is the number of all 2-colorings.

Of course, in order for balanced colorings to exist n must be even and we will assume that in our calculations below. To get Theorem 2 for all sufficiently large n, we observe that if for a given c^*, $H_k(2n, m = 2c^*n)$ is w.h.p. 2-colorable, then for all $c < c^*$, $H_k(n, cn)$ is w.h.p. 2-colorable since deleting a random vertex of $H_k(2n, 2c^*n)$ w.h.p. removes $o(n)$ edges. With this in mind, in the following we let X be the number of balanced 2-colorings and assume that n is even.

Since the vertices in each hyperedge are chosen uniformly with replacement, the probability that a random hyperedge is bichromatic in a fixed balanced partition is $1 - 2^{1-k}$. Since there are $\binom{n}{n/2}$ such partitions and the m hyperedges are drawn independently, we have

$$\mathbf{E}[X] = \binom{n}{n/2}\left(1 - 2^{1-k}\right)^m. \tag{15}$$

To calculate the second moment, as we did for [NAE] k-SAT, we write $\mathbf{E}[X^2]$ as a sum over all pairs of balanced partitions. In order to estimate this sum we first observe that if two balanced partitions σ and τ have exactly z black vertices in common, then they must also have exactly z white vertices in common. Thus σ and τ define four groups of vertices: z that are black in both, z that are white in both, $n/2-z$ that are black in σ and white in τ, and $n/2-z$ that are white in σ and black in τ. Clearly, a random hyperedge is monochromatic in both σ and τ if and only if all its vertices fall into the same group. Since the vertices of each hyperedge are chosen uniformly with replacement, this probability is

$$2\left(\frac{z}{n}\right)^k + 2\left(\frac{n/2-z}{n}\right)^k = 2^{1-k}\left[\left(\frac{2z}{n}\right)^k + \left(1-\frac{2z}{n}\right)^k\right].$$

Thus, by inclusion-exclusion, the probability that a random hyperedge is bichromatic in both σ and τ is

$$1 - 2^{2-k} + 2^{1-k}\left[\left(\frac{2z}{n}\right)^k + \left(1-\frac{2z}{n}\right)^k\right] = f_N(2z/n),$$

where $f_N(\alpha) = 1 - 2^{2-k} + 2^{1-k}(\alpha^k + (1-\alpha)^k)$ is the function we defined for NAE k-SAT in (9).

Moreover, observe that the number of pairs of partitions with such overlap is

$$\binom{n}{z,\, z,\, n/2-z,\, n/2-z} = \binom{n}{n/2}\binom{n/2}{z}^2.$$

Since hyperedges are drawn independently and with replacement, by summing over z we thus get

$$\mathbf{E}[X^2] = \binom{n}{n/2}\sum_{z=0}^{n/2}\binom{n/2}{z}^2 f_N(2z/n)^{cn}.$$

To bound this sum we apply Lemma 3 with $q=2$ and $\phi(\alpha) = f_N(\alpha)^c$. Felicitously, we find ourselves maximizing a function g_c which, if we replace c with r, is exactly the same function g_r we defined in (14) for NAE k-SAT. Thus, setting $c=r$ where k, r and ϵ are as in Lemma 4, g_c is maximized at $\alpha = 1/2$ with $g''(1/2) < 0$, and Lemma 3 implies that there exists a constant $C = C(r,k) > 0$ such that

$$\mathbf{E}[X^2] < C\, n^{-1/2}\binom{n}{n/2} g_c(1/2)^n.$$

We now bound $\mathbf{E}[X]$ from below using Stirling's approximation (29) and get

$$\frac{\mathbf{E}[X^2]}{\mathbf{E}[X]^2} < C \times \frac{n^{-1/2}\binom{n}{n/2} g_c(1/2)^n}{\binom{n}{n/2}^2 (1-2^{1-k})^{2cn}} = C \times \frac{n^{-1/2}\, 2^n}{\binom{n}{n/2}} \to C \times \sqrt{\frac{\pi}{2}}.$$

To complete the proof, analogously to [NAE] k-SAT, we use the following "boosting" corollary of Theorem 4.

COROLLARY 2. *If $H_k(n, c^* n)$ is w.u.p.p. 2-colorable, then for all $c < c^*$, $H_k(n, cn)$ is w.h.p. 2-colorable.*

7. Proof of Lemma 4. We need to prove $g_r''(1/2) < 0$ and $g_r(\alpha) < g_r(1/2)$ for all $\alpha \neq 1/2$. As g_r is symmetric around $1/2$, we can restrict to $\alpha \in (1/2, 1]$. We divide $(1/2, 1]$ into two parts and handle them with two separate lemmata. The first lemma deals with $\alpha \in (1/2, 0.9]$ and also establishes that $g_r''(1/2) < 0$.

LEMMA 5. *Let $\alpha \in (1/2, 0.9]$. For all $k \geq 74$, if $r \leq 2^{k-1} \ln 2$, then $g_r(\alpha) < g_r(1/2)$ and $g_r''(1/2) < 0$.*

The second lemma deals with $\alpha \in (0.9, 1]$.

LEMMA 6. *Let $\alpha \in (0.9, 1]$. For every $\epsilon > 0$ and all $k \geq k_0(\epsilon)$, if $r \leq 2^{k-1} \ln 2 - \frac{\ln 2}{2} - \frac{1}{2} - \epsilon$, then $g_r(\alpha) < g_r(1/2)$.*

Combining Lemmata 5 and 6 we see that for every $\epsilon > 0$ and $k \geq k_0 = k_0(\epsilon)$, if

$$r \leq 2^{k-1} \ln 2 - \frac{\ln 2}{2} - \frac{1}{2} - \epsilon,$$

then $g_r(\alpha) < g_r(1/2)$ for all $\alpha \neq 1/2$ and $g_r''(1/2) < 0$, establishing Lemma 4.

We prove Lemmata 5 and 6 below. The reader should keep in mind that we have made no attempt to optimize the value of k_0 in Lemma 6, aiming instead for proof simplicity. For the lower bounds presented in Table 1 we computed numerically, for each k, the largest value of r for which the conclusions of Lemma 4 hold. In each case, the condition $g''(1/2) < 0$ was satisfied with room to spare, while establishing $g(1/2) > g(\alpha)$ for all $\alpha \neq 1/2$ was greatly simplified by the fact that g never has more than three local extrema in $[0, 1]$.

Proof of Lemma 5. We will first prove that for $k \geq 74$, g_r is strictly decreasing in $\alpha = (1/2, 0.9]$, thus establishing $g_r(\alpha) < g_r(1/2)$. Since g_r is positive, to do this it suffices to prove that $(\ln g_r)' = g_r'/g_r < 0$ in this interval. In fact, since $g_r'(\alpha) = (\ln g_r)' = 0$ at $\alpha = 1/2$, it will suffice to prove that for $\alpha \in [1/2, 0.9]$ we have $(\ln g_r)'' < 0$. Now,

$$\begin{aligned}(\ln g_r(\alpha))'' &= r\left(\frac{f''(\alpha)}{f(\alpha)} - \frac{f'(\alpha)^2}{f(\alpha)^2}\right) - \frac{1}{\alpha(1-\alpha)} \\ &\leq r\,\frac{f''(\alpha)}{f(\alpha)} - \frac{1}{\alpha(1-\alpha)}. \qquad (16)\end{aligned}$$

To show that the right-hand side (R.H.S.) of (16) is negative we first note that for $\alpha \geq 1/2$ and $k > 3$,

$$f''(\alpha) = 2^{1-k}k(k-1)(\alpha^{k-2} + (1-\alpha)^{k-2}) < 2^{2-k}\alpha^{k-2}k^2$$

is monotonically increasing. Therefore, $f''(\alpha) \leq f''(0.9) < 2^{2-k}\,0.9^{k-2}\,k^2$.

Moreover, for all α, $f(\alpha) \geq f(1/2) = (1-2^{-k})^2$. Therefore, since $1/(\alpha(1-\alpha)) \geq 4$ and $r \leq 2^{k-1}\ln 2$, it suffices to observe that for all $k \geq 74$,

$$(2^{k-1}\ln 2) \times \frac{2^{2-k}\,0.9^{k-2}\,k^2}{(1-2^{-74})^2} - 4 < 0.$$

Finally, recalling that $g'(1/2) = 0$ and using

$$(\ln g_r)'' = \frac{g_r''(\alpha)}{g_r(\alpha)} - \frac{g_r'(\alpha)^2}{g_r(\alpha)^2},$$

we see that $g_r''(1/2) < 0$ since $(\ln g_r)''(1/2) < 0$. □

Proof of Lemma 6. We let $h(\alpha) = -\alpha \ln \alpha - (1-\alpha)\ln(1-\alpha)$ denote the entropy function and for all $\alpha > 1/2$ we define

$$w(\alpha) \equiv \frac{f(\alpha) - f(1/2)}{f(1/2)} = \frac{2^{1-k}(\alpha^k + (1-\alpha)^k - 2^{1-k})}{(1-2^{1-k})^2} > 0.$$

By the definition of g_r, we thus see that $g_r(\alpha) < g_r(1/2)$ if and only if

$$\frac{r}{\ln 2 - h(\alpha)} < \frac{1}{\ln(1+w(\alpha))}. \tag{17}$$

Moreover, we observe that for any $x > 0$,

$$\frac{1}{\ln(1+x)} \geq \frac{1}{x} + \frac{1}{2} - \frac{x}{12}.$$

Since $f(\alpha) - f(1/2) < 2^{1-k}$ and $f(1/2) > 1 - 2^{2-k}$, we thus see that (17) holds as long as

$$\frac{r}{\ln 2 - h(\alpha)} < \frac{2^{k-1}-2}{\alpha^k + (1-\alpha)^k - 2^{1-k}} + \frac{1}{2} - \frac{2^{1-k}}{12(1-2^{2-k})}. \tag{18}$$

Now observe that for any $0 < \alpha < 1$ and $0 \leq q < \alpha^k$,

$$\frac{1}{\alpha^k - q} \geq 1 + k(1-\alpha) + q.$$

Since $\alpha > 1/2$ we can set $q = 2^{1-k} - (1-\alpha)^k$, yielding

$$\frac{1}{\alpha^k + (1-\alpha)^k - 2^{1-k}} \geq 1 + k(1-\alpha) + 2^{1-k} - (1-\alpha)^k.$$

Since $2^k(1-\alpha)^k < 5^{-k}$, we find that (18) holds as long as $r \leq \phi(y) - 2^{3-k}$, where

$$\phi(\alpha) \equiv \big(\ln 2 - h(\alpha)\big)\left(2^{k-1} + (2^{k-1}-2)k(1-\alpha) - \frac{1}{2}\right).$$

We are thus left to minimize ϕ in $(0.9, 1]$. Since ϕ is differentiable, its minima can only occur at 0.9 or 1, or where $\phi' = 0$. The derivative of ϕ is

$$\phi'(\alpha) = (2^{k-1}-2) \times \left[-k\left(\ln 2 - h(\alpha)\right) + \left(\ln \alpha - \ln(1-\alpha)\right)\left(1 + k(1-\alpha) + \frac{3}{2^k - 4}\right)\right]. \tag{19}$$

Note now that for all $k > 1$

$$\lim_{\alpha \to 1} \frac{\phi'(\alpha)}{\ln(1-\alpha)} = -\frac{2^k - 1}{2};$$

i.e., the derivative of ϕ as $\alpha \to 1$ becomes positively infinite. At the same time,

$$\phi'(0.9) < -0.07 \times 2^k k + 1.1\,(2^k - 1) + 0.3\,k$$

is negative for $k \geq 16$. Therefore, ϕ is minimized in the interior of $(0.9, 1]$ for all $k \geq 16$. Setting ϕ' to zero gives

$$-\ln(1-\alpha) = \frac{k\,(\ln 2 - h(\alpha))}{1 + k(1-\alpha) + 3/(2^k - 4)} - \ln \alpha. \tag{20}$$

By "bootstrapping" we derive a tightening series of lower bounds on the solution for the left-hand side (L.H.S.) of (20) for $\alpha \in (0.9, 1)$. Note first that we have an easy upper bound,

$$-\ln(1-\alpha) < k \ln 2 - \ln \alpha. \tag{21}$$

At the same time, if $k > 2$, then $3/(2^k - 4) < 1$, implying

$$-\ln(1-\alpha) > \frac{k\,(\ln 2 - h(\alpha))}{2 + k(1-\alpha)} - \ln \alpha. \tag{22}$$

If we write $k(1-\alpha) = B$, then (22) becomes

$$-\ln(1-\alpha) > \frac{\ln 2 - h(\alpha)}{1-\alpha} \left(\frac{B}{B+2} \right) - \ln \alpha. \tag{23}$$

By inspection, if $B \geq 3$, the R.H.S. of (23) is greater than the L.H.S. for all $\alpha > 0.9$, yielding a contradiction. Therefore, $k(1-\alpha) < 3$ for all $k > 2$. Since $\ln 2 - h(\alpha) > 0.36$ for $\alpha > 0.9$, we see that for $k > 2$, (22) implies

$$-\ln(1-\alpha) > 0.07\,k. \tag{24}$$

Finally, observe that (24) implies that as k increases, the denominator of (20) approaches 1.

To bootstrap, we note that since $\alpha > 1/2$ we have

$$h(\alpha) \leq -2(1-\alpha)\ln(1-\alpha) \tag{25}$$

$$\begin{aligned} &< 2\,\mathrm{e}^{-0.07\,k}(k \ln 2 - \ln 0.9) \\ &< 2\,k\,\mathrm{e}^{-0.07\,k}, \end{aligned} \tag{26}$$

where (26) relies on (21) and (24). Moreover, $\alpha > 1/2$ implies $-\ln \alpha \leq 2(1-\alpha) < 2\,\mathrm{e}^{-0.07\,k}$. Thus, by using (24) and the fact $1/(1+x) > 1 - x$ for all $x > 0$, (20) gives for $k \geq 3$

$$\begin{aligned} -\ln(1-\alpha) &> \frac{k\,(\ln 2 - h(\alpha))}{1 + k(1-\alpha) + 3/(2^k - 4)} \\ &> \frac{k\,(\ln 2 - 2\,k\,\mathrm{e}^{-0.07\,k})}{1 + 2\,k\,\mathrm{e}^{-0.07\,k}} \\ &> k\,(\ln 2 - 2\,k\,\mathrm{e}^{-0.07\,k})(1 - 2\,k\,\mathrm{e}^{-0.07\,k}) \\ &> k \ln 2 - 4\,k^2\,\mathrm{e}^{-0.07\,k}. \end{aligned} \tag{27}$$

For $k \geq 166$, $4\,k^2\,\mathrm{e}^{-0.07\,k} < 1$. Thus, by (27), we have $1 - \alpha < 3 \times 2^{-k}$. This, in turn, implies $-\ln \alpha \leq 2(1-\alpha) < 6 \times 2^{-k}$ and thus, by (25) and (21), we have for $\alpha > 0.9$

$$h(\alpha) < 6 \times 2^{-k}(k \ln 2 - \ln \alpha) < 5\,k\,2^{-k}. \tag{28}$$

Plugging (28) into (20) to bootstrap again, we get that for $k \geq 166$

$$\begin{aligned} -\ln(1-\alpha) &> \frac{k\,(\ln 2 - 5\,k\,2^{-k})}{1 + 3\,k\,2^{-k} + 3/(2^k-4)} \\ &> \frac{k\,(\ln 2 - 5\,k\,2^{-k})}{1 + 6\,k\,2^{-k}} \\ &> k\,(\ln 2 - 5\,k\,2^{-k})(1 - 6\,k\,2^{-k}) \\ &> k \ln 2 - 11\,k^2\,2^{-k}. \end{aligned}$$

Since $e^x < 1 + 2x$ for $x < 1$ and $11\,k^2\,2^{-k} < 1$ for $k > 10$, we see that for $k \geq 166$,

$$1 - \alpha < 2^{-k} + 22\,k^2\,2^{-2k}.$$

Plugging into (21) the fact $-\ln\alpha < 6 \times 2^{-k}$, we get $-\ln(1-\alpha) < k \ln 2 + 6 \times 2^{-k}$. Using that $\mathrm{e}^{-x} \geq 1 - x$ for $x \geq 0$, we get the closely matching upper bound,

$$1 - \alpha > 2^{-k} - 6 \times 2^{-2k}.$$

Thus, we see that for $k \geq 166$, ϕ is minimized at an $\alpha_{\min}$ which is within δ of $1 - 2^{-k}$, where $\delta = 22\,k^2\,2^{-2k}$. Let T be the interval $[1 - 2^{-k} - \delta, 1 - 2^{-k} + \delta]$. Clearly the minimum of ϕ is at least $\phi(1-2^{-k}) - \delta \times \max_{\alpha \in T} |\phi'(\alpha)|$. It is easy to see from (19) that if $\alpha \in T$, then $|\phi'(\alpha)| \leq 2\,k\,2^k$.

Now, a simple calculation using that $\ln(1 - 2^{-k}) > -2^{-k} - 2^{-2k}$ for $k \geq 1$ gives

$$\begin{aligned} \phi(1-2^{-k}) &= \frac{1}{2}\big((2^k - k)\ln 2 + (2^k - 1)\ln(1-2^{-k})\big) \times \big(1 + (k-1)\,2^{-k} - k\,2^{2-2k}\big) \\ &> 2^{k-1}\ln 2 - \frac{\ln 2}{2} - \frac{1}{2} - k^2\,2^{-k}. \end{aligned}$$

Therefore,

$$\phi_{\min} \geq 2^{k-1}\ln 2 - \frac{\ln 2}{2} - \frac{1}{2} - 45\,k^3\,2^{-k}.$$

Finally, recall that (17) holds as long as $r < \phi_{\min} - 2^{3-k}$, for example, if

$$r < 2^{k-1}\ln 2 - \frac{\ln 2}{2} - \frac{1}{2} - 46\,k^3\,2^{-k}.$$

Clearly, we can take $k_0 = O(\ln \epsilon^{-1})$ so that for all $k \geq k_0$ the error term $46\,k^3\,2^{-k}$ is smaller than any $\epsilon > 0$. □

8. Proof of Lemma 3. The idea behind Lemma 3 is that sums of this type are dominated by the contribution of $\Theta(n^{1/2})$ terms around the maximum term. The proof amounts to replacing the sum by an integral and using the Laplace method for asymptotic integrals [23].

We start by establishing two upper bounds for the terms of S_n, a crude one and one which is sharper when $\alpha = z/t$ is bounded away from both 0 and 1. For the sharper bound we will use the following form of Stirling's approximation, valid for all $n > 0$:

$$\sqrt{2\pi n} < \frac{n!}{(n/\mathrm{e})^n} < \sqrt{2\pi n}\,(1 + 1/n)\,. \tag{29}$$

The zth term of S_n is $\binom{t}{z}^q \phi(z/t)^n$, where $n = qt$ and $\phi(\alpha) = g(\alpha)\,\alpha^\alpha(1-\alpha)^{1-\alpha}$. Fix any $\delta > 0$ and suppose that $z = \alpha t$, where $\alpha \in [\delta, 1-\delta]$. Then (29) yields

$$\binom{t}{z}^q \phi(z/t)^n < s(\alpha)\, g(\alpha)^n \left(1 + \frac{q}{n}\right)^q, \tag{30}$$

where $s(\alpha) = (2\pi\alpha(1-\alpha)t)^{-q/2}$. In addition to (30), valid for $z \in [t\delta, t(1-\delta)]$, we will also use a cruder bound, valid for all $0 \le z \le t$. Namely, by induction on $t - z$ it is easy to show that $\binom{t}{z} \le t^t/[z^z(t-z)^{t-z}]$, implying

$$\binom{t}{z}^q \phi(z/t)^n \;<\; g(\alpha)^n. \tag{31}$$

Recall now that $g(\alpha_{\max}) > g(\alpha)$ for all $\alpha \ne \alpha_{\max}$. If I_ϵ denotes the interval $[\alpha_{\max} - \epsilon, \alpha_{\max} + \epsilon]$, then for every $\epsilon > 0$, there exists a constant $g_\epsilon < g(\alpha_{\max}) = g_{\max}$ such that $g(\alpha) < g_\epsilon$ for all $\alpha \notin I_\epsilon$. Let $z_\epsilon^- = \lfloor(\alpha_{\max} - \epsilon)t\rfloor$ and $z_\epsilon^+ = \lceil(\alpha_{\max} + \epsilon)t\rceil$, and let

$$S_n^{(\epsilon)} = \sum_{z=z_\epsilon^-}^{z_\epsilon^+} \binom{t}{z}^q \phi(z/t)^n. \tag{32}$$

We use (30) to bound the terms in $S_n^{(\epsilon)}$ and (31) to bound the remaining terms of S_n. Since $\lim_{n\to\infty}(1 + q/n)^q = 1$, and since $\lim_{n\to\infty} n^s\, g_\epsilon^n / g_{\max}^n = 0$ for any s, we see that for every $\epsilon > 0$

$$S_n < (C_\epsilon t)^{-q/2} \times \sum_{z=z_\epsilon^-}^{z_\epsilon^+} g(z/t)^n \tag{33}$$

for any constant $C_\epsilon > 2\pi \times \min\{(\alpha_{\max} - \epsilon)(1 - \alpha_{\max} + \epsilon), (\alpha_{\max} + \epsilon)(1 - \alpha_{\max} - \epsilon)\}$.

Say that a twice-differentiable function $\psi(x)$ is *unimodal* on an interval $[a, b]$ if ψ' has a unique zero $c \in [a, b]$ with $a < c < b$ and, furthermore, $\psi''(c) < 0$. Since $g_{\max} > g(\alpha)$ for all $\alpha \ne \alpha_{\max}$ and $g''(\alpha_{\max}) < 0$, we can take ϵ small enough so that g is unimodal on I_ϵ. This implies that $\ln g$ is also unimodal on I_ϵ and, for $n \ge 1$, that g^n is unimodal also. For any function $\gamma(x)$ which is nonnegative and unimodal on an interval $[a, b]$ with maximum $\gamma_{\max}$, no matter how tightly peaked, we have

$$\sum_{z=\lfloor at\rfloor}^{\lceil bt\rceil} \gamma(z/t) \le t \int_a^b \gamma(x)\,\mathrm{d}x \;+ \gamma_{\max},$$

and thus

$$\sum_{z=z_\epsilon^-}^{z_\epsilon^+} g(z/t)^n \le \frac{n}{q} \int_{I_\epsilon} g(x)^n\,\mathrm{d}x \;+ g_{\max}^n. \tag{34}$$

We evaluate this last integral using Lemma 7, i.e., the Laplace method for asymptotic integrals.

LEMMA 7 (see [23, section 4.2]). *Let $h(x)$ be unimodal on $[a, b]$, where c is the unique zero of h' in $[a, b]$. Then*

$$\lim_{n\to\infty} \int_a^b \mathrm{e}^{nh(x)}\,\mathrm{d}x \;\sim\; \sqrt{\frac{2\pi}{n\,|h''(c)|}}\,\mathrm{e}^{nh(c)}.$$

Applying Lemma 7 to (34) with $h = \ln g$ and $c = \alpha_{\max}$, we see that

$$S_n < C\, n^{-(q-1)/2}\, g_{\max}^n,$$

where $C = (2\pi)^{-(q-1)/2} \times q^{q/2} \times \sqrt{g_{\max}/|g''(\alpha_{\max})|}$. □

9. Conclusions. Before this work, lower bounds on the thresholds of random constraint satisfaction problems were largely derived by analyzing very simple heuristics. Here, instead, we derive such bounds by applying the second moment method to the number of solutions. In particular, for random NAE k-SAT and random hypergraph 2-colorability we determine the location of the threshold within a small additive constant for all k. As a corollary, we establish that the asymptotic order of the random k-SAT threshold is $\Theta(2^k)$, answering a long-standing open question.

Since this work first appeared [4, 5], our methods have been extended and applied to other problems. For random k-SAT, Achlioptas and Peres [7] confirmed our suspicion (see section 3.3) that the main source of correlations in random k-SAT is the "populist" tendency of satisfying assignments toward the majority vote assignment. By considering a carefully constructed random variable which focuses on balanced solutions, i.e., on satisfying assignments that satisfy roughly half of all literal occurrences, they showed $r_k \geq 2^k \ln 2 - k/2 - O(1)$, establishing $r_k \sim 2^k \ln 2$.

In [8], Achlioptas, Naor, and Peres extended the approach of balanced solutions to Max k-SAT. Let us say that a k-CNF formula is p-satisfiable if there exists a truth assignment which satisfies at least $(1 - 2^{-k} + p2^{-k})$ of all clauses; note that every k-CNF is 0-satisfiable. For $p \in (0, 1]$ let $r_k(p)$ denote the threshold for $F_k(n, m = rn)$ to be p-satisfiable (so that $r_k(1) = r_k$). In [8], the result $r_k = r_k(1) \sim 2^k \ln 2$ of [7] was extended to all $p \in (0, 1]$, showing that

$$r_k(p) \sim \frac{2^k \ln 2}{p + (1-p)\ln(1-p)}.$$

In both [7] and [8], controlling the variance crucially depends on focusing on an appropriate subset of solutions (akin to our NAE-assignments but less heavy-handed). In [9], Achlioptas and Naor applied the naive second moment method to the canonical symmetric constraint satisfaction problem, i.e., to the number of k-colorings of a random graph. Bearing out our belief that the naive approach should work for symmetric problems, they obtained asymptotically tight bounds for the k-colorability threshold, and in [10] Achlioptas and Moore extended this analysis to random d-regular graphs. The difficulty here is that the "overlap parameter" is a $k \times k$ matrix rather than a single real $\alpha \in [0, 1]$. Since $k \to \infty$, this makes the asymptotic analysis dramatically harder and much closer to the realm of statistical mechanics calculations.

We propose several questions for further work.

1. Does the second moment method give tight lower bounds on the threshold of all constraint satisfaction problem with a permutation symmetry?
2. Does it perform well for problems that are symmetric "on average"? For example, does it perform well for *regular* random k-SAT where every literal appears an equal number of times?
3. What rigorous connections can be made between the success of the second moment method and the notion of "replica symmetry" in statistical physics?
4. Is there a polynomial-time algorithm that succeeds with uniformly positive probability close to the threshold, or at least for $r = \omega(k) \times 2^k/k$ where $\omega(k) \to \infty$?

Acknowledgments. We are grateful to Paul Beame, Ehud Friedgut, Michael Molloy, Assaf Naor, Yuval Peres, Alistair Sinclair, and Chris Umans for reading earlier versions and making many helpful suggestions, and to Remi Monasson for discussions on the replica method. We would like to thank Henry Cohn for bringing [23] to our attention. Finally, we would like to thank the anonymous referees for a number of excellent suggestions.

REFERENCES

[1] D. Achlioptas, *Setting two variables at a time yields a new lower bound for random* 3*-SAT*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, 2000, pp. 28–37.

[2] D. Achlioptas, P. Beame, and M. Molloy, *A sharp threshold in proof complexity*, J. Comput. System Sci., 68 (2004), pp. 238–268.

[3] D. Achlioptas, P. Beame, and M. Molloy, *Exponential bounds for DPLL below the satisfiability threshold*, in Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, 2004, pp. 132–133.

[4] D. Achlioptas and C. Moore, *The asymptotic order of the random k-SAT threshold*, in Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002, pp. 779–788.

[5] D. Achlioptas and C. Moore, *On the* 2*-colorability of random hypergraphs*, in Randomization and Approximation Techniques in Computer Science, Lecture Notes in Comput. Sci. 2483, Springer-Verlag, Berlin, 2002, pp. 78–90.

[6] D. Achlioptas, J. H. Kim, M. Krivelevich, and P. Tetali, *Two-coloring random hypergraphs*, Random Structures Algorithms, 20 (2002), pp. 249–259.

[7] D. Achlioptas and Y. Peres, *The random k-SAT threshold is* $2^k \ln 2 - O(k)$, J. Amer. Math. Soc., 17 (2004), pp. 947–973.

[8] D. Achlioptas, A. Naor, and Y. Peres, *On the maximum satisfiability of random formulas*, in Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, 2003, pp. 362–370.

[9] D. Achlioptas and A. Naor, *On the k-colorability threshold*, Ann. of Math. (2), 162 (2005), pp. 1333–1349.

[10] D. Achlioptas and C. Moore, *The chromatic number of random regular graphs*, in Proceedings of the 8th International Workshop on Randomization and Computation, Lecture Notes in Comput. Sci. 3122, Springer-Verlag, Berlin, 2004, pp. 219–228.

[11] D. Achlioptas and G. B. Sorkin, *Optimal myopic algorithms for random* 3*-SAT*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, 2000, pp. 590–600.

[12] N. Alon and J. Spencer, *A Note on Coloring Random k-Sets*, manuscript.

[13] N. Alon and J. Spencer, *The Probabilistic Method*, Wiley & Sons, New York, 1992.

[14] J. Beck, *On* 3*-chromatic hypergraphs*, Discrete Math., 24 (1978), pp. 127–137.

[15] F. Bernstein, *Zur theorie der trigonometrische reihe*, Leipz. Ber., 60 (1908), pp. 325–338.

[16] A. Z. Broder, A. M. Frieze, and E. Upfal, *On the satisfiability and maximum satisfiability of random* 3*-CNF formulas*, in Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, Austin, TX, 1993, pp. 322–330.

[17] M.-T. Chao and J. Franco, *Probabilistic analysis of two heuristics for the* 3*-satisfiability problem*, SIAM J. Comput., 15 (1986), pp. 1106–1118.

[18] M.-T. Chao and J. Franco, *Probabilistic analysis of a generalization of the unit-clause literal selection heuristics for the k-satisfiability problem*, Inform. Sci., 51 (1990), pp. 289–314.

[19] P. Cheeseman, R. Kanefsky, and W. Taylor, *Where the really hard problems are*, in Proceedings of the 12th International Joint Conference on Artificial Intelligence, 1991, pp. 331–337.

[20] V. Chvátal and B. Reed, *Mick gets some (the odds are on his side)*, in Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science, 1992, pp. 620–627.

[21] V. Chvátal and E. Szemerédi, *Many hard examples for resolution*, J. ACM, 35 (1988), pp. 759–768.

[22] S. A. Cook, *The complexity of theorem-proving procedures*, in Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, 1971, pp. 151–158.

[23] N. G. de Bruijn, *Asymptotic Methods in Analysis*, Dover, New York, 1981.

[24] O. Dubois and Y. Boufkhad, *A general upper bound for the satisfiability threshold of random r-SAT formulae*, J. Algorithms, 24 (1997), pp. 395–420.

[25] O. Dubois, Y. Boufkhad, and J. Mandler, *Typical random 3-SAT formulae and the satisfiability threshold*, in Electronic Colloquium on Computational Complexity 10, 2003; available online from http://www.informatik.uni-trier.de/~ley/db/journals/eccc/eccc10.html; also available in Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, 2000, pp. 126–127.
[26] A. El Maftouhi and W. Fernandez de la Vega, *On random 3-SAT*, Combin. Probab. Comput., 4 (1995), pp. 189–195.
[27] P. Erdős, *On a combinatorial problem*, Nordisk Mat. Tidskr., 11 (1963), pp. 5–10.
[28] P. Erdős and L. Lovász, *Problems and results on 3-chromatic hypergraphs and some related questions*, in Infinite and Finite Sets, Vol. II, Colloq. Math. Soc. Janos Bolyai 10, North–Holland, Amsterdam, 1975, pp. 609–627.
[29] W. Fernandez de la Vega, *On Random 2-SAT*, manuscript, 1992.
[30] P. Flajolet, D. E. Knuth, and B. Pittel, *The first cycles in an evolving graph*, Discrete Math., 75 (1989), pp. 167–215.
[31] J. Franco and M. Paull, *Probabilistic analysis of the Davis–Putnam procedure for solving the satisfiability problem*, Discrete Appl. Math., 5 (1983), pp. 77–87.
[32] E. Friedgut, *Necessary and sufficient conditions for sharp thresholds of graph properties, and the k-SAT problem*, J. Amer. Math. Soc., 12 (1999), pp. 1017–1054.
[33] A. M. Frieze and S. Suen, *Analysis of two simple heuristics on a random instance of k-SAT*, J. Algorithms, 20 (1996), pp. 312–355.
[34] A. Frieze and N. C. Wormald, *Random k-SAT: A tight threshold for moderately growing k*, Combinatorica, 25 (2005), pp. 297–305.
[35] A. Goerdt, *A threshold for unsatisfiability*, J. Comput. System Sci., 53 (1996), pp. 469–486.
[36] M. Hajiaghayi and G. B. Sorkin, *The satisfiability threshold of random 3-SAT is at least 3.52*, submitted; available online from http://www.arxiv.org/abs/math.CO/0310193.
[37] A. Haken, *The intractability of resolution*, Theoret. Comput. Sci., 39 (1985), pp. 297–308.
[38] S. Janson, T. Łuczak, and A. Ruciński, *Random Graphs*, John Wiley & Sons, New York, 2000.
[39] S. Janson, Y. C. Stamatiou, and M. Vamvakari, *Bounding the unsatisfiability threshold of random 3-SAT*, Random Structures Algorithms, 17 (2000), pp. 103–116.
[40] A. Kamath, R. Motwani, K. Palem, and P. Spirakis, *Tail bounds for occupancy and the satisfiability threshold conjecture*, Random Structures Algorithms, 7 (1995), pp. 59–80.
[41] A. Kaporis, L. M. Kirousis, and E. Lalas, *Selecting complementary pairs of literals*, in Proceedings of LICS 2003 Workshop on Typical Case Complexity and Phase Transitions, 2003.
[42] A. Kaporis, L. M. Kirousis, Y. C. Stamatiou, M. Vamvakari, and M. Zito, *The unsatisfiability threshold revisited*, Discrete Math., to appear.
[43] M. Karoński and T. Łuczak, *Random hypergraphs*, in Combinatorics, Paul Erdős Is Eighty, Vol. 2 (Keszthely, 1993), Bolyai Soc. Math. Stud. 2, Janos Bolyai Math. Soc., Budapest, 1996, pp. 283–293.
[44] L. M. Kirousis, E. Kranakis, D. Krizanc, and Y. Stamatiou, *Approximating the unsatisfiability threshold of random formulas*, Random Structures Algorithms, 12 (1998), pp. 253–269.
[45] M. Krivelevich and B. Sudakov, *The chromatic numbers of random hypergraphs*, Random Structures Algorithms, 12 (1998), pp. 381–403.
[46] L. Lovász, *Coverings and coloring of hypergraphs*, in Proceedings of the Fourth Southeastern Conference on Combinatorics, Graph Theory, and Computing, Boca Raton, FL, 1973, pp. 3–12.
[47] M. Mézard, G. Parisi, and R. Zecchina, *Analytic and algorithmic solution of random satisfiability problems*, Science, 297 (2002), pp. 812–815.
[48] M. Mézard and R. Zecchina, *Random K-satisfiability: From an analytic solution to a new efficient algorithm*, Phys. Rev. E (3), 66 (2002), 056126.
[49] D. G. Mitchell, B. Selman, and H. J. Levesque, *Hard and easy distributions of SAT problems*, in Proceedings of the 10th National Conference on Artificial Intelligence, 1992, pp. 459–462.
[50] R. Monasson and R. Zecchina, *Statistical mechanics of the random K-satisfiability model*, Phys. Rev. E (3), 56 (1997), pp. 1357–1370.
[51] J. Radhakrishnan and A. Srinivasan, *Improved bounds and algorithms for hypergraph 2-coloring*, Random Structures Algorithms, 16 (2000), pp. 4–32.
[52] J. Schmidt-Pruzan, E. Shamir, and E. Upfal, *Random hypergraph coloring algorithms and the weak chromatic number*, J. Graph Theory, 8 (1985), pp. 347–362.
[53] A. Urquhart, *Hard examples for resolution*, J. ACM, 34 (1987), pp. 209–219.

SIAM J. COMPUT.
Vol. 36, No. 3, pp. 763–778

QUANTUM ALGORITHMS FOR SOME HIDDEN SHIFT PROBLEMS*

WIM VAN DAM†, SEAN HALLGREN‡, AND LAWRENCE IP§

Abstract. Almost all of the most successful quantum algorithms discovered to date exploit the ability of the Fourier transform to recover subgroup structures of functions, especially periodicity. The fact that Fourier transforms can also be used to capture shift structure has received far less attention in the context of quantum computation. In this paper, we present three examples of "unknown shift" problems that can be solved efficiently on a quantum computer using the quantum Fourier transform. For one of these problems, *the shifted Legendre symbol problem*, we give evidence that the problem is hard to solve classically, by showing a reduction from breaking algebraically homomorphic cryptosystems. We also define the *hidden coset problem,* which generalizes the hidden shift problem and the hidden subgroup problem. This framework provides a unified way of viewing the ability of the Fourier transform to capture subgroup and shift structure.

Key words. quantum computing, efficient algorithms, Legendre symbol

AMS subject classifications. 81P68, 68W40, 11Y16

DOI. 10.1137/S009753970343141X

1. Introduction.

The first problem to demonstrate a superpolynomial separation between random and quantum polynomial time was the recursive Fourier sampling problem [6]. Exponential separations were subsequently discovered by Simon [35], who defined a problem with respect to an oracle, and by Shor [34], who found polynomial-time quantum algorithms for factoring and discrete logarithms. We now understand that the natural generalization of Simon's problem and the factoring and discrete log problems is the hidden subgroup problem (HSP), and that when the underlying group is abelian and finitely generated, we can solve the HSP efficiently on a quantum computer. While recent results have continued to study important generalizations of the HSP (for example, [19, 21, 24, 25, 27, 37]), only the recursive Fourier sampling problem remains outside the abelian HSP framework.

In this paper, we give quantum algorithms for several hidden shift problems where we are given two functions f, g such that there is a shift s for which $f(x) = g(x + s)$ for all x. The problem is then to find s. We show how to solve this problem for several classes of functions, but perhaps the most interesting example is the shifted Legendre symbol problem, where g is the *Legendre symbol* with respect to a prime

*Received by the editors July 15, 2003; accepted for publication (in revised form) February 28, 2006; published electronically October 24, 2006. A preliminary version appeared as [15].
http://www.siam.org/journals/sicomp/36-3/43141.html

†Departments of Computer Science and Physics, University of California, Santa Barbara, Santa Barbara, CA 93106-5110 (vandam@cs.ucsb.edu). This author's work was supported in part by the NSA and ARDA under ARO contract W911NF-04-R-0009. Part of this work was done while the author was at MIT, University of California, Berkeley, MSRI, and HP Labs.

‡NEC Laboratories America, Inc., Princeton, NJ 08540 (hallgren@nec-labs.com). This author's work was supported at Caltech in part by an NSF Mathematical Sciences Postdoctoral Fellowship and in part by the NSF through the Institute for Quantum Information at the California Institute of Technology. Most of this work was done while the author was at the Mathematical Sciences Research Institute and the University of California, Berkeley, with partial support from DARPA QUIST grant F30602-01-2-0524.

§Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043 (lip@google.com). This author's work was supported by NSF grant CCR-0049092, DARPA grant F30602-00-2-0601, and DARPA QUIST grant F30602-01-2-2054. This work was done while the author was at the University of California, Berkeley, and the Institute for Quantum Information at the California Institute of Technology.

size finite field, and the problem is the following: "Given the function $f(x) = \left(\frac{x+s}{p}\right)$ as an oracle, find s."

The oracle problem that our algorithms solve can be viewed as the problem of predicting a pseudorandom function f. Such tasks play an important role in cryptography and have been studied extensively under various assumptions about how one is allowed to query the function (nonadaptive versus adaptive, deterministic versus randomized, etc.) [7, 31]. In this paper we consider the case where the function is queried in a quantum mechanical superposition of different values x. We show that if $f(x)$ is an s-shifted multiplicative character $\chi(x+s)$—like the Legendre symbol—then a polynomial-time quantum algorithm making such queries can determine the hidden shift s, breaking the pseudorandomness of f.

We conjecture that classically the shifted Legendre symbol is a pseudorandom function; that is, it is impossible to efficiently predict the value of the function after a polynomial number of queries if one is allowed only a classical algorithm with oracle access to f. Damgård gave partial evidence for this conjecture, proposing the related task: "Given a part of the Legendre sequence $\left(\frac{s}{p}\right), \left(\frac{s+1}{p}\right), \ldots, \left(\frac{s+\ell}{p}\right)$, where ℓ is $O(\log p)$, predict the next value $\left(\frac{s+\ell+1}{p}\right)$" as a hard problem with applications in cryptography [17].

As further evidence of our conjecture, we show that breaking certain algebraically homomorphic cryptosystems can be reduced to the shifted Legendre symbol problem. The reduction, together with our quantum algorithm for the shifted Legendre symbol problem, yields a polynomial-time quantum algorithm for breaking such cryptosystems. The best known classical algorithm [9] for breaking these cryptosystems is subexponential and is based on a smoothness assumption. Thus the shifted Legendre symbol problem is a problem for which there is an exponential separation between a quantum algorithm and the fastest known classical algorithm. These cryptosystems can also be broken by Shor's algorithm for period finding, but the two attacks on the cryptosystems appear to use completely different ideas.

While current quantum algorithms solve problems based on an underlying group and the Fourier transform over that group, we initiate the study of problems where there is an underlying ring or field. The Fourier transform over the additive group of the ring is defined using the characters of the additive group, the additive characters of the ring. Similarly, the multiplicative group of units induces multiplicative characters of the ring. The interplay between additive and multiplicative characters is well understood [30, 36], and we show that this connection can be exploited in quantum algorithms. In particular, we put a multiplicative character into the phase of the registers and compute the Fourier transform over the additive group. The resulting phases are the inner products between the multiplicative character and each of the additive characters, a Gauss sum. We hope the new tools presented here will lead to other quantum algorithms.

We give algorithms for three types of hidden shift problems. In the first problem, g is a multiplicative character of a finite field. Given f, a shifted version of g, the shift is uniquely determined from f and g. An example of a multiplicative character of $\mathbb{Z}/p\mathbb{Z}$ is the Legendre symbol. Our algorithm uses the Fourier transform over the additive group of a finite field. In the second problem, g is a multiplicative character of the ring $\mathbb{Z}/n\mathbb{Z}$. This problem has the feature that the shift is not uniquely determined by f and g, and our algorithm identifies all possible shifts. An example of a multiplicative character of $\mathbb{Z}/n\mathbb{Z}$ is the *Jacobi symbol*. In the third problem we have the same setup as in the second problem with the additional twist that n is unknown.

We also define the *hidden coset problem*, which is a generalization of the hidden shift problem and the hidden subgroup problem. This definition provides a unified way of viewing the quantum Fourier transform's ability to capture subgroup and shift structure.

Some of our hidden shift problems can be reduced to the nonabelian HSP, although efficient algorithms for these HSP instances are not known. The shifted Legendre symbol problem over $\mathbb{Z}/p\mathbb{Z}$ can be reduced to an instance of the HSP over the dihedral group $D_p = \mathbb{Z}/p\mathbb{Z} \rtimes \mathbb{Z}/2\mathbb{Z}$ if we assume a conjecture about subsequences of the Legendre symbol. Let $f(x,0) = \left(\left(\frac{x}{p}\right), \left(\frac{x+1}{p}\right), \ldots, \left(\frac{x+\ell}{p}\right)\right)$ and $f(x,1) = \left(\left(\frac{x+s}{p}\right), \left(\frac{x+s+1}{p}\right), \ldots, \left(\frac{x+s+\ell}{p}\right)\right)$, where s is unknown and $\ell = \text{polylog}(p)$. Then the hidden subgroup is $H = \{(0,0),(s,1)\}$. The conjecture that is necessary to ensure that f will be distinct on distinct cosets of H is thus the statement that the subsequence $\left(\left(\frac{x+s}{p}\right), \left(\frac{x+s+1}{p}\right), \ldots, \left(\frac{x+s+\ell}{p}\right)\right)$ is unique for every x (cf. Conjecture 2.1 in [9]). For the general shifted multiplicative character problem, the analogous reduction to the HSP may fail because f may not be distinct on distinct cosets. However, we can efficiently generate random coset states, that is, superpositions of the form $|x,0\rangle + |x+s,1\rangle$, although it is unknown how to use these to efficiently find s [18]. The issue of nondistinctness on cosets in the HSP has been studied for some groups [8, 20, 23, 22].

The existence of a time-efficient quantum algorithm for the shifted Legendre symbol problem was posed as an open question in [13]. The Fourier transform over the additive group of a finite field was independently proposed for the solution of a different problem in [4]. The current paper subsumes [14, 15, 26]. Building on the ideas in this paper, a quantum algorithm for estimating Gauss sums is described in [16].

This paper is organized as follows. Section 2 contains some definitions and facts. In section 3, we give some intuition for the ideas behind the algorithms. In section 4, we present an algorithm for the shifted multiplicative problem over finite fields, of which the shifted Legendre symbol problem is a special case, and show how we can use this algorithm to break certain algebraically homomorphic cryptosystems. In section 5, we extend our algorithm to the shifted multiplicative problem over rings $\mathbb{Z}/n\mathbb{Z}$. This has the feature that, unlike in the case of the finite field, the possible shifts may not be unique. We then show that this algorithm can be extended to the situation where n is unknown. In section 6, we show that all these problems lie within the general framework of the hidden coset problem. We give an efficient algorithm for the hidden coset problem provided g satisfies certain conditions. We also show how our algorithm can be interpreted as solving a deconvolution problem using Fourier transforms.

2. Background.

2.1. Notation and conventions. We use the following notation: ω_n is the nth root of unity $\exp(2\pi\mathrm{i}/n)$, and $\hat{f}$ denotes the Fourier transform of the function f. An algorithm computing in $\mathbb{F}_q$, $\mathbb{Z}/n\mathbb{Z}$, or G runs in polynomial time if it runs in time polynomial in $\log q$, $\log n$, or $\log |G|$.

In a ring $\mathbb{Z}/n\mathbb{Z}$ or a field $\mathbb{F}_q$, additive characters $\psi : \mathbb{Z}/n\mathbb{Z} \to \mathbb{C}^*$ or $\psi : \mathbb{F}_q \to \mathbb{C}^*$ are characters of the additive group, that is, $\psi(x+y) = \psi(x)\psi(y)$, and multiplicative characters $\chi : (\mathbb{Z}/n\mathbb{Z})^* \to \mathbb{C}^*$ or $\chi : \mathbb{F}_q^* \to \mathbb{C}^*$ are characters of the multiplicative group of units, that is, $\chi(xy) = \chi(x)\chi(y)$ for all x and y. We extend the definition of a multiplicative character to the entire ring or field by assigning the value zero to elements outside the unit group. All nonzero $\chi(x)$ values have unit norm and thus $\chi(x^{-1}) = \overline{\chi(x)}$.

We ignore the normalization term in front of a superposition unless we need to explicitly calculate the probability of measuring a particular value.

2.2. Computing superpositions. We will need to be able to compute the superposition $\sum_x f(x)|x\rangle$, where the function $f : G \to \mathbb{C}$ describes the *amplitudes* of the state in an efficient way. The specific functions f that we deal with in this article have the property that for each x either $f(x) = 0$ or $f(x)$ is an mth root of unity. Hence there is a function $f' : G \to \mathbb{Z}$ such that if $f(x) \neq 0$, then $f(x) = \exp(2\pi \mathrm{i} f'(x)/m)$. This additional function helps us in the following lemma, which describes how to construct the superpositions in an efficient way.

LEMMA 2.1 (computing superpositions). *Let $f : G \to \mathbb{C}$ be a complex-valued function with a finite domain G that is characterized by a function $f' : G \to \mathbb{Z}/m\mathbb{Z} \cup \{\mathrm{i}\infty\}$ such that $f(x) = \omega_m^{f'(x)}$ for all x with $f'(x) \in \mathbb{Z}/m\mathbb{Z}$ and $f(x) = 0$ if $f'(x) = \mathrm{i}\infty$. Then there is an efficient algorithm for creating the superposition $\sum_x f(x)|x\rangle$ with success probability equal to the fraction of $x \in G$ with $f(x)$ nonzero and that uses two queries to the function f'.*

Proof. Start with the superposition over all $x \in G$: $\sum_x |x, 0\rangle$. Compute $f'(x)$ into the second register and measure to see whether $f'(x) \neq \mathrm{i}\infty$. This succeeds with probability equal to the fraction of x such that $f(x)$ is nonzero. If successful, we are left with a superposition over all x such that $f(x)$ is nonzero. Next, compute the phase shift $\omega_m^{f'(x)}$ by adding $\bmod m$ the value $f'(x)$ to the superposition $\sum_j \omega_m^{-j}|j\rangle$ (which by itself is the Fourier transform over $\mathbb{Z}/m\mathbb{Z}$ of the state $|-1\rangle$), such that

$$\begin{aligned}\sum_{x\in G, f(x)\neq 0} |x\rangle \otimes \frac{1}{\sqrt{m}} \sum_{j\in\mathbb{Z}/m\mathbb{Z}} \omega_m^{-j}|j\rangle &\mapsto \sum_{x\in G, f(x)\neq 0} |x\rangle \otimes \frac{1}{\sqrt{m}} \sum_{j\in\mathbb{Z}/m\mathbb{Z}} \omega_m^{-j}|j + f'(x)\rangle \\ &= \sum_{x\in G, f(x)\neq 0} \omega_m^{f'(x)}|x\rangle \otimes \frac{1}{\sqrt{m}} \sum_{j\in\mathbb{Z}/m\mathbb{Z}} \omega_m^{-j}|j\rangle \\ &= \sum_{x\in G} f(x)|x\rangle \otimes \frac{1}{\sqrt{m}} \sum_{j\in\mathbb{Z}/m\mathbb{Z}} \omega_m^{-j}|j\rangle.\end{aligned}$$

If necessary, the state $\sum_j \omega_m^{-j}|j\rangle = \mathcal{F}|-1\rangle$ can be approximated arbitrarily closely (see [33, section 5.1]). □

2.3. The Fourier transform and approximate Fourier sampling. Although it is not known how to efficiently compute the quantum Fourier transform over $\mathbb{Z}/n\mathbb{Z}$ exactly, it is known how to efficiently approximate such transformations [12, 23, 28, 29]. The current section deals with the problem of approximating the right probability distribution induced by the Fourier transform if the size of the group is not known. This result will be used in section 5.2.

Fourier sampling a quantum state is the process of computing the Fourier transform and measuring the resulting state. The best-known example is Shor's factoring algorithm which finds the period of a function f defined on $\mathbb{Z}$. In that case the function f is periodic with period r and is injective in $\{0, \ldots, r-1\}$. By evaluating the function in superposition up to some chosen value q and measuring the function value, the state $|\phi\rangle = \sqrt{\frac{r}{q}} \sum_{i=0}^{q/r-1} |k + ir\rangle$ is created, where k depends on which function value was measured. If q were a multiple of r, then Fourier sampling $|\phi\rangle$ would result in a random integer multiple of q/r.

When a multiple of r is not known, we must understand the distribution induced by Fourier sampling $|\phi\rangle$ for values of q that we choose. This understanding was at the

heart of Shor's factoring algorithm when he showed that it is possible to still compute multiples of q/r using continued fractions. This principle has been generalized for arbitrary states $|\phi\rangle$ and is known as *approximate* Fourier sampling [23]. This process, described below, allows one to sample from a distribution which is close to the one that could be generated if a multiple of r were known. This will be required in section 5.2 for finding the period of the shifted character problem. In that case the shifted character problem will have a property very different from that of Shor's periodic function. In particular, the periodic function f in Shor's case takes r different values, whereas there are nontrivial cases of the shifted character problem when the function only takes *two* values (ignoring zero amplitudes, which appear in an exponentially small fraction of the amplitudes and thus are insignificant).

Approximate Fourier sampling works as follows: Let $|\phi\rangle = \sum_{x=0}^{n-1} \phi_x |x\rangle$ be an arbitrary superposition, and let $\hat{\mathcal{D}}_{|\phi\rangle}$ be the distribution induced by Fourier sampling $|\phi\rangle$ over $\mathbb{Z}/n\mathbb{Z}$. Let the superposition $|\tilde{\phi}\rangle = \sum_{x=0}^{q'-1} \phi_{x \bmod n} |x\rangle$ be $|\phi\rangle$ repeated until some arbitrary integer q', not necessarily a multiple of n. Let $\hat{\mathcal{D}}_{|\tilde{\phi}\rangle}$ be the distribution induced by Fourier sampling $|\tilde{\phi}\rangle$ over $\mathbb{Z}/q\mathbb{Z}$, where $q > q'$ and $\phi_x = 0$ if $x \geq q'$.

Since $\hat{\mathcal{D}}_{|\phi\rangle}$ is a distribution on $\mathbb{Z}/n\mathbb{Z}$ and $\hat{\mathcal{D}}_{|\tilde{\phi}\rangle}$ is a distribution on $\mathbb{Z}/q\mathbb{Z}$, we define new distributions over fractions which can be compared. Define $\hat{\mathcal{D}}^{\mathrm{RF}}_{|\phi\rangle}(j,k) = \hat{\mathcal{D}}_{|\phi\rangle}(jm)$ if $mk = n$. The distribution $\hat{\mathcal{D}}^{\mathrm{RF}}_{|\phi\rangle}$ is the distribution on the reduced fractions of $\hat{\mathcal{D}}_{|\phi\rangle}$ since it describes the process of sampling x from $\hat{\mathcal{D}}_{|\phi\rangle}$ and returning the fraction x/n in lowest terms.

Let $\hat{\mathcal{D}}^{\mathrm{CF}}_{|\tilde{\phi}\rangle}$ be the distribution induced on fractions from sampling $\hat{\mathcal{D}}_{|\tilde{\phi}\rangle}$ to obtain x, and then using continued fractions to compute the closest approximation to x/q with denominator at most n. It is a theorem that if $q' = \Omega(\frac{n^2}{\epsilon^2})$ and $q = \Omega(\frac{q'}{\epsilon})$, then $|\hat{\mathcal{D}}^{\mathrm{RF}}_{|\phi\rangle} - \hat{\mathcal{D}}^{\mathrm{CF}}_{|\tilde{\phi}\rangle}|_1 < \epsilon$ [23].

It is easy to apply this to the periodic function with period r that is injective on $\{0, \ldots, r-1\}$. Let $n = r$ and consider the state $|\phi\rangle = \sum_{i=1}^{r} |k\rangle$. The distribution $\hat{\mathcal{D}}^{\mathrm{RF}}_{|\phi\rangle}$ is uniform over $\{0, 1/r, 2/r, \ldots, (r-1)/r\}$. By the theorem, if a large enough value q is chosen, then $\hat{\mathcal{D}}^{\mathrm{CF}}_{|\tilde{\phi}\rangle}$ will be ϵ-close to this and efficiently computable.

2.4. Legendre symbol and Jacobi symbol. The Legendre symbol $\left(\frac{\cdot}{p}\right) : \mathbb{F}_p \to \{0, \pm 1\}$ is a quadratic multiplicative character of $\mathbb{F}_p$ defined by

$$\left(\frac{x}{p}\right) = \begin{cases} 0 & \text{if } x = 0, \\ +1 & \text{if } x \text{ is a nonzero square in } \mathbb{F}_p, \\ -1 & \text{if } x \text{ is not a square in } \mathbb{F}_p. \end{cases}$$

The Legendre symbol satisfies $\left(\frac{x}{p}\right) = x^{(p-1)/2} \bmod p$, which shows that we can efficiently compute the Legendre symbol using repeated squaring mod$p$.

The Jacobi symbol $\left(\frac{\cdot}{n}\right) : \mathbb{Z}/n\mathbb{Z} \to \{0, \pm 1\}$ is a quadratic multiplicative character of $\mathbb{Z}/n\mathbb{Z}$ with n an odd integer. It is defined so that it satisfies the relation $\left(\frac{a}{bc}\right) = \left(\frac{a}{b}\right)\left(\frac{a}{c}\right)$ and reduces to the Legendre symbol when the lower parameter is prime. With $n = p_1^{r_1} \cdots p_k^{r_k}$ and all p_i odd primes, this gives the definition $\left(\frac{x}{n}\right) = \left(\frac{x}{p_1}\right)^{r_1} \cdots \left(\frac{x}{p_k}\right)^{r_k}$ such that $\left(\frac{x}{n}\right) \neq 0$ if and only if $x \in \mathbb{Z}/n\mathbb{Z}^*$. The value of the Jacobi symbol can be calculated efficiently without factoring n using the *quadratic reciprocity theorem,* which states that $\left(\frac{m}{n}\right) = (-1)^{(m-1)(n-1)/4}\left(\frac{n}{m}\right)$ in combination with the rule that $\left(\frac{m}{n}\right) = \left(\frac{m'}{n}\right)$ if $m = m' \bmod n$.

2.5. Finite fields. The elements of a finite field $\mathbb{F}_q$ (where $q = p^r$ for some prime p) can be represented as polynomials in $\mathbb{F}_p[X]$ modulo a degree r irreducible polynomial in $\mathbb{F}_p[X]$. In this representation, addition, subtraction, multiplication, and division can all be performed in $O((\log q)^2)$ time [2].

We will need to compute the Fourier transform over the additive group of a finite field, which is isomorphic to $(\mathbb{Z}/p\mathbb{Z})^r$. The additive characters are of the form $\psi_y(x) = \omega_p^{\mathrm{Tr}(xy)}$, where $\mathrm{Tr} : \mathbb{F}_q \to \mathbb{F}_p$ is the trace of the finite field $\mathrm{Tr}(x) = \sum_{j=0}^{r-1} x^{p^j}$ and $y \in \mathbb{F}_q$ [30]. We can efficiently compute the Fourier transform over the additive group of a finite field. (The efficiency of this transform was independently shown in [4].)

An operation U approximates U' to within ϵ if for any unit vector $|\psi\rangle$, $||U|\psi\rangle - U'|\psi\rangle||_2 \leq \epsilon$.

LEMMA 2.2 (Fourier transform over $\mathbb{F}_q$). *The Fourier transform*

$$|x\rangle \mapsto \frac{1}{\sqrt{q}} \sum_{y \in \mathbb{F}_q} \omega_p^{\mathrm{Tr}(xy)} |y\rangle$$

for all $x \in \mathbb{F}_q$ can be approximated to within error ϵ in time polynomial in $\log q$ and $\log 1/\epsilon$.

Proof. Let $q = p^r$, where p is the prime number that denotes the base field: $\mathbb{F}_q = \mathbb{F}_p[X]/f(X)$, with $f(X)$ an irreducible polynomial of degree r. Assume that the mapping $|x\rangle \mapsto \bigotimes_{j=0}^{r-1} |\mathrm{Tr}(xX^j)\rangle$ can be computed in polynomial time. First apply this map and then compute the Fourier transform over $(\mathbb{Z}/p\mathbb{Z})^r$. This gives us the final state

$$\bigotimes_{j=0}^{r-1} \frac{1}{\sqrt{p}} \sum_{y_j \in \mathbb{F}_p} \omega_p^{\mathrm{Tr}(xX^j)y_j} |y_j\rangle = \frac{1}{\sqrt{q}} \sum_{y \in \mathbb{F}_q} \omega_p^{\mathrm{Tr}(xy)} |y\rangle.$$

We first show that the map $|x\rangle \mapsto |\mathrm{Tr}(x), \mathrm{Tr}(xX), \ldots, \mathrm{Tr}(xX^{r-1})\rangle$ is reversible and then that it can be computed in polynomial time. Let $T(x) = (\mathrm{Tr}(x), \mathrm{Tr}(xX), \ldots, \mathrm{Tr}(xX^{r-1}))$. T is additive since Tr is; thus if $T(a) = T(b)$, then $T(a-b)$ is the zero vector. If T is not one-to-one, there is a nonzero x with $T(x)$ equal to the zero vector. Since Tr is not the zero map, choose $a \in \mathbb{F}_q$ such that $\mathrm{Tr}(a) \neq 0$. Choose elements of the base field $z_0, \ldots, z_{r-1}$ such that $x \cdot \sum_j z_j X^j = a$ (these must exist because x is nonzero). Then $\mathrm{Tr}(a) = \mathrm{Tr}(\sum_j z_j x X^j) = \sum_j z_j \mathrm{Tr}(xX^j) = 0$, since $\mathrm{Tr}(xX^j) = 0$ for all j. But this contradicts $\mathrm{Tr}(a) \neq 0$. Thus T is one-to-one.

We now show that the map is computable in polynomial time. Write $x = \sum_{j=0}^{r-1} x_j X^j$, where the x_j are from the base field of $\mathbb{F}_q$. Then for the trace $\mathrm{Tr}(xX^k) = \sum_{j=0}^{r-1} x_j \mathrm{Tr}(X^{j+k})$; hence the components of $T(x)$ are linear combinations of the x_js and thus can be computed in polynomial time.

So far we have assumed that all operations are performed exactly. As we observed earlier in Lemma 2.1 we can approximate the powers of ω_p to within ϵ in time polynomial in $\log p$ and $\log 1/\epsilon$. The Fourier transform over $(\mathbb{Z}/p\mathbb{Z})^r$ can also be approximated to within ϵ in time polynomial in $\log p^r$ and $\log 1/\epsilon$. ∎

For clarity of exposition we assume throughout the rest of the paper that the Fourier transform over $\mathbb{F}_q$ can be performed exactly, as we can make the errors due to the approximation exponentially small with only polynomial overhead.

2.6. Multiplicative characters and their Fourier transforms. The multiplicative group $\mathbb{F}_q^*$ of a finite field $\mathbb{F}_q$ is cyclic. Let g be a generator of $\mathbb{F}_q^*$. Then the

multiplicative characters of $\mathbb{F}_q$ are of the form $\chi(g^\ell) = \omega_{q-1}^{k\ell}$ for all $\ell \in \{0, \ldots, q-2\}$, where the $q-1$ different multiplicative characters are indexed by $k \in \{0, \ldots, q-2\}$. The trivial character is the character with $k = 0$. We can extend the definition of χ to $\mathbb{F}_q$ by defining $\chi(0) = 0$. On a quantum computer we can efficiently compute $\chi(x)$ because the value is determined by the discrete logarithm $\log_g(x)$, which can be computed efficiently using Shor's algorithm [34]. The Fourier transform of a multiplicative character χ of the finite field $\mathbb{F}_q$ is given by $\hat{\chi}(0) = 0$, and $\hat{\chi}(y) = \overline{\chi(y)}\hat{\chi}(1) = \omega_{q-1}^{-k\ell} \sum_j \omega_{q-1}^{kj} \omega_p^{\mathrm{Tr}(g^j)}$, where $y = g^\ell$ [30, 36].

Let $n = p_1^{m_1} \cdots p_k^{m_k}$ be the prime factorization of n. Then by the Chinese remainder theorem, $(\mathbb{Z}/n\mathbb{Z})^* \cong (\mathbb{Z}/p_1^{m_1}\mathbb{Z})^* \times \cdots \times (\mathbb{Z}/p_k^{m_k}\mathbb{Z})^*$. Every multiplicative character χ of $\mathbb{Z}/n\mathbb{Z}$ can be written as the product $\chi(x) = \chi_1(x_1) \cdots \chi_k(x_k)$, where χ_i is a multiplicative character of $\mathbb{Z}/p_i^{m_i}\mathbb{Z}$ and $x_i \equiv x \bmod p_i^{m_i}$. We say χ is *completely nontrivial* if each of the χ_i is nontrivial. We extend the definition of χ to all of $\mathbb{Z}/n\mathbb{Z}$ by defining $\chi(y) = 0$ if $\gcd(y, n) \neq 1$. The character χ is aperiodic on $\{0, \ldots, n-1\}$ if and only if all its χ_i factors are aperiodic over their respective domains $\{0, \ldots, p_i^{m_i} - 1\}$. We call χ a *primitive character* if it is completely nontrivial and aperiodic. Hence, χ is primitive if and only if all its χ_i terms are primitive.

If χ is primitive, the Fourier transform of χ is the product of the Fourier transform of its components and has an expression analogous to the Fourier transform of a multiplicative transform of a finite field. That is,

$$\begin{aligned}\hat{\chi}(y) &= \hat{\chi}_1(y_1) \cdots \hat{\chi}_k(y_k) \\ &= \overline{\chi_1(y_1)}\hat{\chi}_1(1) \cdots \overline{\chi_k(y_k)}\hat{\chi}_k(1) \\ &= \overline{\chi_1(y_1) \cdots \chi_k(y_k)}\hat{\chi}_1(1) \cdots \hat{\chi}_k(1) \\ &= \overline{\chi}(y)\hat{\chi}(1).\end{aligned}$$

If χ is completely nontrivial but periodic with period ℓ, let χ' be the primitive character of $\mathbb{Z}/\ell\mathbb{Z}$ given by $\chi'(x) = \chi(x)$ for $x \in \{0, \ldots, \ell - 1\}$. The Fourier transform of χ is then given by

$$\hat{\chi}(y) = \begin{cases} 0 & \text{if } n/\ell \nmid y, \\ \hat{\chi}'(y\ell/n) = \overline{\chi'(y\ell/n)}\hat{\chi}'(1) & \text{if } n/\ell \mid y. \end{cases}$$

See the book by Tolimieri, An, and Lu [36] for details.

3. The intuition behind the algorithms for the hidden shift problem. We give some intuition for the ideas behind our algorithms for the hidden shift problem. We use the shifted Legendre symbol problem as our running example, but the approach works more generally. In the shifted Legendre symbol problem we are given a function $f_s : \mathbb{Z}/p\mathbb{Z} \to \{0, \pm 1\}$ such that $f_s(x) = \left(\frac{x+s}{p}\right)$, and are asked to find s.

The algorithm starts by putting the function value in the phase to get $|f_s\rangle = \sum_x f_s(x)|x\rangle = \sum_x \left(\frac{x+s}{p}\right)|x\rangle$. For random f we can expect the functions f_z to be mutually (near) orthogonal, so that the inner product squared $|\langle f_z|f_s\rangle|^2$ approximates the delta function $\delta_s(z)$. The Legendre sequence $\left(\frac{0}{p}\right), \left(\frac{1}{p}\right), \ldots, \left(\frac{p-1}{p}\right)$ has many pseudo-random properties, and for its autocorrelation we have, in fact, $|\langle f_z|f_s\rangle|^2 = \delta_s(z) - \frac{1}{p}$. With this, we define the (near) unitary matrix C, where the zth row is $|f_{-z}\rangle$. The state $|f_s\rangle$ is one of the rows; hence $C|f_s\rangle = |-s\rangle$. The problem then reduces to the following: How do we efficiently implement C? By definition, C is a circulant

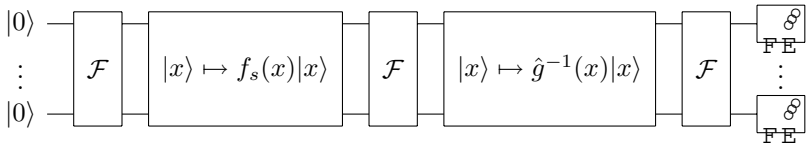


FIG. 3.1. *Circuit for hidden shift problem for a known function g and an unknown shift s of the black-box $f_s(x) = g(x+s)$. Notice that the function values of f_s and $\hat{g}^{-1}$ are computed into the phase.*

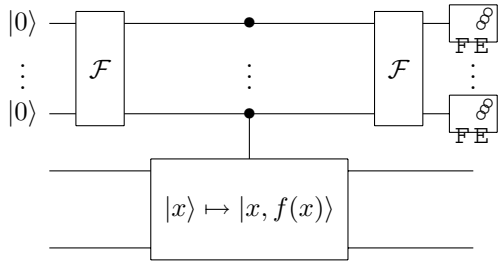


FIG. 3.2. *Circuit for hidden subgroup problem. Here f is computed into a register.*

matrix ($c_{x,y} = f_{-x}(y) = f_0(y-x) = f_{-(x+1)}(y+1) = c_{x+1,y+1}$). Since the Fourier transform matrix diagonalizes a circulant matrix, we can write $C = \mathcal{F}(\mathcal{F}^{-1}C\mathcal{F})\mathcal{F}^{-1} = \mathcal{F}D\mathcal{F}^{-1}$, where D is diagonal. Thus we can implement C if we can implement D. The vector on the diagonal of D is the vector $\mathcal{F}^{-1}|f_0\rangle = \mathcal{F}^{-1}\sum_x \left(\frac{x}{p}\right)|x\rangle$, the inverse Fourier transform of the Legendre symbol. The Legendre symbol is an eigenvector of the Fourier transform, so the diagonal matrix contains the values of the Legendre symbol times a global constant that can be ignored. Because the Legendre symbol can be computed efficiently classically, it can be computed into the phase, so C can be implemented efficiently.

In summary, to implement C for the hidden shift problem for the Legendre symbol, compute the Fourier transform, compute $\left(\frac{x}{p}\right)$ into the phase at $|x\rangle$, and then compute the Fourier transform again (it is not important whether we use $\mathcal{F}$ or $\mathcal{F}^{-1}$). Figure 3.1 shows a circuit diagram outlining the algorithm for the hidden shift problem for a general function g. Contrast this with the circuit for the hidden subgroup problem shown in Figure 3.2.

4. Shifted multiplicative characters of finite fields. In this section we show how to solve the hidden shift problem for any nontrivial multiplicative character of a finite field. The Fourier transform we use is the Fourier transform over the additive group of the finite field.

DEFINITION 4.1 (shifted multiplicative character problem over $\mathbb{F}_q$). *Given a nontrivial multiplicative character $\chi : \mathbb{F}_q \to \mathbb{C}$ (where $q = p^r$ for some prime p) and a black-box function f for which there is an s such that $f(x) = \chi(x+s)$ for all x, find s.*

ALGORITHM 1 (shifted multiplicative character problem over finite field $\mathbb{F}_q$).

1. Create $\sum_{x\in\mathbb{F}_q} \chi(x+s)|x\rangle$, using Lemma 2.1.

2. Compute the Fourier transform to obtain the state $\sum_{y\in\mathbb{F}_q}\omega_p^{\mathrm{Tr}(-sy)}\hat{\chi}(y)|y\rangle$, using Lemma 2.2.
3. For all $y\neq 0$, compute $\chi(y)$ into the phase to obtain $\hat{\chi}(1)\sum_{y\in\mathbb{F}_q^*}\omega_p^{\mathrm{Tr}(-sy)}|y\rangle$.
4. Compute the inverse Fourier transform and measure the outcome $-s$.

THEOREM 4.2. *For any finite field $\mathbb{F}_q$ and any nontrivial multiplicative character, Algorithm* 1 *solves the shifted multiplicative character problem over finite fields with probability* $(1-1/q)^2$.

Proof.

1. Since $\chi(x)=0$ only at $x=0$, by Lemma 2.1 we can create the superposition with probability $1-1/q$.
2. By Lemma 2.2 we can compute the Fourier transform efficiently. The Fourier transform moves the shift s into the phase as described.
3. Because $\hat{\chi}(y)=\overline{\chi(y)}\hat{\chi}(1)$ for every nonzero y, the phase change $|y\rangle\mapsto\chi(y)|y\rangle$ establishes the required transformation.
4. The amplitude of $|-s\rangle$ is

$$\frac{1}{\sqrt{q}}\frac{1}{\sqrt{q-1}}\sum_{y\in\mathbb{F}_q^*}\omega_p^{\mathrm{Tr}(-sy)}\omega_p^{\mathrm{Tr}(sy)}=\frac{1}{\sqrt{q}}\frac{1}{\sqrt{q-1}}\sum_{y\in\mathbb{F}_q^*}1$$
$$=\sqrt{\frac{q-1}{q}},$$

and thus the probability of measuring $-s$ is $1-1/q$. ∎

4.1. The Legendre symbol and homomorphic encryption. The quantum algorithm of the previous section showed us how we can determine the shift $s\in\mathbb{F}_p$ given the function $f_s(x)=\left(\frac{x+s}{p}\right)$. We now show how this algorithm enables us to break schemes for "algebraically homomorphic encryption."

A cryptosystem is *algebraically homomorphic* [9] if given the encryption of two plaintexts $E(x)$, $E(y)$ with $x,y\in\mathbb{F}_p$, an untrusted party can construct the encryption of the plaintexts $E(x+y)$ and $E(xy)$ in polynomial time. More formally, we have the secret encryption and decryption functions $E:\mathbb{F}_p\to S$ and $D:S\to\mathbb{F}_p$, in combination with the public add and multiplication transformations $A:S^2\to S$ and $M:S^2\to S$ such that $D(A(E(x),E(y)))=x+y$ and $D(M(E(x),E(y)))=xy$ for all $x,y\in\mathbb{F}_p$. We assume that the functions E, D, A, and M are deterministic. This definition is slightly more general than the definition in [9, Definition 4.1] because we require equality between texts after unencryption rather than equality between encrypted texts. In other words, the decryption function may be many-to-one. As a result the encryption of a given number can vary depending on how the number is constructed. For example, $A(E(4),E(2))$ may not be equal to $M(E(2),E(3))$. In addition to the public A and M functions, we also assume the existence of a public zero tester $Z:S\to\{0,1\}$, with $Z(E(x))=0$ if $x=0$, and $Z(E(x))=1$ otherwise. In [9] the existence of a zero tester is trivial because the decryption function is injective.

An algebraically homomorphic cryptosystem is a cryptographic primitive that enables two players to perform noninteractive secure function evaluation. It is an open problem whether or not such a cryptosystem can be constructed. We say we can break such a cryptosystem if, given $E(s)$, we can recover s in time polylog(p) with the help of the public functions A, M, and Z. The best known classical attack, due to Boneh and Lipton [9], has expected running time $O\left(\exp\left(c\sqrt{\log p\log\log p}\right)\right)$ for the field $\mathbb{F}_p$ and is based on a smoothness assumption.

Suppose we are given the ciphertext $E(s)$. Test $E(s)$ using the Z function. If s is not zero, create the encryption $E(1)$ via the identity $x^{p-1} \equiv 1 \bmod p$, which holds for all nonzero x. In particular, using $E(s)$ and the M function, we can use repeated squaring and compute $E(s)^{p-1} = E(1)$ in $\log p$ steps.

Clearly, from $E(1)$ and the A function we can construct $E(x)$ for every $x \in \mathbb{F}_p$. Then, given such an $E(x)$, we can compute $f(x) = \left(\frac{x+s}{p}\right)$ in the following way. Add $E(s)$ and $E(x)$, yielding $E(x+s)$, and then compute the encrypted $(p-1)/2$th power of $x+s$, giving $E(\left(\frac{x+s}{p}\right))$. Next, add $E(0)$, $E(-1)$, or $E(1)$ and test if it is an encryption of zero, and return 0, 1, or -1 accordingly. Applying this method on a superposition of $|x\rangle$ states, we can create (after reversibly uncomputing the garbage of the algorithm) the state $\frac{1}{\sqrt{p-1}} \sum_x f_s(x)|x\rangle$. We can then recover s by using Algorithm 1.

COROLLARY 4.3. *Given an efficient test to decide if a value is an encryption of zero, Algorithm* 1 *can be used to break any algebraically homomorphic encryption system.*

We can also break algebraically homomorphic cryptosystems using Shor's discrete log algorithm as follows. Suppose g is a generator for $\mathbb{F}_p^*$ and that we are given the unknown ciphertext $E(g^s)$. Create the superposition $\sum_{i,j} |i, j, E(g^{si+j})\rangle$ and then append the state $|\psi_{si+j}\rangle = \sum_t \left(\frac{g^{si+j}+t}{p}\right)|t\rangle$ to the superposition in i, j by the procedure described above. Next, uncompute the value $E(g^{si+j})$, which gives $\sum_{i,j} |i, j\rangle|\psi_{si+j}\rangle$. Rewriting this as $\sum_{i,r} |i, r - si\rangle|\psi_r\rangle$ and observing that the ψ_r are almost orthogonal, we see that we can apply the methods used in Shor's discrete log algorithm to recover s and thus g^s.

5. Shifted multiplicative characters of finite rings. In this section we show how to solve the shifted multiplicative character problem for $\mathbb{Z}/n\mathbb{Z}$ for any completely nontrivial multiplicative character of the ring $\mathbb{Z}/n\mathbb{Z}$ and extend this to the case when n is unknown. Unlike in the case for finite fields, the characters may be periodic. Thus the shift may not be unique. The Fourier transform is now the familiar Fourier transform over the additive group $\mathbb{Z}/n\mathbb{Z}$.

5.1. Shifted multiplicative characters of $\mathbb{Z}/n\mathbb{Z}$ for known n. We start with the following definition.

DEFINITION 5.1 (shifted multiplicative character problem over $\mathbb{Z}/n\mathbb{Z}$). *Given χ, a completely nontrivial multiplicative character of $\mathbb{Z}/n\mathbb{Z}$, and a function f for which there is an s such that $f(x) = \chi(x + s)$ for all x, find all t satisfying $f(x) = \chi(x + t)$ for all x.*

Multiplicative characters of $\mathbb{Z}/n\mathbb{Z}$ may be periodic, so to solve the shifted multiplicative character problem we first find the period and then we find the shift. If the period is ℓ, then the possible shifts will be $\{s, s + \ell, s + 2\ell, \dots\}$. Note that step 1 of Algorithm 2, which computes the period of χ, uses different properties of a periodic function than Shor's algorithm. In particular, when χ is the Legendre symbol, the function takes only three values, whereas Shor's algorithm assumes functions are injective in $\{0, \dots, r - 1\}$ when the period is r.

ALGORITHM 2 (shifted multiplicative character problem over $\mathbb{Z}/n\mathbb{Z}$).

1. Find the period ℓ of χ. Let χ' be χ restricted to $\{0, \dots, \ell - 1\}$.
   (a) Create $\sum_{x=0}^{n-1} \chi(x + s)|x\rangle$ using f.
   (b) Compute the Fourier transform over $\mathbb{Z}/n\mathbb{Z}$ to obtain the superposition $\sum_{y=0}^{\ell-1} \omega_\ell^{-sy} \hat{\chi}'(y)|yn/\ell\rangle$.
   (c) Measure $|yn/\ell\rangle$. Compute $n/\ell = \gcd(n, yn/\ell)$.

2. Find s using the period ℓ and χ'.
   (a) Create $\sum_{x=0}^{\ell-1} \chi'(x+s)|x\rangle$.
   (b) Compute the Fourier transform over $\mathbb{Z}/\ell\mathbb{Z}$ to obtain $\sum_y \omega_\ell^{-sy}\hat{\chi}'(y)|y\rangle$.
   (c) For all y coprime to ℓ, compute $\hat{\chi}'(y)^{-1}$ into the phase to obtain $\sum_{y:\hat{\chi}'(y)\neq 0} \omega_\ell^{-sy}|y\rangle$.
   (d) Compute the inverse Fourier transform and measure.

THEOREM 5.2. *Algorithm 2 solves the shifted multiplicative character problem over $\mathbb{Z}/n\mathbb{Z}$ for completely nontrivial multiplicative characters of $\mathbb{Z}/n\mathbb{Z}$ in polynomial time with probability at least $(\phi(n)/n)^3 = \Omega((\frac{1}{\log\log n})^3)$, where ϕ is Euler's totient function; $\phi(n)$ is the number of positive integers less than n that are coprime to n.*

Proof. Note that because χ is completely nontrivial, χ' is a primitive character of $\mathbb{Z}/\ell\mathbb{Z}$.

1. (a) $\chi(x+s)$ is nonzero exactly when $\gcd(x+s,n)=1$; thus by Lemma 2.1 we can create the superposition with probability $\phi(n)/n$.
   (b) Since χ has period ℓ, the Fourier transform is nonzero only on multiples of n/ℓ.
   (c) Since $\hat{\chi}'(y) = \overline{\chi'(y)}\hat{\chi}'(1)$, and $\chi'(y)$ is nonzero precisely when $\gcd(y,n) = 1$, when we measure yn/ℓ we have $n/\ell = \gcd(n, yn/\ell)$.
2. (a) Similar to the argument above, we can create the superposition with probability $\phi(\ell)/\ell$.
   (b) The Fourier transform moves the shift s into the phase.
   (c) As in the case for the finite field, this can be done by computing the phase of $\chi'(y)$ into the phase of $|y\rangle$.
   (d) Let $A = \{y \in \mathbb{Z}/\ell\mathbb{Z} : \hat{\chi}'(y) \neq 0\}$. $A = (\mathbb{Z}/\ell\mathbb{Z})^*$ and thus $|A| = \phi(\ell)$. Then the amplitude of $|-s\rangle$ after the Fourier transform is

$$\frac{1}{\sqrt{\phi(\ell)}}\frac{1}{\sqrt{\ell}}\left(\sum_{y\in A}\omega_\ell^{-ys}\,\omega_\ell^{ys}\right) = \frac{1}{\sqrt{\phi(\ell)}}\frac{1}{\sqrt{\ell}}\left(\sum_{y\in A}1\right) = \sqrt{\frac{\phi(\ell)}{\ell}}.$$

   Hence the probability of measuring $|-s\rangle$ is $\phi(\ell)/\ell$.

Thus the algorithm succeeds with probability $(\phi(n)/n)(\phi(\ell)/\ell)^2$, which is lower bounded by $\Omega((\frac{1}{\log\log n})^3)$ (because of the bound $\phi(n) = \Omega(n/\log\log n)$). ∎

5.2. Shifted multiplicative characters of $\mathbb{Z}/n\mathbb{Z}$ for unknown n. We now consider the case when n is unknown.

DEFINITION 5.3 (shifted multiplicative character problem over $\mathbb{Z}/n\mathbb{Z}$ with unknown n). *Given a completely nontrivial multiplicative character $\chi : \mathbb{Z}/n\mathbb{Z} \to \mathbb{C}$ for some unknown n, and a function f for which there is an s such that $f(x) = \chi(x+s)$ for all x, find all t satisfying $f(x) = \chi(x+t)$ for all x.*

THEOREM 5.4. *Given an upper bound on the size of the period of f, we can efficiently solve the shifted multiplicative character problem over $\mathbb{Z}/n\mathbb{Z}$ for unknown n on a quantum computer.*

Proof. Let ℓ be the period of f and χ' be χ restricted to $\mathbb{Z}/\ell\mathbb{Z}$. Using the Fourier sampling algorithm described in section 2.3, we can approximately Fourier sample f over $\mathbb{Z}/\ell\mathbb{Z}$. Because $\chi'(y)$ is nonzero precisely when $\gcd(y,\ell) = 1$, this Fourier sampling algorithm returns y/ℓ with high probability, where y is coprime to ℓ. Thus we can find ℓ with high probability. Next, apply Algorithm 2 to find $s \bmod \ell$. ∎

6. The hidden coset problem. In this section we define the hidden coset problem and give an algorithm for solving the problem for abelian groups under certain conditions on the functions. As we will show, this problem abstracts out two properties that appeared in the hidden shift problems in earlier sections. We will also show how finding the coset representative can be interpreted as solving a deconvolution problem.

DEFINITION 6.1 (hidden coset problem). *Given functions f and g defined on a group G such that for some $s \in G$, $f(x) = g(x+s)$ for all x in G, find the set of all t satisfying $f(x) = g(x+t)$ for all x in G. The function f is given as an oracle, and g is known but not necessarily efficiently computable.*

LEMMA 6.2. *The answer to the hidden coset problem is a coset of some subgroup H of G, and g is constant on cosets of H.*

Note that g is not necessarily a hidden subgroup problem instance because while it is constant on cosets, it does not have to be distinct on different cosets.

Proof. Let $S = \{t \in G : f(x) = g(x+t) \text{ for all } x \in G\}$ be the set of all solutions and let H be the largest subgroup of G such that g is constant on cosets of H. Clearly this is well defined (note that H may be the trivial subgroup as in the shifted Legendre symbol problem). Suppose t_1, t_2 are in S. Then we have $g(x+(-t_2+t_1)) = g((x-t_2)+t_1) = f(x-t_2) = g((x-t_2)+t_2) = g(x)$ for all x in G; thus $-t_2+t_1$ is in H. This shows that S is contained in a coset of H. Since s is in S we must have that S is contained in $s+H$. Conversely, suppose $s+h$ is in $s+H$ (where h is in H). Then $g(x+s+h) = g(x+s) = f(x)$ for all x in G; hence $s+h$ is in S. It follows that $S = s+H$. While this proof was written with additive notation, it carries through if the group is nonabelian. ∎

A familiar example of a nonabelian case of the hidden coset problem is given by the graph isomorphism problem. For each n vertex graph X we let $M(X) \in \{0,1\}^{n\times n}$ denote the adjacency matrix of X, and given X we define the function $g : S_n \to \{0,1\}^{n\times n}$ by $g : \sigma \mapsto M(\sigma(X))$ for all permutations $\sigma \in S_n$ in the symmetric group. Now, the shifted function $f : S_n \to \{0,1\}^{n\times n}$ coincides with a description of the permuted adjacency matrix $M(\pi(X))$ and has $f(\sigma) = M(\sigma \cdot \pi(X))$ for all σ. This shows how the search for an element $\sigma' \in S_n$ such that $f(\sigma) = g(\sigma \cdot \sigma')$ is identical to the search for a permutation that transforms X to $\pi(X)$: the GRAPH ISOMORPHISM problem. Moreover, the determination of the constant subgroup of g of all the permutations σ that have $M(X) = M(\sigma(X))$ solves the GRAPH AUTOMORPHISM problem of the graph X.

Unlike for the hidden subgroup problem, we can prove that there are cases of the hidden coset problem that cannot be solved efficiently on a quantum computer. Let $g : \mathbb{Z}/n\mathbb{Z} \to \{0,1\}$ be the delta function δ_0 with $g(0) = 1$, and $g(x) = 0$ otherwise. Consequently, f_s will be the unknown delta function δ_{-s}, which determines the hidden coset $\{s\}$. However, given f_s and g, finding s amounts to searching a list of n items, which requires $\Omega(\sqrt{n})$ queries to f_s [5].

The algorithm for the hidden coset problem instances that we can solve consists of two parts: identifying the subgroup on which g is constant and finding a coset representative, where computing a coset representative corresponds to computing one hidden shift. The algorithms in this article that compute the subgroup and a coset representative exploit different facets of the power of the quantum Fourier transform. After computing a Fourier transform, the subgroup structure is captured in the magnitude, whereas the shift structure is captured in the phase. In the hidden subgroup problem we measure after computing the Fourier transform and so discard information about shifts. Our algorithms for hidden shift problems do additional processing

to take advantage of the information encoded in the phase. Thus the solution to the hidden coset problem requires fully utilizing the abilities of the Fourier transform.

6.1. Identifying the unknown subgroup. The first step of the hidden coset problem algorithm is to compute the unknown subgroup of g. As the examples in the previous section show, computing the subgroup may be difficult for at two least reasons. First, g may be an HSP instance over a nonabelian group, for which no efficient algorithm is known. Second, while g is constant on cosets, it may not be distinct on different cosets; that is, it may not be an HSP instance.

We start by finding the subgroup H. We need two different algorithms for determining H: the "standard" algorithm for the hidden subgroup problem and the algorithm we used in section 5.

In the standard algorithm for the hidden subgroup problem we form a superposition over all inputs, compute $g(x)$ into a register, measure the function value, compute the Fourier transform, and then sample. The standard algorithm may fail when g is not distinct on different cosets of H. In such cases, we need other restrictions on g to be able to find the hidden subgroup H using the standard algorithm. Boneh and Lipton [8], Mosca and Ekert [32], and Hales and Hallgren [23] have all given criteria under which the standard hidden subgroup algorithm outputs H even when g is not distinct on different cosets of H.

In section 5 we used a different algorithm to determine H because the function we were considering did not satisfy the conditions mentioned above. In this algorithm we compute the value of g into the amplitude, Fourier transform, and then sample, whereas in the standard hidden subgroup algorithm we compute the value of g into a register. In general, this algorithm works when the fraction of values for which $\hat{g}$ is zero is sufficiently small and the nonzero values of $\hat{g}$ have constant magnitude.

6.2. Finding a coset representative as a deconvolution problem. Once we have identified H, we can find a coset representative by solving the associated hidden coset problem for f' and g', where f' and g' are defined on the quotient group G/H and are consistent in the natural way with f and g. For notational convenience we assume that f and g are defined on G and that H is trivial, that is, the shift is uniquely defined.

The hidden shift problem may be interpreted as a *deconvolution* problem. In a deconvolution problem, we are given functions g and $f = g \star h$ (the convolution of g with some unknown function h) and asked to find this h. Let $\delta_y(x) = \delta(x - y)$ be the delta function centered at y. In the hidden shift problem, f is the convolution of δ_{-s} and g, that is, $f = g \star \delta_{-s}$. Finding s or, equivalently, finding δ_{-s}, given f and g, is therefore a deconvolution problem.

Recall that under the Fourier transform convolution becomes pointwise multiplication. Thus, taking Fourier transforms, we have $\hat{f} = \hat{g} \cdot \hat{\delta}_{-s}$ and hence $\hat{\delta}_{-s} = \hat{g}^{-1} \cdot \hat{f}$, provided that $\hat{g}$ is everywhere nonzero. For the multiplication by $\hat{g}^{-1}$ to be performed efficiently on a quantum computer would require $\hat{g}$ to have constant magnitude and be everywhere nonzero. However, even if only a fraction of the values of $\hat{g}$ are zero we can still approximate division of $\hat{g}$ by only dividing when $\hat{g}$ is nonzero and doing nothing otherwise. The zeros of $\hat{g}$ correspond to loss of information about δ_{-s}.

ALGORITHM 3.

1. Create $\sum_{x \in G} g(x + s)|x\rangle$.
2. Compute the Fourier transform to obtain $\sum_{y \in G} \overline{\psi_y(s)} \hat{g}(\psi_y)|y\rangle$, where ψ_y are the characters of the group G.

3. For all y for which $\hat{g}(\psi_y)$ is nonzero compute $\hat{g}(\psi_y)^{-1}$ into the phase of $|y\rangle$ to obtain $\sum_{y,\hat{g}(\psi_y)\neq 0} \overline{\psi_y(s)}|y\rangle$.
4. Compute the inverse Fourier transform and measure to obtain $-s$.

THEOREM 6.3. *Suppose f and $\hat{g}$ are efficiently computable, the magnitude of $f(x)$ is constant for all values of x in G for which $f(x)$ is nonzero, and the magnitude of $\hat{g}(\psi_y)$ is constant for all values of ψ_y in $\hat{G}$ for which $\hat{g}(\psi_y)$ is nonzero. Let α be the fraction of x in G for which $f(x)$ is nonzero and let β be the fraction of ψ_y in $\hat{G}$ for which $\hat{g}(\psi_y)$ is nonzero. Then Algorithm 3 outputs $-s$ with probability $\alpha\beta$.*

Proof.

1. By Lemma 2.1 we can create the superposition with probability α.
2. The Fourier transform moves the shift s into the phase.
3. Because $\hat{g}$ has constant magnitude, for values where $\hat{g}$ is nonzero, $\hat{g}(\psi_y)^{-1} = C\overline{\hat{g}(\psi_y)}$ for some constant C. So we can perform this step by computing the phase of $\overline{\hat{g}}$ into the phase. For the values where $\hat{g}$ is zero we can just leave the phase unchanged as those terms are not present in the superposition.
4. Let $A = \{y \in G : \hat{g}(\psi_y) \neq 0\}$. Then the amplitude of $|-s\rangle$ is

$$\begin{aligned}\frac{1}{\sqrt{|A|}}\frac{1}{\sqrt{|G|}}\left(\sum_{y\in A}\overline{\psi_y(s)}\,\overline{\psi_y(-s)}\right) &= \frac{1}{\sqrt{|A|}}\frac{1}{\sqrt{|G|}}\left(\sum_{y\in A}1\right)\\ &= \sqrt{\frac{|A|}{|G|}}\\ &= \sqrt{\beta}.\end{aligned}$$

Hence we measure $|-s\rangle$ with probability β.

Thus the algorithm succeeds in identifying s with probability $\alpha\beta$ and requires only one query of f and one query of $\hat{g}$. ∎

6.3. Examples. We show how the hidden shift problems we considered earlier fit into the framework of the hidden coset problem. In the shifted multiplicative character problem over finite fields, G is the additive group of $\mathbb{F}_q$, $g = \chi$, and H is trivial since the shift is unique for nontrivial χ. In the shifted multiplicative character problem over $\mathbb{Z}/n\mathbb{Z}$, G is the additive group of $\mathbb{Z}/n\mathbb{Z}$, $g = \chi$, and H is the subgroup $\{0, \ell, \ldots, n/\ell\}$, where ℓ (which is a factor of n) is the period of χ. In the shifted period multiplicative character problem over $\mathbb{Z}/n\mathbb{Z}$ for unknown n, G is the additive group of $\mathbb{Z}$, $g = \chi$, and H is the infinite subgroup $\ell\mathbb{Z}$.

Acknowledgments. We would like to thank the anonymous referee who pointed out the application of the shifted Legendre symbol problem to algebraically homomorphic cryptosystems, and Umesh Vazirani, whose many suggestions greatly improved this paper. We also thank Dylan Thurston and an anonymous referee for pointing out that algebraically homomorphic cryptosystems can be broken using Shor's algorithm for discrete log. Thanks to Lisa Hales for helpful suggestions.

REFERENCES

[1] M. ABADI AND J. FEIGENBAUM, *Secure circuit evaluation. A protocol based on hiding information from an oracle*, J. Cryptology, 2 (1990), pp. 1–12.

[2] E. BACH AND J. SHALLIT, *Algorithmic Number Theory—Efficient Algorithms*, Vol. I, MIT Press, Cambridge, MA, 1996.

[3] R. Beals, *Quantum computation of Fourier transforms over symmetric groups*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, 1997, pp. 48–53.
[4] J. Niel de Beaudrap, R. Cleve, and J. Watrous, *Sharp quantum versus classical query complexity separations*, Algorithmica, 34 (2002), pp. 449–461.
[5] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, *Strengths and weaknesses of quantum computing*, SIAM J. Comput., 26 (1997), pp. 1510–1523.
[6] E. Bernstein and U. Vazirani, *Quantum complexity theory*, SIAM J. Comput., 26 (1997), pp. 1411–1473.
[7] M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudo-random bits*, SIAM J. Comput., 13 (1984), pp. 850–864.
[8] D. Boneh and R. J. Lipton, *Quantum cryptanalysis of hidden linear functions*, in Advances in Cryptology—CRYPTO '95, Lecture Notes in Comput. Sci. 963, Springer-Verlag, Berlin, 1995, pp. 424–437.
[9] D. Boneh and R. J. Lipton, *Algorithms for black-box fields and their application to cryptography*, in Advances in Cryptology—CRYPTO '96, Lecture Notes in Comput. Sci. 1109, Springer-Verlag, Berlin, 1996, pp. 283–297.
[10] R. Cleve, *The query complexity of order-finding*, Inform. and Comput., 192 (2004), pp. 162–171.
[11] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca, *Quantum algorithms revisited*, R. Soc. Lond. Proc. Ser. A Math. Phys. Eng. Sci., 454 (1998), pp. 339–354.
[12] R. Cleve and J. Watrous, *Fast parallel circuits for the quantum Fourier transform*, in Proceedings of the 41st Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 2000, pp. 526–536.
[13] W. van Dam, *Quantum algorithms for weighing matrices and quadratic residues*, Algorithmica, 34 (2002), pp. 413–428.
[14] W. van Dam and S. Hallgren, *Efficient Quantum Algorithms for Shifted Quadratic Character Problems*, http://www.arxiv.org/abs/quant-ph/0011067 (2000).
[15] W. van Dam, S. Hallgren, and L. Ip, *Quantum algorithms for some hidden shift problems*, in Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2003, pp. 489–498.
[16] W. van Dam and G. Seroussi, *Efficient Quantum Algorithms for Estimating Gauss Sums*, http://www.arxiv.org/abs/quant-ph/0207131 (2002).
[17] I. B. Damgård, *On the randomness of Legendre and Jacobi sequences*, in Advances in Cryptology—CRYPTO'88, Lecture Notes in Comput. Sci. 403, Springer-Verlag, Berlin, 1990, pp. 163–172.
[18] M. Ettinger and P. Høyer, *On quantum algorithms for noncommutative hidden subgroups*, Adv. in Appl. Math., 25 (2000), pp. 239–251.
[19] M. Ettinger, P. Høyer, and E. Knill, *Hidden Subgroup States Are Almost Orthogonal*, http://www.arxiv.org/abs/quant-ph/9901034 (1999).
[20] K. Friedl, F. Magniez, M. Santha, and P. Sen, *Quantum testers for hidden group properties*, in Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 2747, Springer-Verlag, Berlin, 2003, pp. 419–428.
[21] M. Grigni, L. Schulman, M. Vazirani, and U. Vazirani, *Quantum mechanical algorithms for the nonabelian hidden subgroup problem*, Combinatorica, 24 (2004), pp. 137–154.
[22] L. Hales, *The Quantum Fourier Transform and Extensions of the Abelian Hidden Subgroup Problem*, Ph.D. thesis, University of California-Berkeley, Berkeley, CA, 2002.
[23] L. Hales and S. Hallgren, *An improved quantum Fourier transform algorithms and applications*, in Proceedings of the 41st Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 2000, pp. 515–525.
[24] S. Hallgren, *Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, 2002, pp. 653–658.
[25] S. Hallgren, A. Russell, and A. Ta-Shma, *The hidden subgroup problem and quantum computation using group representations*, SIAM J. Comput., 32 (2003), pp. 916–934.
[26] L. Ip, *Solving Shift Problems and the Hidden Coset Problem Using the Fourier Transform*, http://www.arxiv.org/abs/quant-ph/0205034 (2002).
[27] G. Ivanyos, F. Magniez, and M. Santha, *Efficient quantum algorithms for some instances of the non-abelian hidden subgroup problem*, Internat. J. Found. Comput. Sci., 14 (2003), pp. 723–739.
[28] A. Yu. Kitaev, *Quantum Measurements and the Abelian Stabilizer Problem*, http://www.arxiv.org/abs/quant-ph/9511026 (1995).

[29] A. Yu. Kitaev, *Quantum computations: Algorithms and error correction*, Russian Math. Surveys, 52 (1997), pp. 1191–1249.
[30] R. Lidl and H. Niederreiter, *Finite Fields*, 2nd ed., Encyclopedia Math. Appl. 20, Cambridge University Press, Cambridge, UK, 1997.
[31] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, Handbook of Applied Cryptology, CRC Press, Boca Raton, FL, 1997.
[32] M. Mosca and A. Ekert, *The hidden subgroup problem and eigenvalue estimation on a quantum computer*, in Proceedings of the 1st NASA International Conference on Quantum Computing and Quantum Communication, Lecture Notes in Comput. Sci. 1509, Springer-Verlag, Berlin, 1999, pp. 174–188.
[33] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, UK, 2000.
[34] P. W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Comput., 26 (1997), pp. 1484–1509.
[35] D. R. Simon, *On the power of quantum computation*, SIAM J. Comput., 26 (1997), pp. 1474–1483.
[36] R. Tolimieri, M. An, and C. Lu, *Algorithms for Discrete Fourier Transform and Convolution*, Springer-Verlag, New York, 1989.
[37] J. Watrous, *Quantum algorithms for solvable groups*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, 2001, pp. 60–67.

SIAM J. COMPUT.
Vol. 36, No. 3, pp. 779–802

TESTING POLYNOMIALS OVER GENERAL FIELDS*

TALI KAUFMAN† AND DANA RON‡

In memory of Bilha Segev (1963–2005)*, a true scientist*

Abstract. In this work we fill the knowledge gap concerning testing polynomials over finite fields. As previous works show, when the cardinality of the field, q, is sufficiently larger than the degree bound, d, then the number of queries sufficient for testing is polynomial or even linear in d. On the other hand, when $q = 2$ then the number of queries, both sufficient and necessary, grows exponentially with d. Here we study the intermediate case where $2 < q = O(d)$ and show a smooth transition between the two extremes. Specifically, let p be the characteristic of the field (so that p is prime and $q = p^s$ for some integer $s \geq 1$). Then the number of queries performed by the test grows like $\ell \cdot q^{2\ell+1}$, where $\ell = \lceil \frac{d+1}{q-q/p} \rceil$. Furthermore, $q^{\Omega(\ell)}$ queries are necessary when $q = O(d)$. The test itself provides a unifying view of the tests for these two extremes: it considers random affine subspaces of dimension ℓ and verifies that the function restricted to the selected subspaces is a polynomial of degree at most d. Viewed in the context of coding theory, our result shows that Reed–Muller codes over general fields (usually referred to as *generalized Reed–Muller (GRM)* codes) are locally testable. In the course of our analysis we provide a characterization of small-weight words that span the code. Such a characterization was previously known only when the field size is a prime or is sufficiently large, in which case the minimum-weight words span the code.

Key words. testing, polynomials, Reed–Muller codes

AMS subject classifications. 68Q25, 68W40, 68W20, 12Y05, 94B05

DOI. 10.1137/S0097539704445615

1. Introduction.

In this paper we consider the problem of testing, for a given finite field F and degree-bound d, whether a function $f : F^n \to F$ is a multivariate polynomial of total degree at most d over F. Specifically, the testing algorithm is given query access to f and a parameter $\epsilon > 0$. If f is a polynomial of degree at most d then the testing algorithm must accept. On the other hand, if f differs from every such polynomial on more than an ϵ-fraction of the domain elements, then the test should reject with probability at least 2/3. Our aim is to design algorithms that perform this task using as few queries as possible.

The problem of testing multivariate low-degree polynomials over finite fields has been studied extensively, mainly due to its applications to probabilistically checkable proofs systems (see, for example, [32, 3]). This is true both for the special case of linear functions (degree-1 polynomials) [15, 7, 18, 10, 11, 9, 31] and for the more general case of degree-d polynomials [7, 6, 20, 18, 30, 19, 4]. However, all these results apply only to testing polynomials over fields that are *larger than the degree-bound, d*. In particular, when the field size $|F|$ is at least $c \cdot d$, for some sufficiently large

*Received by the editors August 7, 2004; accepted for publication (in revised form) May 30, 2006; published electronically October 30, 2006. An extended abstract of this work appeared in the *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2004.

http://www.siam.org/journals/sicomp/36-3/44561.html

†Computer Science and Artificial Intelligence Lab, MIT, Cambridge, MA 02138 (kaufmant@MIT.EDU). This work is part of the author's Ph.D. thesis done at Tel Aviv University under the supervision of Professor Noga Alon and Professor Michael Krivelevich. The research was performed in part while visiting the Radcliffe Institute of Advanced Study at Harvard.

‡Department of Electrical Engineering-Systems, Tel Aviv University, Tel Aviv 69978, Israel (danar@eng.tau.ac.il). This research was done during a fellowship year at the Radcliffe Institute of Advanced Study at Harvard.

constant c, then a number of queries that is linear in d is sufficient [29, 19], and when $d+2 \leq |F| < c \cdot d$ then the dependence on d is known to be polynomial [19, 30]. In recent work, Alon et al. [1] studied the same property for the case $|F| = 2$ and for $d \geq 2$. Namely, they considered the case in which the degree-bound may be (much) larger than the field size, but their result holds only for $F = \mathrm{GF}(2)$. They showed that the number of queries both necessary and sufficient in this case is *exponential* in d. Hence we encounter a very large gap in terms of the dependence on d between the query complexity when $|F| > d$ and the query complexity when $|F| = 2$.

Our main result. In this work we bridge the gap between the two cases mentioned above and show a smooth transition between them. In particular, we prove the following theorem.

THEOREM 1. *There exists a testing algorithm for polynomials of degree at most d over finite fields of cardinality q where $2 \leq q = O(d)$. The algorithm performs $O\left(\ell \cdot q^{2\ell+1} + 1/\epsilon\right)$ queries, where for prime q, $\ell = \left\lceil \frac{d+1}{q-1} \right\rceil$, and, more generally, when q is a power of a prime p, then $\ell = \left\lceil \frac{d+1}{q-q/p} \right\rceil$.*

Observe that as the field size q increases, the dependence on d decreases from being exponential to being polynomial. We note that this query complexity (when $q = O(d)$) is almost tight: for prime fields (and constant ϵ) $\Omega(q^{\ell-1})$ queries are necessary, and for nonprime fields $\Omega\left(q^{\lceil \ell/2 \rceil - 1}\right)$ queries are necessary. As we discuss in more detail subsequently, the "gap phenomenon" that we observe is not unique to testing polynomials: analogous gaps arise in other property testing problems.

Characterization of degree-d polynomials over $\mathrm{GF}(q)$. One of the building blocks of our analysis is a characterization of (total) degree-d multivariate polynomials over finite fields. In particular, we prove the following theorem.

THEOREM 2. *Let $F = \mathrm{GF}(q)$ where $q = p^s$ and p is prime. Let d be an integer, and let $f : F^n \to F$. Then f is a polynomial of degree at most d if and only if its restriction to every affine subspace of dimension $\ell = \left\lceil \frac{d+1}{q-q/p} \right\rceil$ is a polynomial of degree at most d.*

Theorem 2 generalizes the characterization result of Friedl and Sudan [19] that refers to the case $q - q/p \geq d+1$ (that is, $\ell = 1$). We also note that this value, ℓ, of the dimension of the considered subspaces, is tight. Namely, there exist polynomials of degree greater than d whose restrictions to affine subspaces of dimension less than ℓ are all degree-d polynomials.

A unifying approach to testing low-degree polynomials over $\mathrm{GF}(q)^n$. The testing algorithm presented in this work utilizes the characterization in Theorem 2 (which is shown to be *robust* in the sense defined in [30]). Specifically, the algorithm selects random affine subspaces (of dimension ℓ as defined in Theorem 2) and checks that the restriction of the input function f to each of the selected subspaces is indeed a polynomial of degree at most d. Such a check is implemented by verifying that various linear combinations of the values of f on the subspace sum to 0. Observe that when the size of the field F is sufficiently larger than the degree-bound d, then $\ell = 1$. That is, when the field is sufficiently large, then the algorithm checks whether the univariate polynomials that correspond to restrictions of the function f to random *lines* in F^n all have degree at most d. This is essentially the original low-degree test of Rubinfeld and Sudan [30]. The reason we say "essentially" is that when $|F| > d$ then it is not necessary to query f on all points on a selected line, but rather it suffices to interpolate using $d+1$ points and check that the resulting degree-d polynomial agrees with a random point on the line.

On the other hand, when $q = 2$ then the test in [1] works by uniformly selecting

$\frac{d+1}{q-1} = d+1$ points in $\mathrm{GF}(2)^n$ and verifying that the sum of the values of f taken over all sums of subsets of these points is 0. This too can be shown to amount to checking whether the restriction of f to the $(d+1)$-dimensional subspace spanned by the selected points is a polynomial of degree at most d. Thus our test suggests a uniform view of the aforementioned tests for low-degree polynomials.

Relation to coding. The *generalized Reed–Muller (GRM)* code of rank d and length q^n over $\mathrm{GF}(q)$, which we denote by $\mathcal{GRM}_q(d,n)$,[1] consists of all words of length q^n that correspond to the evaluations of degree-d polynomials over $\mathrm{GF}(q)^n$. (When $q=2$ then the code is simply referred to as *Reed–Muller (RM).*) The *weight* of a codeword is the number of its nonzero symbols. An equivalent view of our main result, from a coding theory perspective, is that GRM codes are locally testable. Furthermore, our characterization of low-degree polynomials translates into a characterization of a set of small-weight words that span the dual code, where the dual code of $\mathcal{GRM}_q(d,n)$ is the GRM code $\mathcal{GRM}_q(n(q-1)-(d+1),n)$ (see [5, Theorem 5.4.2]). The question concerning when GRM codes are spanned by their *minimum-weight* words has been studied in the coding theory literature.[2] Specifically, Ding and Key [16] have shown that if q is prime or q is sufficiently larger than d, then the minimum-weight words of a GRM code indeed span the code, but this is not true in general. In particular, this is not true when q is not prime and $q-q/p < d+1$. To be precise, there are two special cases in which the minimum-weight words *do* span the code though q is not prime and $q-q/p<d+1$: the special case of $n=1$ (Reed–Salomon codes) and the special case that d is almost the maximum possible degree $n\cdot(q-1)$.

We complement the result of Ding and Key by deducing the following corollary from Theorem 2.

COROLLARY 3. *Every GRM code is spanned by words of weight that is at most quadratic in the weight of its minimum-weight words.*

We note that our interest in small-weight words that span a GRM code is due to the way our test works. Similarly to other low-degree tests, our test can be viewed as randomly selecting small-weight words from the dual code (which is a GRM code itself) and checking that each is indeed orthogonal to the tested word. Thus the weight of the selected words is an important factor in the query complexity of our algorithm.

The paper of Jutla et al. [25]. Independently from our work, Jutla et al. [25] studied the problem of testing low-degree polynomials and described a testing algorithm that has the same query complexity as our algorithm. However, their algorithm works only for prime fields. We further discuss the relation between our approaches in section 6.

A broader perspective: The gap phenomenon in property testing. Property testing [30, 21] in general deals with distinguishing between objects that have a particular predetermined property and objects that are far from having the property. Low-degree testing clearly falls under this framework. As discussed above, for this property we encounter a huge gap in a different setting of the problem. Interestingly, a similar gap phenomenon is encountered for the property of graph bipartiteness. Specifically, dense graphs can be tested for bipartiteness with complexity $\Theta(1)$ [21], while the complexity

[1] We note that in the coding theory literature, the code is usually denoted by $\mathcal{GRM}_q(r,m)$ (or $\mathcal{R}_q(r,m)$), and n is used to denote the codeword length q^m.

[2] The minimum-weight words of $\mathcal{GRM}_q(d,n)$ have weight $(q-t)q^{n-r-1}$, where $r(q-1)+t=d$ and $0\le t<q-1$, and the points in the support of each such word belong to an affine subspace of dimension $(n-r)$.

of testing bipartiteness in constant-degree graphs is $\Theta(\sqrt{n})$ [23, 22], where n is the number of vertices in the graph. It has been shown [26] that for this property too there is an algorithm that gives a smooth transition between the two extremes, and there is an almost matching lower bound. Additional properties that exhibit similar gaps are k-colorability and subgraph-freeness, where there has been recent progress on the latter problem [2].

Other related results on locally testing codes. The problem of designing good codes that are locally testable was explicitly defined, e.g., in [19, 30, 3]. By *good* we mean that they have high rate and large distance. This question has regained attention recently. Goldreich and Sudan [24] initiated a systematic study of the problem and showed (by probabilistic arguments) that there exist locally testable (linear) codes over binary alphabets, with almost constant rate, and linear minimum distance. Their construction was derandomized in [14] and further improved in [12]. The best locally testable codes known to date are given by Dinur [17]. In [13] it was shown that local testing of random low-density parity-check codes, which have linear minimum distance and constant rate, requires $\Omega(n)$ queries. In [8] it was proved that there are no locally testable cyclic codes that have constant rate and linear minimum distance.

Subsequent work. In recent work, Kaufman and Litsyn [27] provided a sufficient condition for local testability of linear codes. The sufficient condition is related to the weight distribution (spectra) of the code and of its dual. They use the condition to show that every linear code of length n and minimum distance $\frac{n}{2} - \Theta(\sqrt{n})$ is locally testable. Such codes contain polynomially many codewords, and hence they are called sparse codes. Moreover, in [27] it is shown that in such a case the dual code is spanned by small-weight codewords. Their results imply that dual-BCH codes and some subcodes of algebraic-geometric codes are locally testable.

2. Preliminaries. Let F be a field of cardinality q and characteristic p (that is, $q = p^s$ where p is prime). Let $\omega \in F$ be a generator of the field F, so that $F = \{0, \omega^1 = \omega, \omega^2, \ldots, \omega^{q-2}, \omega^{q-1} = 1\}$. We note that this order of the elements in F in which 1 is represented by ω^{q-1} rather than ω^0 will serve us better than the more standard order $F = \{0, \omega^0 = 1, \omega^1 = \omega, \omega^2, \ldots, \omega^{q-2}\}$. In particular, using this order, for any $n \geq 1$ we consider a one-to-one mapping between $[q-1]^n$ and F^n (where $[q-1] \stackrel{\text{def}}{=} \{0, 1, \ldots, q-1\}$). Specifically, for each $\beta \in [q-1]^n$, the point $x \in F^n$ that *corresponds* to β is defined as follows: $x_i = 0$ if $\beta_i = 0$, and otherwise $x_i = \omega^{\beta_i}$.

Consider the standard (and unique) representation of the function f as a polynomial over F with degree at most $q-1$ in each variable:

$$f(x) = \sum_{\alpha \in [q-1]^n} C^f_\alpha \cdot x^\alpha, \qquad \text{where} \qquad x^\alpha = \prod_{i=1}^{n} x_i^{\alpha_i}. \tag{2.1}$$

Here $\vec{C^f}$ is the *coefficients vector* (indexed by points $\alpha \in [q-1]^n$). If we view the function f as a q^n-dimensional vector $\vec{f}$ (where for $\beta \in [q-1]^n$, f_β is the evaluation of f on the point $x \in F^n$ that corresponds to β), then we can write (2.1) in the following equivalent form:

$$\vec{f} = \mathcal{H}_n \cdot \vec{C^f}.$$

Here $\mathcal{H}_n$ is the $q^n \times q^n$ matrix whose entries are indexed by pairs $\beta, \alpha \in [q-1]^n$ where $\mathcal{H}_n(\beta, \alpha)$ is the evaluation of the term x^α at the point $x \in F^n$ that corresponds to β.

By inverting $\mathcal{H}_n$ (which is nonsingular) we can represent the coefficients vector $\vec{C}^f$ in terms of $\vec{f}$ as follows:

$$\vec{C}^f = \mathcal{A}_n \cdot \vec{f}. \tag{2.2}$$

Both $\mathcal{H}_n$ and $\mathcal{A}_n$ can be defined recursively using tensor products:

$$\mathcal{H}_n = \mathcal{H}_1 \otimes \mathcal{H}_{n-1} \quad \text{and} \quad \mathcal{A}_n = \mathcal{A}_1 \otimes \mathcal{A}_{n-1}, \tag{2.3}$$

where

$$\mathcal{H}_1 = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & \omega & \omega^2 & \dots & 1 \\ 1 & \omega^2 & \omega^4 & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{q-2} & \omega^{2(q-2)} & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \end{pmatrix} \tag{2.4}$$

and

$$\mathcal{A}_1 = (-1) \cdot \begin{pmatrix} -1 & 0 & 0 & \dots & 0 \\ 0 & \omega^{-1} & \omega^{-2} & \dots & 1 \\ 0 & \omega^{-2} & \omega^{-4} & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \omega^{-(q-2)} & \omega^{-2(q-2)} & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \end{pmatrix}. \tag{2.5}$$

For our purposes, the important thing that should be noted is that every coefficient C^f_α of the polynomial representation of f is some (easy to compute) linear combination of the values of f on different domain elements $x \in F^n$. In particular, for $\alpha = \langle q-1, \dots, q-1 \rangle$, $C^f_\alpha = (-1)^n \cdot \sum_{x \in F^n} f(x)$.

DEFINITION 1. *Let* $\mathrm{POLY}_{n,d}$ *denote the class of all functions* $f : F^n \to F$ *that are polynomials of total degree at most* d *(where the degree in each variable is at most* $q-1$*). Namely, if we consider the representation of* f *as defined in* (2.1)*, then* $C^f_\alpha = 0$ *for every* $\alpha \in [q-1]^n$ *such that* $\sum_{i=1}^n \alpha_i > d$.

DEFINITION 2. *For* $m \geq 1$ *and any choice of a point* $y \in F^n$ *and* m *linearly independent points* $y_1, \dots, y_m \in F^n$*, let* $S(y_0, y_1, \dots, y_m)$ *denote the affine subspace of dimension* m *that contains all points of the form* $y_0 + \sum_{i=1}^m a_i y_i$*, where* $a_1, \dots, a_m \in F$*. Note that* y_0 *has a different role from all other* y_i*'s.*

Observe that in the special case that y_0 is a linear combination of $y_1, \dots, y_m$, then $S(y_0, y_1, \dots, y_m)$ is a linear subspace of dimension m.

DEFINITION 3. *For a function* $f : F^n \to F$*, a point* $y_0 \in F^n$*, and* m *linearly independent points* $y_1, \dots, y_m \in F^n$*, we denote by* $f_{|(y_0, y_1, \dots, y_m)}$ *the restriction of* f *to the affine subspace* $S(y_0, y_1, \dots, y_m)$*. Namely,* $f_{|(y_0, y_1, \dots, y_m)} : F^m \to F$ *is defined as follows: for every* $v \in F^m$*,* $f_{|(y_0, y_1, \dots, y_m)}(v) = f(y_0 + \sum_{i=1}^m v_i y_i)$*. With slight abuse of notation (and for the sake of succinctness), we shall sometimes use the notation* $f_{|S}$ *instead, where* $S = S(y_0, y_1, \dots, y_m)$ *is the subspace spanned by the points. In case the set of points spanning the subspace* S *is not explicitly stated, then* $f_{|S}$ *is determined by some canonical choice of a basis.*[3]

[3]Our interest lies in the *degree* of these functions (represented as polynomials). Since for any given subspace this degree is invariant with respect to the choice of the basis, the particular choice of the basis is only a matter of convenience.

DEFINITION 4. *For any two functions $f, g : F^n \to F$, let* $\mathrm{dist}(f,g) = \Pr_{y \in F^n}[f(y) \neq g(y)]$ *(where y is selected uniformly).*

3. The characterization. In this section we prove Theorem 2, which was stated in the introduction and provides a characterization of polynomials of total degree at most d over finite fields. Using the notation introduced in section 2, Theorem 2 states that a function $f : F^n \to F$ belongs to $\mathrm{POLY}_{n,d}$ if and only if for every affine subspace S of F^n having dimension $\ell = \lceil \frac{d+1}{q-q/p} \rceil$, we have that $f_{|S} \in \mathrm{POLY}_{\ell,d}$.

In particular, when q is prime, that is, $q = p$, we get that $\ell = \lceil \frac{d+1}{q-1} \rceil$ and we obtain a set of words having weight at most $q^{\lceil \frac{d+1}{q-1} \rceil}$ that span the code $\mathcal{GRM}_q(n(q-1)-(d+1), n)$ (which is dual to $\mathcal{GRM}_q(d,n)$). If we further have that $d+1$ is divisible by $q-1$, then these are exactly the minimum-weight words of $\mathcal{GRM}_q(n(q-1)-(d+1), n)$ (i.e., the characteristic vectors of affine subspaces of dimension ℓ). Otherwise, they have weight that is at most a factor of q larger than the minimum weight words (which correspond to weighted sums of characteristic vectors of affine subspaces of dimension $\ell - 1$). As noted in the introduction, in the case of GRM codes over prime fields, it is known that the minimum-weight words span the code [16]. However, when q is not prime then this is not the case (unless $n = 1$ or d is very large) [16]. Thus, as stated in Corollary 3 in the introduction, we obtain a new result concerning small-weight words that span GRM codes: Since $p \geq 2$, we have that $q - q/p > (q-1)/2$, and so the words that span the code have weight that is at most quadratic in the weight of the minimum weight words.

The following theorem shows that Theorem 2 is tight.

THEOREM 4. *For any given d and $q = p^s$, let ℓ be as defined in Theorem 2. Then there exists a function $f : F^n \to F$ such that for every affine subspace S of F^n having dimension less than ℓ, the function f restricted to S is a degree-d polynomial,* but *f is not a degree-d polynomial.*

Proof. Let $f = x_1^{(p-1)p^{s-1}} \cdot x_2^{(p-1)p^{s-1}} \cdots x_\ell^{(p-1)p^{s-1}}$, so that the degree of f is $\ell \cdot (p-1)p^{s-1}$, which is at least $d+1$. On the other hand, consider any choice of ℓ points $y_0, y_1, \ldots, y_{\ell-1}$ (where $y_1, \ldots, y_{\ell-1}$ are linearly independent). Then

$$\begin{aligned} f_{|(y_0,y_1,\ldots,y_{\ell-1})}(z_1,\ldots,z_{\ell-1}) &= \prod_{j=1}^{\ell} \left(y_{0,j} + \sum_{i=1}^{\ell-1} z_i \cdot y_{i,j} \right)^{(p-1)p^{s-1}} \\ &= \prod_{j=1}^{\ell} \left(y_{0,j}^{p^{s-1}} + \sum_{i=1}^{\ell-1} \left(y_{i,j}^{p^{s-1}} \cdot z_i^{p^{s-1}} \right) \right)^{p-1} \end{aligned} \tag{3.1}$$

(where we have used the fact that $(a+b)^p = a^p + b^p$). If we group together all terms that correspond to the same monomial in the z_i's, then we get a sum of terms of the following form:

$$C \cdot \prod_{i=1}^{\ell-1} z_i^{p^{s-1} \cdot A_i}, \tag{3.2}$$

where $\sum_{i=1}^{\ell-1} A_i \leq \ell(p-1)$ and C is a coefficient that is a function of the $y_{i,j}$'s. Recall that $z_i^q = z_i$ for every $z_i \in F$. We claim that this implies that the total degree of each of the monomials in (3.2) is at most $(\ell-1) \cdot (p-1)p^{s-1}$.

To verify this, consider any choice of $A = A_1, \ldots, A_{\ell-1}$ such that $\sum_{i=1}^{\ell-1} A_i \leq \ell(p-1)$, and let D_A denote the degree of the corresponding monomial. That is,

$$D_A = \sum_{i=1}^{\ell-1} \left((p^{s-1} \cdot A_i) \bmod (q-1) \right). \tag{3.3}$$

Note that each summand on the right-hand side of (3.3) is computed modulo $q-1$, but the sum is then taken over the integers. Let us fix any choice of $A = A_1, \ldots, A_{\ell-1}$ and show that $D_A \leq (\ell-1)(p-1)p^{s-1}$. Assume, without loss of generality, that $A_1 = \max_{1 \leq i \leq \ell-1}\{A_i\}$. If $A_1 \leq p-1$ then clearly $D_A \leq (\ell-1)(p-1)p^{s-1}$. Otherwise, $A_1 = (p-1)+a$ where $0 < a \leq (\ell-1)(p-1)$ and $\sum_{i=2}^{\ell-1} A_i \leq (\ell-1)(p-1)-a$. Now

$$\begin{aligned}
D_A &\leq (p^{s-1} \cdot A_1) \bmod (q-1) \quad + \sum_{i=2}^{\ell-1} p^{s-1} \cdot A_i \\
&\leq (p^{s-1} \cdot (p-1+a)) \bmod (q-1) \quad + p^{s-1} \cdot \Big((\ell-1)(p-1) - a \Big) \\
&= (\ell-1)(p-1)p^{s-1} - ap^{s-1} + ((p-1+a)p^{s-1}) \bmod (q-1) \\
&< (\ell-1)(p-1)p^{s-1} \,.
\end{aligned}$$

We have thus established that the monomial with the highest degree has degree at most $(\ell-1) \cdot (p-1)p^{s-1}$. But now,

$$(\ell-1) \cdot (p-1)p^{s-1} \;=\; \left(\left\lceil \frac{d+1}{(p-1)p^{s-1}} \right\rceil - 1 \right) \cdot (p-1)p^{s-1} \;<\; d+1$$

and since $(\ell-1) \cdot (p-1)p^{s-1}$ is an integer, we are done. □

Proof of Theorem 2. As noted in the introduction, our proof of Theorem 2 generalizes the proof of the special case of $\ell = 1$, which is presented in [19]. If $f \in \mathrm{POLY}_{n,d}$, then clearly $f_{|S} \in \mathrm{POLY}_{\ell,d}$ for every affine subspace S of dimension ℓ. We prove the other direction by induction. Namely, we prove that for every $m \geq \ell$ and every affine subspace S of F^n having dimension m, if for every affine subspace S' of S that has dimension ℓ, we have $f_{|S'} \in \mathrm{POLY}_{\ell,d}$, then $f_{|S} \in \mathrm{POLY}_{m,d}$. Theorem 2 follows by setting $m = n$. The base case $m = \ell$ clearly holds, and so we turn to the induction step.

Assume the induction claim holds for every $m \geq \ell$; we prove it for $m+1$. Namely, we take any $(m+1)$-dimensional affine subspace T of F^n and consider the function $h = f_{|T} : F^{m+1} \to F$. We then use the induction hypothesis by which for every affine subspaces S of F^{m+1} having dimension m (which is isomorphic to an affine subspace of T having dimension m), the restriction of $h = f_{|T}$ to S is a degree-d polynomial.

Let the coefficients of the polynomial representation of h be denoted by $\{C^h_\alpha\}_{\alpha \in [q-1]^{m+1}}$. Our goal is to show that for each $\alpha \in [q-1]^{m+1}$ such that $\sum_{i=1}^{m+1} \alpha_i > d$, we have that $C^h_\alpha = 0$. Let us fix any such α, denote it by α^*, and prove that $C^h_{\alpha^*} = 0$. We break the proof into three cases.

Case 1. *There exists a subset $R \subset \{1, \ldots, m+1\}$, where $|R| = m$, such that $\sum_{i \in R} \alpha^*_i > d$.*

Assume, without loss of generality (since the variables of h, and the corresponding α^*_i's can be reordered), that $R = \{1, \ldots, m\}$, and assume, contrary to the claim, that $C^h_{\alpha^*} \neq 0$. We will show that this implies that there exist $y_0, y_1, \ldots, y_m \in F^{m+1}$ and $\gamma = \gamma_1, \ldots, \gamma_m$, where $\gamma_i \in [q-1]$, and $\sum_{i=1}^m \gamma_i > d$, such that for the affine

subspace of $S = S(y_0, y_2, \ldots, y_m)$ of dimension m we have $C_\gamma^{h_{|S}} \neq 0$, contradicting the induction hypothesis.

Specifically, for $i = 1, \ldots, m+1$ we denote by e_i the ith unit vector of dimension $m+1$. Let $y_i = e_i$ for $i = 1, \ldots, m$, and for each $b \in F$, let $y_0(b) = b \cdot e_{m+1}$. Let $g_b = h_{|(y_0(b), y_1, \ldots, y_m)}$ (so that $g_b : F^m \to F$). Then for each choice of $b \in F$ we have

$$\begin{aligned} g_b(z_1, \ldots, z_m) &= h(z_1, \ldots, z_m, b) \\ &= \sum_{\alpha \in [q-1]^{m+1}} C_\alpha^h \cdot \prod_{i=1}^{m} z_i^{\alpha_i} \cdot b^{\alpha_{m+1}}. \end{aligned}$$

Consider the coefficient of the term $z_1^{\alpha_1^*} \cdot z_2^{\alpha_2^*} \cdots z_m^{\alpha_m^*}$ in g_b, that is, $C^{g_b}_{\alpha_1^*, \ldots, \alpha_m^*}$ (where recall that α^* satisfies $\sum_{i=1}^{m+1} \alpha_i^* > d$, as well as the premise of this case). This coefficient has the following form:

$$C^{g_b}_{\alpha_1^*, \ldots, \alpha_m^*} = \sum_{j=0}^{q-1} C^h_{\alpha_1^*, \ldots, \alpha_m^*, j} \cdot b^j.$$

Namely, it is the evaluation, at b, of the univariate polynomial: $\sum_{j=0}^{q-1} C^h_{\alpha_1^*, \ldots, \alpha_m^*, j} \cdot x^j$. Note that for $j = \alpha_{m+1}^*$, the coefficient of x^j in this polynomial is $C^h_{\alpha^*}$, which is nonzero by our counterassumption. Hence, this polynomial is a nonzero polynomial of degree at most $q-1$ over F. This implies that for at least one value of b, this polynomial attains a nonzero value. But this means that for some choice of b, $C^{g_b}_{\alpha_1^*, \ldots, \alpha_m^*} \neq 0$. Since $\sum_{i=1}^{m} \alpha_i^* > d$, we have reached a contradiction and hence completed the proof for this case.

Case 2. *There exists a pair of indices $i, j \in \{1, \ldots, m+1\}$ such that $\alpha_i^*, \alpha_j^* \neq 0$, and $\alpha_i^* + \alpha_j^* < q$.*

Assume, without loss of generality, that $i = m$ and $j = m+1$. Here too we assume, contrary to the claim, that $C^h_{\alpha^*} \neq 0$, and we reach a contradiction to the induction hypothesis.

Let y_0 be the all-0 vector, let $y_i = e_i$ for $i = 1, \ldots, m-1$, and for each $b \in F$, let $y_m(b) = \langle 0, \ldots, 0, 1, b \rangle$ (recall that $y_0, \ldots, y_m \in F^{m+1}$). Here too we denote $g_b = h_{|(y_0, \ldots, y_m(b))}$. Then for each choice of $b \in F$ we have

$$\begin{aligned} g_b(z_1, \ldots, z_m) &= h(z_1, \ldots, z_m, b \cdot z_m) \\ &= \sum_{\alpha \in [q-1]^{m+1}} C_\alpha^h \cdot \prod_{i=1}^{m-1} z_i^{\alpha_i} \cdot z_m^{\alpha_m} \cdot (z_m \cdot b)^{\alpha_{m+1}} \\ &= \sum_{\alpha \in [q-1]^{m+1}} C_\alpha^h \cdot \prod_{i=1}^{m-1} z_i^{\alpha_i} \cdot z_m^{\alpha_m + \alpha_{m+1}} \cdot b^{\alpha_{m+1}}. \end{aligned}$$

Consider the coefficient of the term $z_1^{\alpha_1^*} \cdots z_{m-1}^{\alpha_{m-1}^*} \cdot z_m^{\alpha_m^* + \alpha_{m+1}^*}$ in g_b (recall that $\alpha_m^* + \alpha_{m+1}^* < q$). This coefficient has the following form:

$$C^{g_b}_{\alpha_1^*, \ldots, \alpha_{m-1}^*, \alpha_m^* + \alpha_{m+1}^*} = \sum_{\substack{j,k \in [q-1]^2 \\ j+k = \alpha_m^* + \alpha_{m+1}^*}} C^h_{\alpha_1^*, \ldots, \alpha_{m-1}^*, j, k} \cdot b^k.$$

That is, it is the evaluation, at b, of the univariate polynomial

$$\sum_{k=0}^{q-1} C^h_{\alpha^*_1,\ldots,\alpha^*_{m-1},\alpha^*_m+\alpha^*_{m+1}-k,k} \cdot x^k.$$

Note that for $k = \alpha^*_{m+1}$, the coefficient of x^k in this polynomial is $C^h_{\alpha^*}$, which is nonzero by our counterassumption. Hence, this polynomial is a nonzero polynomial of degree at most $q-1$ over F, which implies that for at least one value of b it attains a nonzero value. But this means that for some choice of b, $C^{g_b}_{\alpha^*_1,\ldots,\alpha^*_{m-1},\alpha^*_m+\alpha^*_{m+1}} \neq 0$ and the proof of this case follows.

Remark. We observe that if $\ell = \lceil \frac{2(d+1)}{q} \rceil$ then either Case 1 or Case 2 must hold. This implies that Theorem 2 is established for q that is a power of 2 (since in this case $q - q/p = q/2$). It also follows that for any value of q, a variant of Theorem 2 which takes ℓ to be at most a factor of 2 larger than that stated in the theorem, is established as well.

In order to get the tighter result which holds for $\ell = \lceil \frac{d+1}{q-(q/p)} \rceil$ and any q, we need to analyze the third and final case.

Case 3 (neither Case 1 nor Case 2 holds). *For every subset $R \subset \{1,\ldots,m+1\}$, $|R| = m$, we have that $\sum_{i\in R} \alpha^*_i \leq d$, and for every pair of indices $i, j \in \{1,\ldots,m+1\}$ we have that $\alpha^*_i + \alpha^*_j \geq q$.*

Our proof of this case is similar in its general structure to the proofs of Cases 1 and 2, but is somewhat more involved since we take into account a larger set of m-dimensional affine subspaces. In what follows, when we write $t \cdot a$, where $a \in F$ and t is an integer, we mean the sum $\underbrace{a + a + \ldots + a}_{t}$ in the field F. For every choice of $a_1,\ldots,a_m,b$ each in F, let $y_0 = b \cdot e_{m+1}$, and for $i = 1,\ldots,m$, let $y_i = a_i \cdot e_i + e_{m+1}$. Consider the function $g_{a_1,\ldots,a_m,b} = h_{|(y_0(b),y_1(a_1),\ldots,y_m(a_m))} : F^m \to F$. By definition,

$$\begin{aligned}
&g_{a_1,\ldots,a_m,b}(z_1,\ldots,z_m)\\
&= h\Big(a_1 \cdot z_1,\ldots,a_m \cdot z_m, \sum_{i=1}^m z_i + b\Big)\\
&= \sum_{\alpha\in[q-1]^{m+1}} C^h_\alpha \cdot \prod_{i=1}^m (a_i \cdot z_i)^{\alpha_i} \cdot \left(\sum_{i=1}^m z_i + b\right)^{\alpha_{m+1}}\\
&= \sum_{\alpha\in[q-1]^{m+1}} \sum_{\substack{\delta\in[q-1]^m\\ \sum_{i=1}^m \delta_i \leq \alpha_{m+1}}} \binom{\alpha_{m+1}}{\delta_1,\ldots,\delta_m} \cdot C^h_\alpha\\
&\qquad\qquad \cdot \prod_{i=1}^m a_i^{\alpha_i} \cdot b^{\alpha_{m+1}-\sum\delta_i} \cdot \prod_{i=1}^m z_i^{\alpha_i+\delta_i}. \qquad (3.4)
\end{aligned}$$

Roughly speaking, for each $\alpha \in [q-1]^{m+1}$, the exponent α_{m+1} "gets distributed" among the different z_i's ($i = 1,\ldots,m$) and b. Note that if $\alpha_i + \delta_i = q$ then $z_i^{\alpha_i+\delta_i} = z_i$, and, more generally, if $\alpha_i + \delta_i \geq q$ then $z_i^{\alpha_i+\delta_i} = z_i^{(\alpha_i+\delta_i) \bmod (q-1)}$. In what follows we use the shorthand $(j)_q$ to denote $(j \bmod (q-1))$.

For any choice of $\gamma = \gamma_1,\ldots,\gamma_m$, $\gamma_i \in [q-1]$, we consider the coefficient of the term $\prod_{i=1}^m z_i^{\gamma_i}$ in the representation of $g_{a_1,\ldots,a_m,b}(z_1,\ldots,z_m)$ as a polynomial of degree at most $q-1$ in each variable (that is, $C_\gamma^{g_{a_1,\ldots,a_m,b}}$). It follows from (3.4) that this

coefficient is the evaluation of the *multivariate* polynomial

$$H_\gamma(x_1,\ldots,x_{m+1}) = \sum_{\substack{\alpha\in[q-1]^{m+1}\\ \alpha_{m+1}\geq\sum_{i=1}^m(\gamma_i-\alpha_i)_q}} \binom{\alpha_{m+1}}{(\gamma_1-\alpha_1)_q,\ldots,(\gamma_m-\alpha_m)_q} \cdot C_\alpha^h \cdot \prod_{i=1}^m x_i^{\alpha_i} \cdot x_{m+1}^{\alpha_{m+1}-\sum_{i=1}^m(\gamma_i-\alpha_i)_q} \tag{3.5}$$

at the point $x_1=a_1,\ldots,x_m=a_m,x_{m+1}=b$.

As in Cases 1 and 2, we would like to show that under the assumption that $C^h_{\alpha^*}\neq 0$ for α^* such that $\sum_{i=1}^{m+1}\alpha_i^*>d$, we can get the following. There exist $a_1,\ldots,a_m$ and b in F and $\gamma_1,\ldots,\gamma_m\in[q-1]$ such that $\sum_{i=1}^m\gamma_i>d$ and the coefficient of the term $\prod_{i=1}^m z_i^{\gamma_i}$ in the representation of $g_{a_1,\ldots,a_m,b}$ as a polynomial of degree at most $q-1$ (in each variable) is nonzero. Since we want to exploit the existence of $\alpha^*\in[q-1]^{m+1}$ as stated above, we shall consider $\gamma_1,\ldots,\gamma_m$ of the form $\gamma_i=\alpha_i^*+\delta_i$, where $\delta_1,\ldots,\delta_m$ ($\delta_i\in[q-1]$) obey the following conditions:

C1. $\sum_{i=1}^m\delta_i\leq\alpha^*_{m+1}$;
C2. $\alpha_i^*+\delta_i\ (=\gamma_i)\ \leq q-1$ for every i, $1\leq i\leq m$;
C3. $\sum_{i=1}^m(\alpha_i^*+\delta_i)>d$ (that is, $\sum_{i=1}^m\gamma_i>d$);
C4. $\binom{\alpha^*_{m+1}}{\delta_1,\ldots,\delta_m}$ is not divisible by p. (If q is prime, that is, $q=p$, then this condition follows from condition C1, but this is not true in general.)

Suppose we have a setting of the δ_i's that satisfies conditions C1–C4 (where we later show how to obtain such a setting). Let $\gamma=\gamma_1,\ldots,\gamma_m$ be such that $\gamma_i=\alpha_i^*+\delta_i$. By condition C3, $\sum_{i=1}^m\gamma_i>d$, and by condition C2 we have that $\gamma_i\leq q-1$ and $\delta_i=(\gamma_i-\alpha_i^*)_q$. We claim that H_γ (which is defined in (3.5)) includes at least one nonzero coefficient. To verify this first note that by condition C1 (and since $\delta_i=(\gamma_i-\alpha_i^*)_q$), the sum in (3.5) includes $\alpha=\alpha^*$. By our counterassumption, $C^h_{\alpha^*}\neq 0$. Combining this with condition C4 we get that $\binom{\alpha^*_{m+1}}{\delta_1,\ldots,\delta_m}\cdot C^h_{\alpha^*}$ is a nonzero coefficient of the term $\prod_{i=1}^m x_i^{\alpha_i^*}\cdot x_{m+1}^{\alpha^*_{m+1}-\sum_{i=1}^m\delta_i}$ in the polynomial H_γ, so that H_γ is a nonzero polynomial. That is, there exists a choice of $a_1,\ldots,a_m$ and b on which the value of H_γ is nonzero. But by definition of H_γ this means that $C_\gamma^{g_{a_1,\ldots,a_m,b}}\neq 0$ for γ that satisfies $\sum_{i=1}^m\gamma_i>d$, in contradiction to the induction hypothesis.

It remains to show how we find a setting of the δ_i's that satisfies conditions C1–C4.

Subcase 1: *q is prime.* Consider first the case that q is prime. That is, $q=p$. In this case $m\geq\ell=\lceil\frac{d+1}{q-1}\rceil$. Let $\delta_1=q-1-\alpha_1^*$, and recall that, by the premise of this case, for every i,j we have $\alpha_i^*+\alpha_j^*\geq q$, so that necessarily $\delta_1<\alpha^*_{m+1}$. Next let $\delta_2=\min\{q-1-\alpha_2^*,\alpha^*_{m+1}-\delta_1\}$, and in general, $\delta_i=\min\{q-1-\alpha_i^*,\alpha^*_{m+1}-\sum_{j<i}\delta_j\}$. Conditions C1 and C2 directly follow from the definition of the δ_i's, and condition C4 is implied by C1 (since q is prime). It remains to verify that condition C3 holds. If there exists an index i such that $\delta_i=\alpha^*_{m+1}-\sum_{j<i}\delta_j$, then

$$\sum_{i=1}^m(\alpha_i^*+\delta_i)=\sum_{i=1}^{m+1}\alpha_i^*>d.$$

Otherwise, $\delta_i=q-1-\alpha_i^*$ for every $1\leq i\leq m$ and so

$$\sum_{i=1}^m(\alpha_i^*+\delta_i)=m\cdot(q-1)\geq\ell\cdot(q-1)\geq d+1.$$

Subcase 2: *q is not prime.* When q is not a prime number, so that $q = p^s$ for $s > 1$, then the setting of the δ_i's is a bit more involved because condition C4 does not follow from any of the other conditions, and we have to attend to it separately. Since

$$\binom{\alpha^*_{m+1}}{\delta_1, \ldots, \delta_m} = \binom{\alpha^*_{m+1}}{\delta_1} \cdot \binom{\alpha^*_{m+1} - \delta_1}{\delta_2} \cdots \binom{\alpha^*_{m+1} - \sum_{j<i} \delta_j}{\delta_i} \cdots \binom{\alpha^*_{m+1} - \sum_{j<m} \delta_j}{\delta_m}$$

it suffices to show that each term in the above product is not divisible by p. Let us hence rewrite condition C4.

C4. For every $1 \le i \le m$, $\binom{\alpha^*_{m+1} - \sum_{j<i} \delta_j}{\delta_i}$ is not divisible by p.

We shall use the following notation: For each α^*_i, let $k_{i,j}$ for $j \in [p-1]$ be such that $\alpha^*_i = \sum_{j=0}^{s-1} k_{i,j} p^j$. That is, in the representation of α^*_i in basis p, $k_{i,j}$ is the coefficient of p^j. Let us assume without loss of generality that α^*_{m+1} is the smallest α^*_i and that $\alpha^*_1 \le \alpha^*_2 \le \cdots \le \alpha^*_m$ (we may reorder the variables to get that). We know that $\alpha^*_1 < (p-1)p^{s-1}$ (or else $\sum_{i=1}^m \alpha^*_i \ge (p-1)p^{s-1} \cdot \ell \ge d+1$). Since, by the premise of this case, $\alpha^*_i + \alpha^*_1 \ge q = p^s$ for every $i > 1$, necessarily $\alpha^*_i > p^{s-1}$ for $i > 1$. Similarly, $\alpha^*_1 > p^{s-1}$ (since $\alpha^*_1 + \alpha^*_{m+1} \ge q$). Hence for each α^*_i we have that $1 \le k_{i,s-1} \le p-1$.

For $1 \le i \le m$, let $t_i \stackrel{\text{def}}{=} p - 1 - k_{i,s-1}$ (for technical purposes $t_0 \stackrel{\text{def}}{=} 0$). That is, t_i indicates the maximum number that can be added to the coefficient $k_{i,s-1}$ of p^{s-1} without "going over" $p-1$. Recall that $\alpha^*_1, \ldots, \alpha^*_m$ are in nondecreasing order, and so the t_i's are in nonincreasing order. In particular, if $t_i = 0$ then $t_{i'} = 0$ for every $i' > i > 0$. We shall use the following technical claim that was given in [19] and whose proof is provided for the sake of completeness in the appendix.

CLAIM 1. *Let $q = p^s$ for a prime number p and an integer s, and let r and t be integers that satisfy $0 < r \le t \le q-1$. If $r = kp^{s-1}$ for some integer k then $\binom{t}{r}$ is not divisible by p.*

The setting of the δ_i's. We now show how to set the δ_i's so that conditions C1–C4 hold. We start with an informal description of how to "distribute" the weight of α^*_{m+1} between the δ_i's. Consider the representation of α^*_{m+1} in basis p as described above. The highest coefficient in the representation is $k_{m+1,s-1}$. That is,

$$\alpha^*_{m+1} = \sum_{j=0}^{s-1} k_{m+1,j} p^j = k_{m+1,s-1} p^{s-1} + \sum_{j=0}^{s-2} k_{m+1,j} p^j.$$

We view α^*_{m+1} as having $k_{m+1,s-1}$ "units of p^{s-1}" to be distributed, and some "left over." Note that this left-over, $\sum_{j=0}^{s-2} k_{m+1,j} p^j$, is strictly smaller than p^{s-1}.

Starting from δ_1, each δ_i in its turn will be assigned the maximum possible integer multiple of p^{s-1}. Namely, as long as the number of remaining "p^{s-1} units" is more than the number of such units that can still be added to α^*_i (i.e., t_i) and $t_i > 0$, then δ_i is assigned t_i units of p^{s-1}. For these i's we get that $k_{i,s-1}p^{s-1} + \delta_i = (p-1)p^{s-1}$. If we reach an index i such that $t_i = 0$, then we stop distributing what is left of α^*_{m+1}. Alternatively, if we reach an index i for which the number of p^{s-1} units that can be added to it (i.e., t_i) is bigger than the amount of p^{s-1} units that remain to be distributed, then we assign δ_i the rest of the weight of α^*_{m+1} that was not distributed yet.

We now turn to a more formal definition. We initialize i to 1 and do the following:

1. While $i \leq m$ and $0 < t_i \leq k_{m+1,s-1} - \sum_{j<i} t_j$:
   Set $\delta_i = t_i p^{s-1}$, and increase i by 1.
2. If $i \leq m$:
   (a) If $t_i = 0$ then for every i', $i \leq i' \leq m$, set $\delta_i = 0$. (Recall that the t_i's are nonincreasing so that if $t_i = 0$ then for every $i' > i$ we also have $t_{i'} = 0$.)
   (b) Else ($t_i > k_{m+1,s-1} - \sum_{j<i} t_j$), set $\delta_i = \alpha^*_{m+1} - \sum_{j<i} \delta_j$, and for every $i' > i$ set $\delta_{i'} = 0$. Note that

$$\delta_i = \Big(k_{m+1,s-1} - \sum_{j<i} t_j\Big)p^{s-1} + \sum_{j=0}^{s-2} k_{m+1,j}p^j \;<\; t_i p^{s-1}.$$

We next verify that conditions C1–C4 hold for this setting of the δ_i's. Condition C1 ($\sum_{i=1}^m \delta_i \leq \alpha^*_{m+1}$) directly follows from the above process. By the definition of the δ_i's, for every $1 \leq i \leq m$, $\delta_i \leq t_i \cdot p^{s-1}$. Since $t_i = p - 1 - k_{i,s-1}$, we get that

$$\alpha^*_i + \delta_i \;\leq\; (p-1)\cdot p^{s-1} + \sum_{\ell=2}^{s} k_{i,s-\ell}p^{s-\ell} \;<\; q,$$

and so condition C2 holds.

We next verify that condition C3 holds, that is, $\sum_{i=1}^m (\alpha^*_i + \delta_i) > d$. Let i_0 be the index reached at the end of step 1 in the process. Observe that for every $i < i_0$, $\delta_i = t_i \cdot p^{s-1}$. By definition of t_i this implies that $\alpha^*_i + \delta_i \geq (p-1)p^{s-1}$. If $i_0 > m$ then, since $m \geq \ell = \lceil \frac{d+1}{(p-1)p^{s-1}} \rceil$, we get that $\sum_{i=1}^m (\alpha^*_i + \delta_i) \geq d+1$, as required. If $i_0 \leq m$, then there are two cases. In case $t_{i_0} = 0$ (so that for every $i' \geq i_0$ we have $t_{i'} = 0$), then $\alpha^*_i \geq (p-1)p^{s-1}$ for every $i \geq i_0$, so again we get that $\sum_{i=1}^m (\alpha^*_i + \delta_i) \geq d+1$. In case $t_{i_0} > k_{m+1,s-1} - \sum_{j<i} t_j$, then we set $\delta_{i_0} = \alpha^*_{m+1} - \sum_{j<i_0} \delta_j$, so that

$$\sum_{i=1}^{m} (\alpha^*_i + \delta_i) = \sum_{i=1}^{m+1} \alpha^*_i > d.$$

Finally, we verify that condition C4 holds. That is, for every $1 \leq i \leq m$, $\binom{\alpha^*_{m+1} - \sum_{j<i} \delta_j}{\delta_i}$ is not divisible by p. Let i_0 be as defined above in our verification of condition C3. Since for every $i < i_0$ we have that $\delta_i = t_i p^{s-1}$, by Claim 1, $\binom{\alpha^*_{m+1} - \sum_{j<i} \delta_j}{\delta_i}$ is not divisible by p. If $i_0 > m$ then we are done. Otherwise there are two cases. In the first case $\delta_i = 0$ for every $i \geq i_0$, so clearly the condition holds. In the second case, $\delta_{i_0} = \alpha^*_{m+1} - \sum_{j<i_0} \delta_j$ and $\delta_i = 0$ for every $i > i_0$. Thus condition C4 holds in this case too. We have thus completed the proof of Case 3 (Subcase 2), and hence of Theorem 2. □

4. The test. In this section we present and analyze our testing algorithm for degree-d polynomials over fields of cardinality $q = O(d)$.

ALGORITHM 1. Testing Algorithm for Degree-d Polynomials

1. *Let* $\ell = \ell(q,d) = \lceil \frac{d+1}{q - q/p} \rceil$ *and repeat the following* $t = \Theta\big(\ell \cdot q^{\ell+1} + \frac{1}{\epsilon \cdot q^\ell}\big)$ *times:*
   (a) *Uniformly at random select* ℓ *linearly independent points* $y_1, \ldots, y_\ell \in F^n$, *and a point* $y_0 \in F^n$.
   (b) *If* $f_{|(y_0,y_1,\ldots,y_\ell)} \notin \mathrm{POLY}_{\ell,d}$ *then output* reject.
2. *If no step caused rejection then output* accept.

Recall that checking whether $f_{|S} \notin \mathrm{POLY}_{\ell,d}$ (where $S = S(y_0, y_1, \ldots, y_\ell)$) can

be done by querying f on all points in the subspace S and verifying that all linear constraints corresponding to the coefficients $C_\alpha^{f|_S}$ such that $\sum_{i=1}^{\ell} \alpha_i > d$ hold. Hence the total number of queries performed by the algorithm is $O(t \cdot q^\ell) = O(\ell \cdot q^{2\ell+1} + \frac{1}{\epsilon})$ (where the q^ℓ term is due to the number of points in each affine subspace).

As noted in the introduction, when q is sufficiently larger than d so that $\ell = 1$ (the subspaces are lines), then it is not necessary to query f on all points on the line, but rather $d+2$ points suffice. We note that these checks involving $d+2$ points on a line can be interpreted as selecting minimum-weight words from the dual GRM code and checking that they are orthogonal to the word defined by f.

Given Algorithm 1, Theorem 1, as stated in the introduction, follows from the next lemma.

LEMMA 2. *If $f \in \mathrm{POLY}_{n,d}$ then Algorithm 1 accepts with probability 1, and if $\mathrm{dist}(f, \mathrm{POLY}_{n,d}) > \epsilon$ then Algorithm 1 rejects with probability at least $2/3$.*

Lemma 2 shall be proved using the "self-correcting approach," which has been applied in the analysis of many previous low-degree tests. Namely, given the function f we define another function g based on certain "majority votes" of f. We then show that if f passes the test with sufficiently high probability, then g is close to f and g is a polynomial of degree at most d. Bounding the distance between f and g follows easily from the definition of g, and hence the analysis is focused on showing that g is a polynomial of degree at most d. The analysis can be viewed as generalizing both the analysis in [30] (where the subspaces considered by the test are lines) and the analysis in [1] (where the subspaces are larger but the field is $GF(2)$, and the analysis relies on the fact that the field is $GF(2)$).

We start by introducing several notations.

DEFINITION 5. *Let*

$$\eta = \eta(f,d) \stackrel{\mathrm{def}}{=} \Pr_{y_0,y_1,\ldots,y_\ell}\Big[f_{|(y_0,y_1,\ldots,y_\ell)} \notin \mathrm{POLY}_{\ell,d}\Big], \tag{4.1}$$

where the probability is taken over $y_0, y_1, \ldots, y_\ell$ such that $y_1, \ldots, y_\ell$ are linearly independent.

By definition of Algorithm 1, η is the probability that a single step of the algorithm causes f to be rejected. That is, it is the probability that the restriction of f to a random affine subspace of dimension ℓ is *not* a polynomial of degree at most d.

DEFINITION 6. *For each $\alpha \in [q-1]^\ell$, $y \in F^n$, and linearly independent points $y_1, \ldots, y_\ell \in F^n$, let $C_\alpha^f(y_0, y_1, \ldots, y_\ell)$ denote the coefficient C_α of the polynomial representation of $f_{|(y_0,y_1,\ldots,y_\ell)}$. We shall use the notation $B^f(y_0, y_1, \ldots, y_\ell)$ as a shorthand for the coefficient $C^f_{\langle q-1,\ldots,q-1\rangle}(y_0, y_1, \ldots, y_\ell)$. That is, $B^f(y_0, y_1, \ldots, y_\ell)$ denotes the coefficient of the highest-degree monomial $x_1^{q-1} \cdot x_2^{q-1} \cdots x_\ell^{q-1}$ in the polynomial representation of $f_{|(y_0,y_1,\ldots,y_\ell)}$. Recall that this coefficient equals $(-1)^\ell$ times the sum of the values of f taken over all points in the subspace.*

We denote by $V^f(y; y_1, \ldots, y_\ell)$ the value that $f(y)$ "should have" so that $B^f(y, y_1, \ldots, y_\ell) = 0$. That is,

$$V^f(y; y_1, \ldots, y_\ell) = \quad - \sum_{\substack{b_1,\ldots,b_\ell \in F \\ \exists i \text{ s.t. } b_i \neq 0}} f\Big(y + \sum_{i=1}^{\ell} b_i \cdot y_i\Big)\,. \tag{4.2}$$

We refer to $V^f(y; y_1, \ldots, y_\ell)$ as the vote *of $(y_1, \ldots, y_\ell)$ on the value assigned to y.*

For succinctness of the notation, we shall remove f from the last two notations (i.e., $B(\cdot) = B^f(\cdot)$ and $V(\cdot) = V^f(\cdot)$).

Note that for η as in Definition 5,

$$\eta \geq \Pr_{y,y_1,\ldots,y_\ell}[V(y;y_1,\ldots,y_\ell) \neq f(y)],$$

where the probability is taken over $y_1,\ldots,y_\ell$ that are linearly independent. This is true since the test checks that *all* coefficients $C^f_\alpha(y,y_1,\ldots,y_\ell)$ for which $\sum_{i=1}^{\ell}\alpha_i > d$ are 0.

In our analysis, we shall sometimes have to address the case that $y_1,\ldots,y_\ell$ are linearly dependent and we shall use the notation $V(y;y_1,\ldots,y_\ell)$ (as defined in (4.2)), in this case as well. This is despite the fact that it no longer has the same meaning of a "vote" on the value of $f(y)$ (or at least not an "objective vote"). We show the following.

LEMMA 3. *For every $y \in F^n$ and for $y_1,\ldots,y_\ell \in F^n$ that are linearly dependent, $V(y;y_1,\ldots,y_\ell) = f(y)$.*

Proof. Since $y_1,\ldots,y_\ell$ are linearly dependent, we can write y_ℓ as a linear combination of the other points. That is, $y_\ell = \sum_{i=1}^{\ell-1} a_i y_i$, where $a_1,\ldots,a_{\ell-1} \in F$. By definition of $V(\cdot)$,

$$V(y;y_1,\ldots,y_\ell) = -\sum_{b_1,\ldots,b_\ell \in F} f\Big(y + \sum_{i=1}^{\ell} b_i \cdot y_i\Big) + f(y)\,. \tag{4.3}$$

Since $y_\ell = \sum_{i=1}^{\ell-1} a_i y_i$,

$$\sum_{b_1,\ldots,b_\ell \in F} f\Big(y + \sum_{i=1}^{\ell} b_i \cdot y_i\Big) = \sum_{b_1,\ldots,b_\ell \in F} f\Big(y + \sum_{i=1}^{\ell-1} b_i \cdot y_i + b_\ell \sum_{i=1}^{\ell-1} a_i y_i\Big) \tag{4.4}$$

$$= \sum_{b_\ell \in F} \sum_{b_1,\ldots,b_{\ell-1} \in F} f\Big(y + \sum_{i=1}^{\ell} (b_i + b_\ell \cdot a_i) y_i\Big) \tag{4.5}$$

$$= |F| \cdot \sum_{c_1,\ldots,c_{\ell-1} \in F} f\Big(y + \sum_{i=1}^{\ell} c_i y_i\Big) \tag{4.6}$$

$$= 0\,. \tag{4.7}$$

In the above sequence of equalities, (4.6) follows from the fact that for each $b_\ell \in F$, and for every choice of $c_1,\ldots,c_{\ell-1} \in F$, there exists a choice of $b_1,\ldots,b_{\ell-1} \in F$ such that $c_i = b_i + b_\ell \cdot a_i$ (i.e., $b_i = c_i - b_\ell \cdot a$). ∎

We are now ready to define the *self-corrected* version of f, denoted by g.

DEFINITION 7. *Let g be a* plurality *function that is defined as follows. For each $y \in F^n$,*

$$g(y) = \mathrm{argmax}_{a \in F}\Big\{\Pr_{y_1,\ldots,y_\ell \in F^n}[V(y;y_1,\ldots,y_\ell) = a]\Big\}\,. \tag{4.8}$$

The next lemma readily follows from the definition of g.

LEMMA 4. *For any function f and for η and g as defined in* (4.1) *and* (4.8)*, respectively,* $\mathrm{dist}(f,g) \leq 2\eta$.

Proof. First observe that if the test selects points $y_0,y_1,\ldots,y_\ell \in F^n$ such that $f(y_0) \neq V(y_0;y_1,\ldots,y_\ell)$, then this means that $B(y_0;y_1,\ldots,y_\ell) \neq 0$ (where $B(\cdot)$ is as defined in Definition 6), which causes the test to reject. Recall that the test selects $y_1,\ldots,y_\ell$ that are linearly independent. If $y_1,\ldots,y_\ell$ are linearly dependent

then by Lemma 3, $V(y_0; y_1, \ldots, y_\ell) = f(y_0)$. Let $U \subseteq F^n$ consist of all ("bad") points $y \in F^n$ such that $\Pr_{y_1,\ldots,y_\ell \in F^n}[f(y) \neq V(y; y_1, \ldots, y_\ell)] > 1/2$. By definition of η (and Lemma 3) we know that $|U|/q^n < 2\eta$. But for every $x \in F^n \setminus U$, by definition of g we have that $f(x) = g(x)$, and the lemma follows. □

In the next series of lemmas we prove that if η is sufficiently small then g is a polynomial of total degree at most d. In the first, and central lemma, we show that for every y, the value of $g(y)$, which by Definition 7 is the "plurality vote" of $V(y; y_1, \ldots, y_\ell)$, taken over all $y_1, \ldots, y_\ell$, equals the vote of a large fraction of the ℓ-tuples $y_1, \ldots, y_\ell$ (assuming η is sufficiently small).

LEMMA 5. *For any fixed $y \in F^n$, let*

$$\gamma(y) \stackrel{\text{def}}{=} \Pr_{y_1,\ldots,y_\ell}\Big[V(y; y_1, \ldots, y_\ell) = g(y)\Big]. \tag{4.9}$$

Then $\gamma(y) \geq 1 - 2q\ell\eta$.

In order to prove Lemma 5 it will actually be more convenient to work with another measure of "correctness" (or "consistency") of a point y.

LEMMA 6. *For any fixed $y \in F^n$, let*

$$\delta(y) = \Pr_{y_1,\ldots,y_\ell,z_1,\ldots,z_\ell}\Big[V(y; y_1, \ldots, y_\ell) = V(y; z_1, \ldots, z_\ell)\Big] \tag{4.10}$$

and let $\gamma(y)$ be as defined in (4.9). *Then $\gamma(y) \geq \delta(y)$.*

Proof. Let $\beta_a(y) = \Pr_{y_1,\ldots,y_\ell}[V(y; y_1, \ldots, y_\ell) = a]$ (so that in particular, $\sum_{a \in F} \beta_a(y) = 1$). By definition of $\gamma(y)$ we have that $\gamma(y) = \max_a \beta_a(y)$, and by definition of $\delta(y)$ we have that $\delta(y) = \sum_{a \in F}(\beta_a(y))^2$. By convexity, $\max_a \beta_a(y) \geq \sum_{a \in F}(\beta_a(y))^2$, and the claim follows. □

An auxiliary "voting graph." In order to show that $\delta(y)$ is large (and hence $\gamma(y)$ is large), it will be useful to consider the following auxiliary graph. The definition of this graph was inspired by the way Shpilka and Wigderson used Cayley graphs in their work [31] and can also be viewed as formalizing and generalizing part of the analysis in [1]. Each vertex in this graph is labeled by a subset (multiset) of ℓ points, $\{y_1, \ldots, y_\ell\}$, $y_i \in F^n$. The neighbors of $\{y_1, \ldots, y_\ell\}$ are of the form $\{y_2, \ldots, y_{\ell+1}\}$. Each vertex corresponds to ℓ points that can "vote" on the value of $f(y)$ for any given y and hence we refer to it as the *voting graph*.

For a fixed point $y \in F^n$, we say that an edge between $\{y_1, \ldots, y_\ell\}$ and $\{y_2, \ldots, y_{\ell+1}\}$ is *good with respect to y* if $V(y; y_1, \ldots, y_\ell) = V(y; y_2, \ldots, y_{\ell+1})$.

Recall that for linearly independent $y_1, \ldots, y_\ell$, $B(y_0, y_1, \ldots, y_\ell)$ denotes the coefficient $C_{\langle q-1,\ldots,q-1\rangle}(y_0, y_1, \ldots, y_\ell)$, of the restriction of f to the ℓ-dimensional affine subspace determined by $y_0, y_1, \ldots, y_\ell$. That is,

$$B(y_0, y_1, \ldots, y_\ell) = (-1)^\ell \cdot \sum_{b_1,\ldots,b_\ell \in F} f\Big(y + \sum_{i=1}^{\ell} y_i\Big).$$

LEMMA 7. *For any choice of $y, y_1, \ldots, y_{\ell+1} \in F^n$ such that $y_1, \ldots, y_{\ell+1}$ are linearly independent,*

$$\begin{aligned} &V(y; y_1, \ldots, y_\ell) - V(y; y_2, \ldots, y_{\ell+1}) \\ &= (-1)^\ell \left(\sum_{a \in F,\ a \neq 0} B(y + a \cdot y_{\ell+1}, y_1, \ldots, y_\ell) - \sum_{a \in F,\ a \neq 0} B(y + a \cdot y_1, y_2, \ldots, y_{\ell+1}) \right). \end{aligned}$$

Proof. By definition of $V(y;\cdot)$ we have

$$
\begin{aligned}
&V(y;y_1,\ldots,y_\ell)-V(y;y_2,\ldots,y_{\ell+1})\\
&=-\sum_{\substack{b_1,\ldots,b_\ell\in F\\ b_1\neq 0}} f\Big(y+\sum_{i=1}^{\ell} b_i\cdot y_i\Big)+\sum_{\substack{b_2,\ldots,b_{\ell+1}\in F\\ b_{\ell+1}\neq 0}} f\Big(y+\sum_{i=2}^{\ell+1} b_i\cdot y_i\Big)\\
&=-\sum_{\substack{b_1,\ldots,b_\ell\in F\\ b_1\neq 0}} f\Big(y+\sum_{i=1}^{\ell} b_i\cdot y_i\Big)-\sum_{\substack{b_1,\ldots,b_{\ell+1}\in F\\ b_1,b_{\ell+1}\neq 0}} f\Big(y+\sum_{i=1}^{\ell+1} b_i\cdot y_i\Big)\\
&\quad+\sum_{\substack{b_2,\ldots,b_{\ell+1}\in F\\ b_{\ell+1}\neq 0}} f\Big(y+\sum_{i=2}^{\ell+1} b_i\cdot y_i\Big)+\sum_{\substack{b_1,\ldots,b_{\ell+1}\in F\\ b_1,b_{\ell+1}\neq 0}} f\Big(y+\sum_{i=1}^{\ell+1} b_i\cdot y_i\Big)\\
&=-\sum_{\substack{b_1,\ldots,b_{\ell+1}\in F\\ b_1\neq 0}} f\Big(y+b_1\cdot y_1+\sum_{i=2}^{\ell+1} b_i\cdot y_i\Big)+\sum_{\substack{b_1,\ldots,b_{\ell+1}\in F\\ b_{\ell+1}\neq 0}} f\Big(y+b_{\ell+1}\cdot y_{\ell+1}+\sum_{i=1}^{\ell} b_i\cdot y_i\Big)\\
&=(-1)^{\ell}\Bigg(\sum_{a\in F,\ a\neq 0} B(y+a\cdot y_{\ell+1},y_1,\ldots,y_\ell)\\
&\qquad\qquad-\sum_{a\in F,\ a\neq 0} B(y+a\cdot y_1,y_2,\ldots,y_{\ell+1})\Bigg).
\end{aligned}
$$

□

Proof of Lemma 5. Given Lemma 6, it suffices to show that for every $y\in F^n$, $\delta(y)\geq 1-2q\ell\eta$. For any (random) choice of $y_1,\ldots,y_\ell$ and $z_1,\ldots,z_\ell$, and for each $0\leq i\leq \ell$ let $v_i=\{y_1,\ldots,y_i,z_{i+1},\ldots,z_\ell\}$, where we view v_i as a vertex in the voting graph. In particular, $v_\ell=\{y_1,\ldots,y_\ell\}$ and $v_0=\{z_1,\ldots,z_\ell\}$. Since $y_1,\ldots,y_\ell,z_1,\ldots,z_\ell$ are selected uniformly and at random, each v_i is a random variable. Consider the path $v_\ell,\ldots,v_0$ between v_ℓ and v_0. In what follows we shall use the shorthand $V(y;v_i)$ for the vote $V(y;y_1,\ldots,y_i,z_{i+1},\ldots,z_\ell)$. Recall that an edge (v_i,v_{i-1}) is good if $V(y;v_i)=V(y;v_{i-1})$.

We next show that the probability (taken over the choice of $y_1,\ldots,y_\ell,z_1,\ldots,z_\ell$) that an edge (v_i,v_{i-1}) on the path is not good is at most $2q\eta$. By taking a union bound it follows that the probability that all the edges on the path are good is at least $1-2q\ell\eta$. That is, with probability at least $1-2q\ell\eta$, $V(y;y_1,\ldots,y_\ell)=V(y;y_1,\ldots,y_{\ell-1},z_\ell)=\cdots=V(y;z_1,\ldots,z_\ell)$, and the lemma follows.

Consider any edge (v_i,v_{i-1}). We say that $V(y;v_i)$ is an *independent vote* for y if $y_1,\ldots,y_i,z_{i+1},\ldots,z_\ell$ are linearly independent points; otherwise we say that $V(y;v_i)$ is a *dependent vote* for y. If both votes for y are dependent, then by Lemma 3, $V(y;v_i)=f(y)$ and $V(y;v_{i-1})=f(y)$ so that $V(y;v_i)=V(y;v_{i-1})$ and the edge is good. If one of the votes is a dependent vote and the other is an independent vote, then the probability that the edge is not good is the probability that an independent vote for y differs from $f(y)$, which is η.

We next show that if both $V(y;v_i)$ and $V(y;v_{i-1})$ are independent votes for y then the edge (v_i,v_{i-1}) is not good with probability at most $2q\eta$. Indeed, by Lemma 7

such an edge is good if

$$\sum_{a\in F,a\neq 0} B(y+a\cdot y_i, y_1,\ldots,y_{i-1},z_i,\ldots,z_\ell) - \sum_{a\in F,a\neq 0} B(y+a\cdot z_i, y_1,\ldots,y_i,z_{i+1},\ldots,z_\ell) = 0. \tag{4.11}$$

Since $y_1,\ldots,y_i,z_{i+1},\ldots,z_\ell$ and $y_1,\ldots,y_{i-1},z_i,\ldots,z_\ell$. are two sets of linearly independent vectors selected uniformly at random, each of the $B(\cdot)$'s in the above summation is nonzero with probability at most η. Hence, by applying a union bound, the probability that (v_i,v_{i-1}) is not good is at most $2q\eta$ as claimed. □

We next show that if η is sufficiently small then $g\in \mathrm{POLY}_{n,d}$. This is obtained by showing that the restriction of g to every affine subspace of dimension ℓ results in a polynomial of degree at most d. By applying Theorem 2 we conclude that in such a case g is indeed in $\mathrm{POLY}_{n,d}$. In order to show that the restriction of g to every affine subspace of dimension ℓ is a polynomial of degree at most d, we generalize the proof technique applied in [1] for the case of $F=GF(2)$. Roughly speaking, we show that the high degree coefficients in the polynomial representation of the restriction of g to any *fixed* subspace, can be expressed as linear combinations of these coefficients in the restriction of f to *random* subspaces.

LEMMA 8. *If* $\eta < \frac{1}{2(\ell+1)q^{\ell+1}}$, *then* $g\in \mathrm{POLY}_{n,d}$.

Proof. Consider any fixed set of points $y_0,y_1,\ldots,y_\ell\in F^n$ such that $y_1,\ldots,y_\ell$ are linearly independent. We shall show that $g_{|(y_0,y_1,\ldots,y_\ell)}\in \mathrm{POLY}_{\ell,d}$. Lemma 8 follows by applying Theorem 2. We start by describing the high-level idea of the proof. By using a probabilistic argument we shall show that there exists a choice of a subset of elements, denoted $\{z_{i,j}\}$, for which the following conditions hold. First, the value of g on every point w in the subspace $S(y_0,y_1,\ldots,y_\ell)$ equals the vote on w of a set, T_w, of ℓ points that are linear combinations of the $z_{i,j}$'s. Next, for each of these sets of points T_w, the restriction of f to the affine subspace defined by w and T_w is a polynomial of degree at most d. That is, all high degree coefficients in each of these restrictions are 0. We then show that each high degree coefficient in the restriction of g to the subspace $S(y_0,y_1,\ldots,y_\ell)$ is a linear combinations of the high degree coefficients in the abovementioned restrictions of f, and is hence 0. A formal proof follows.

Each point w in the affine subspace $S(y_0,y_1,\ldots,y_\ell)$ is of the form $y_0+\sum_{i=1}^{\ell} b_i y_i$, where $b_i\in F$. Now consider $(\ell+1)\cdot\ell$ elements in F^n, denoted $\{z_{i,j}\}_{i=0,\ldots,\ell}^{j=1,\ldots,\ell}$. Suppose that for every choice of $b_1,\ldots,b_\ell\in F$,

$$g\Big(y_0+\sum_{i=1}^{\ell} b_i\cdot y_i\Big) = V\Big(y_0+\sum_{i=1}^{\ell} b_i\cdot y_i\ ;\ z_{0,1}+\sum_{i=1}^{\ell} b_i\cdot z_{i,1},\ldots,z_{0,\ell}+\sum_{i=1}^{\ell} b_i\cdot z_{i,\ell}\Big). \tag{4.12}$$

If we select the elements $\{z_{i,j}\}$ uniformly and at random, then by Lemma 5 and the union bound, this event occurs with probability at least $1-2q\ell\eta\cdot q^{\ell}$. We assume from this point on that (4.12) holds for every choice of $b_1,\ldots,b_\ell\in F$.

In order to show that $g_{|(y_0,y_1,\ldots,y_\ell)}\in \mathrm{POLY}_{\ell,d}$ we need to show that for every $\alpha\in[q-1]^\ell$ such that $\sum_{i=1}^{\ell}\alpha_i > d$, we have that $C_\alpha^g(y_0,y_1,\ldots,y_\ell)=0$. Let us fix any such α, and let R_α denote the row vector $\mathcal{A}_\ell(\alpha,\cdot)$ (where the matrix $\mathcal{A}_\ell$ is as

defined by (2.3) and (2.5)). Recall that the coordinates of R_α are indexed by strings $\beta \in [q-1]^\ell$ (where we denote the corresponding coordinate by $R_\alpha^\beta \in F$). In what follows we use the notation $ex(\beta_i) = \omega^{\beta_i}$ if $\beta_i \neq 0$ and $ex(\beta_i) = 0$ if $\beta_i = 0$. Consider the structure of $\mathcal{A}_\ell$ presented in section 2. We need to show that

$$\sum_{\beta \in [q-1]^\ell} R_\alpha^\beta \cdot g\Big(y_0 + \sum_{i=1}^{\ell} ex(\beta_i) \cdot y_i\Big) = 0. \tag{4.13}$$

For any fixed $\beta \in [q-1]^\ell$, by our assumption that (4.12) holds, and by definition of $V(\cdot)$, we have

$$\begin{aligned}
&g\Big(y_0 + \sum_{i=1}^{\ell} ex(\beta_i) \cdot y_i\Big) \\
&= \; - \sum_{\substack{\gamma \in [q-1]^\ell \\ \gamma \neq (0,0,\ldots,0)}} f\left(y_0 + \sum_{i=1}^{\ell} ex(\beta_i) \cdot y_i \;\; + \sum_{j=1}^{\ell} ex(\gamma_j) \cdot \Big(z_{0,j} + \sum_{i=1}^{\ell} ex(\beta_i) \cdot z_{i,j}\Big)\right) \\
&= \; - \sum_{\substack{\gamma \in [q-1]^\ell \\ \gamma \neq (0,0,\ldots,0)}} f\left(y_0 + \sum_{j=1}^{\ell} ex(\gamma_j) \cdot z_{0,j} \;\; + \sum_{i=1}^{\ell} ex(\beta_i) \cdot \Big(y_i + \sum_{j=1}^{\ell} ex(\gamma_j) \cdot z_{i,j}\Big)\right).
\end{aligned}$$

This implies (by switching the order of summations) that $\sum_{\beta \in [q-1]^\ell} R_\alpha^\beta \cdot g(y_0 + \sum_{i=1}^{\ell} ex(\beta_i) \cdot y_i)$, which we would like to show is 0, is the sum over all nonzero $\gamma \in [q-1]^\ell$ and all $\beta \in [q-1]^\ell$ of R_α^β times the evaluation of f at

$$y_0 + \sum_{j=1}^{\ell} ex(\gamma_j) \cdot z_{0,j} + \sum_{i=1}^{\ell} ex(\beta_i) \cdot \Big(y_i + \sum_{j=1}^{\ell} ex(\gamma_j) \cdot z_{i,j}\Big).$$

But for any given choice of $\gamma = \gamma_1, \ldots, \gamma_\ell$,

$$\begin{aligned}
&\sum_{\beta \in [q-1]^\ell} R_\alpha^\beta \cdot f\left(y_0 + \sum_{j=1}^{\ell} ex(\gamma_j) \cdot z_{0,j} \;\; + \sum_{i=1}^{\ell} ex(\beta_i) \cdot \Big(y_i + \sum_{j=1}^{\ell} ex(\gamma_j) \cdot z_{i,j}\Big)\right) \\
&= C_\alpha^f\Big(y_0 + \sum_{j=1}^{\ell} ex(\gamma_j) \cdot z_{0,j},\; y_1 + \sum_{j=1}^{\ell} ex(\gamma_j) \cdot z_{1,j},\; \ldots,\; y_\ell + \sum_{j=1}^{\ell} ex(\gamma_j) \cdot z_{\ell,j}\Big).
\end{aligned}$$

Consider the event that for every choice of $\gamma_1, \ldots, \gamma_\ell$ that are not all 0,

$$y_1 + \sum_{j=1}^{\ell} ex(\gamma_j) \cdot z_{1,j},\; \ldots,\; y_\ell + \sum_{j=1}^{\ell} ex(\gamma_j) \cdot z_{\ell,j}$$

are linearly independent. This event occurs with probability at least $1 - q^\ell \cdot q^{\ell-n}$. In what follows we shall assume that this is indeed the case. Since $\gamma_1, \ldots, \gamma_\ell$ are not all 0, then we know that for each setting of $\gamma_1, \ldots, \gamma_\ell$, with probability at most η over the choice of the $z_{i,j}$'s, for every α such that $\sum_{i=1}^{\ell} \alpha_i > d$,

$$C_\alpha^f\Big(y_0 + \sum_{j=1}^{\ell} ex(\gamma_j) \cdot z_{0,j},\; y_1 + \sum_{j=1}^{\ell} ex(\gamma_j) \cdot z_{1,j},\; \ldots,\; y_\ell + \sum_{j=1}^{\ell} ex(\gamma_j) \cdot z_{\ell,j}\Big) \neq 0.$$

By taking a union bound over all $\gamma_1, \ldots, \gamma_\ell$, adding the probability that we have at least one linearly dependent combination, and adding the probability that (4.12) does not hold for some $\beta \in [q-1]^\ell$, we get that with probability at least

$$1 - q^\ell \eta - 2q\ell\eta q^\ell - q^{2\ell - n} \tag{4.14}$$

there exist $z_{i,j}$'s that satisfy all required constraints. Hence (4.13) holds as desired. Note that our algorithm performs $\Theta(\ell q^{2\ell+1} + 1/\epsilon)$ queries, so that we may assume that $\ell q^{2\ell+1} < q^n$ (or else the algorithm would simply query all q^n points). Therefore, the expression in (4.14) is greater than 0 and the lemma follows. □

By combining Lemmas 4 and 8 we obtain that if f is $\Omega(\frac{1}{\ell q^\ell})$-far from $\mathrm{POLY}_{n,d}$, then $\eta = \Omega(\frac{1}{\ell q^\ell})$, and so the algorithm rejects f with sufficiently high constant probability.

The next lemma, which will help us deal with the case in which η is small, is a variant of a very similar lemma that was proved in [1]. For the sake of completeness we provide its proof in the appendix.

LEMMA 9. *Let* $\zeta \stackrel{\text{def}}{=} \frac{1-q^\ell \cdot \mathrm{dist}(f,g)}{1+q^\ell \cdot \mathrm{dist}(f,g)} \cdot q^\ell \cdot \mathrm{dist}(f,g)$. *If we uniformly and independently select* $y_0, y_1, \ldots, y_\ell \in F^n$ *where* $y_1, \ldots, y_\ell$ *are linearly independent, then the probability that for* exactly *one choice of* $b_1, \ldots, b_\ell \in F$, *we have that* $f(y_0 + \sum_{i=1}^{\ell} b_i \cdot y_i) \neq g(y_0 + \sum_{i=1}^{\ell} b_i \cdot y_i)$ *is at least* ζ.

We are now ready to wrap up the proof of Lemma 2 (and hence Theorem 1).

Proof of Lemma 2. If $f \in \mathrm{POLY}_{n,d}$, then, as stated in Theorem 2, $f_{|S} \in \mathrm{POLY}_{\ell,d}$ for every affine subspace S of dimension ℓ, and so the tester accepts with probability 1. We next show that if f is ϵ-far from $\mathrm{POLY}_{n,d}$, then the tester rejects with probability at least $\frac{2}{3}$.

Suppose that $\mathrm{dist}(f, \mathrm{POLY}_{n,d}) > \epsilon$. We shall show that $\eta \geq \min\{\frac{1}{2}q^\ell\epsilon, \frac{1}{2(\ell+1)q^{\ell+1}}\}$. Since η is the probability that a single iteration of the algorithm causes f to be rejected, and the algorithm performs $\Theta(1/\eta)$ iterations, the lemma follows. If $\eta \geq \frac{1}{2(\ell+1)q^{\ell+1}}$ then we are done. Hence, assume that $\eta < \frac{1}{2(\ell+1)q^{\ell+1}}$. We shall show that in such a case $\eta \geq \frac{1}{2} \cdot q^\ell \cdot \mathrm{dist}(f,g) > \frac{1}{2} \cdot q^\ell \cdot \epsilon$. To verify this, first note that by Lemma 8 we have that $g \in \mathrm{POLY}_{n,d}$ (since $\eta < \frac{1}{2(\ell+1)q^{\ell+1}}$). Next observe that if f and g disagree on exactly one point in a subspace S of dimension ℓ, then $f_{|S} \notin \mathrm{POLY}_{\ell,d}$. It follows from Lemma 9 and the definition of η that $\eta \geq \zeta$ (where ζ is as defined in Lemma 9). In particular, since by Lemma 4 $\mathrm{dist}(f,g) \leq 2\eta \leq \frac{1}{(\ell+1)q^{\ell+1}}$ where $\ell \geq 1$ and $q \geq 2$, we get that

$$\begin{aligned}
\eta &\geq \frac{1 - q^\ell \cdot \mathrm{dist}(f,g)}{1 + q^\ell \cdot \mathrm{dist}(f,g)} \cdot q^\ell \cdot \mathrm{dist}(f,g) \\
&\geq \frac{1 - \frac{1}{q(\ell+1)}}{1 + \frac{1}{q(\ell+1)}} \cdot q^\ell \cdot \mathrm{dist}(f,g) \\
&\geq \frac{1 - 1/4}{1 + 1/4} \cdot q^\ell \cdot \mathrm{dist}(f,g) \\
&> \frac{1}{2} \cdot q^\ell \cdot \mathrm{dist}(f,g)
\end{aligned}$$

as claimed. □

5. A lower bound.

THEOREM 5. *Every algorithm for testing* $\mathrm{POLY}_{n,d}$ *with distance parameter* ϵ *must perform* $\Omega\left(\max\{\frac{1}{\epsilon}, q^{\ell-1}\}\right)$ *queries when* q *is prime, and* $\Omega\left(\max\{\frac{1}{\epsilon}, q^{\lceil \ell/2 \rceil - 1}\}\right)$ *queries otherwise.*

In order to establish Theorem 5, we consider the relation between polynomials and codes. Specifically, recall that the family $\mathrm{POLY}_{n,d}$ over a field $F = \mathrm{GF}(q) = \mathrm{GF}(p^s)$ corresponds to the GRM code $\mathcal{GRM}_q(d,n)$. Namely, each codeword (having length q^n) is determined by the evaluation of a polynomial in $\mathrm{POLY}_{n,d}$ on all points in the domain F^n. The minimum distance, $\Delta(\mathcal{GRM}_q(d,n))$, of the code is the following (cf. [16]): If $d = r(q-1)+t$, where $0 \le t < q-1$, and r is an integer, then $\Delta(\mathcal{GRM}_q(d,n)) = (q-t)q^{n-r-1}$. The dual code of $\Delta(\mathcal{GRM}(d,n))$ is the GRM code $\mathcal{GRM}_q(n(q-1)-(d+1),n)$, so that it has distance $\Omega(q^{\lceil \frac{d+1}{q-1} \rceil - 1})$. Let us denote the distance of the dual code by $\overline{\Delta}(\mathcal{GRM}_q(d,n)$, and let $\ell = \lceil \frac{d+1}{q-q/p} \rceil$ be as in our previous notation. Hence, if q is prime then $\overline{\Delta}(\mathcal{GRM}_q(d,n)) = \Omega(q^{\ell-1})$, and for nonprime q we can say that $\overline{\Delta}(\mathcal{GRM}_q(d,n)) = \Omega(q^{\lceil \ell/2 \rceil - 1})$.

Theorem 5 follows by applying the theorem below, which is a straightforward generalization of a similar theorem proved in [1] for binary codes. For the sake of completeness we include the proof in the appendix.

THEOREM 6. *Let* $\mathcal{F}$ *be any family of functions* $f : F^n \to F$ *that corresponds to a linear code* $\mathcal{C}$*. Let* $\Delta(\mathcal{C})$ *denote the minimum distance of the code* $\mathcal{C}$ *and let* $\overline{\Delta}(\mathcal{C})$ *denote the minimum distance of the dual code of* $\mathcal{C}$*.*

Every testing algorithm for the family $\mathcal{F}$ *must perform* $\Omega(\overline{\Delta}(\mathcal{C}))$ *queries, and if the distance parameter* ϵ *is at most* $\Delta(\mathcal{C})/(2q^n)$*, then* $\Omega(1/\epsilon)$ *is also a lower bound for the necessary number of queries.*

6. The paper of Jutla et al. [25]. As noted in the introduction, independently from our work, Jutla et al. [25] give a testing algorithm for low-degree polynomials over prime fields. They too provide a characterization of low-degree polynomials (over prime fields) and define their test based on this characterization. As we discuss below, our characterization (in the case of prime fields) is related to the one in [25]. However, our approach, and hence the proofs for the characterizations, are different, and in particular our characterization holds for all fields. Our testing algorithms and their analysis have a similar structure (which follows that of previous low-degree tests), but there are several technical and expositional differences (partly due to our unifying view of low-degree testing). We next discuss how the characterization in [25] relates to ours.

Recall that we show that for $F = \mathrm{GF}(q)$ (where $q = p^s$ and p is prime), a function $f : F^n \to F$ is a polynomial of degree at most d if and only if its restriction to every affine subspace of dimension $\ell = \lceil \frac{d+1}{q-q/p} \rceil$ is a polynomial of degree at most d. In other words, if we consider the unique representation of each such restriction of f as a polynomial over ℓ variables, then all coefficients that correspond to monomials having degree greater than d must be 0. Each such coefficient can be shown to equal a certain linear combination of the values of f in the subspace (see section 2).

The characterization of Jutla et al. for prime fields ($q = p$) is that one particular linear constraint over each subspace must hold. They do not approach the problem as we do (that is, by characterizing low-degree polynomials as functions whose restrictions to lower-dimensional spaces are low-degree polynomials). However, translating their result using our terminology, it can be shown that the linear constraint they consider corresponds to the coefficient of exactly one monomial of degree $d+1$. This

monomial has the following form: $x_1^{q-1} \cdot x_2^{q-1} \cdots x_{\ell-1}^{q-1} \cdot x_\ell^t$, where $1 \le t \le q-1$ and $d+1 = (\ell-1)(q-1)+t$.

In retrospect we observe that our analysis can be slightly extended so as to show that in the case that q is prime then the characterization can be restricted to a single polynomial coefficient. When q is not prime then it is not clear whether the characterization can be restricted in a similar manner.

Appendix. Proofs of Claim 1, Lemma 9, and Theorem 6.

CLAIM 1. *Let $q = p^s$ for a prime number p and an integer s, and let r and t be integers that satisfy $0 < r \le t \le q-1$. If $r = kp^{s-1}$ for some integer k then $\binom{t}{r}$ is not divisible by p.*

Proof. For any positive integer j, the largest power of p that divides $j!$ is

$$\lfloor j/p \rfloor + \lfloor j/p^2 \rfloor + \lfloor j/p^3 \rfloor + \cdots \ .$$

But for $r = kp^{s-1}$, the identity $\lfloor t/p^i \rfloor = \lfloor r/p^i \rfloor + \lfloor (t-r)/p^i \rfloor$ holds. Thus the largest power of p that divides $t!$ is

$$\sum_{i=1}^{\infty} \lfloor t/p^i \rfloor = \sum_{i=1}^{\infty} \left(\lfloor r/p^i \rfloor + \lfloor (t-r)/p^i \rfloor \right) \ .$$

Therefore $t!$ and $r!(t-r)!$ are divisible by exactly the same power of p. ∎

LEMMA 9. *Let $\zeta \stackrel{\text{def}}{=} \frac{1-q^\ell \mathrm{dist}(f,g)}{1+q^\ell \mathrm{dist}(f,g)} \cdot q^\ell \mathrm{dist}(f,g)$. If we uniformly and independently select $y_0, y_1, \ldots, y_\ell \in F^n$ where $y_1, \ldots, y_\ell$ are linearly independent, then the probability that for* exactly *one choice of $b_1, \ldots, b_\ell \in F$, we have that $f(y_0 + \sum_{i=1}^{\ell} b_i \cdot y_i) \neq g(y_0 + \sum_{i=1}^{\ell} b_i \cdot y_i)$ is at least ζ.*

Proof. For each $\beta = \beta_1, \ldots, \beta_\ell, \beta_i \in [q-1]$ let X_β be the indicator random variable whose value is 1 if and only if $f(y_0 + \sum_{i=1}^{\ell} ex(\beta_i) y_i) \neq g(y_0 + \sum_{i=1}^{\ell} ex(\beta_i) y_i)$. Thus $\Pr[X_\beta = 1] = \mathrm{dist}(f,g)$ for every β. It is not difficult to verify that the random variables X_β are pairwise independent. This is true since for any two distinct β^1, β^2, the points $(y_0 + \sum_{i=1}^{\ell} ex(\beta_i^1) y_i)$ and $(y_0 + \sum_{i=1}^{\ell} ex(\beta_i^2) y_i)$ attain each pair of distinct values in F^n with equal probability. It follows that the random variable $X = \sum_\beta X_\beta$, which counts the number of points $v = (y_0 + \sum_{i=1}^{\ell} ex(\beta_i) y_i)$ in which $f(v) \neq g(v)$, has expectation $\mathrm{E}[X] = q^\ell \cdot \mathrm{dist}(f,g)$ and variance $\mathrm{Var}[X] = q^\ell \cdot \mathrm{dist}(f,g) \cdot (1 - \mathrm{dist}(f,g)) \le \mathrm{E}[X]$. Our objective is to lower bound the probability that $X = 1$. We need the well-known fact that for a random variable X that attains nonnegative, integer values,

$$\Pr[X > 0] \ge \frac{\left(\mathrm{E}[X]\right)^2}{\mathrm{E}[X^2]} \ .$$

Indeed, if X attains the value i with probability ν_i for $i > 0$, then, by Cauchy–Schwarz,

$$\begin{aligned} \left(\mathrm{E}[X]\right)^2 = \left(\sum_{i>0} i\nu_i\right)^2 &= \left(\sum_{i>0} i\sqrt{\nu_i}\sqrt{\nu_i}\right)^2 \\ &\le \left(\sum_{i>0} i^2\nu_i\right) \cdot \left(\sum_{i>0} \nu_i\right) \\ &= \mathrm{E}[X^2] \cdot \Pr[X > 0] \ . \end{aligned}$$

In our case, this implies

$$\Pr[X > 0] \geq \frac{\Big(\mathrm{E}[X]\Big)^2}{\mathrm{E}[X^2]} \geq \frac{\Big(\mathrm{E}[X]\Big)^2}{\mathrm{E}[X] + \Big(\mathrm{E}[X]\Big)^2} = \frac{\mathrm{E}[X]}{1 + \mathrm{E}[X]} \ .$$

Therefore

$$\mathrm{E}[X] \geq \Pr[X = 1] + \Big(\frac{\mathrm{E}[X]}{1 + \mathrm{E}[X]} - \Pr[X = 1]\Big) \cdot 2 = \frac{2\mathrm{E}[X]}{1 + \mathrm{E}[X]} - \Pr[X = 1] \ ,$$

implying that

$$\Pr[X = 1] \geq \frac{\mathrm{E}[X] - (\mathrm{E}[X])^2}{1 + \mathrm{E}[X]} \ .$$

Substituting the value of $\mathrm{E}[X]$, the desired result follows. □

THEOREM 6. *Let $\mathcal{F}$ be any family of functions $f : F^n \to F$ that corresponds to a linear code $\mathcal{C}$. Let $\Delta(\mathcal{C})$ denote the minimum distance of the code $\mathcal{C}$ and let $\overline{\Delta}(\mathcal{C})$ denote the minimum distance of the dual code of $\mathcal{C}$.*

Every testing algorithm for the family $\mathcal{F}$ must perform $\Omega(\overline{\Delta}(\mathcal{C}))$ queries, and if the distance parameter ϵ is smaller than $\Delta(\mathcal{C})/(2q^n)$, then $\Omega(1/\epsilon)$ is also a lower bound for the necessary number of queries.

Proof. We start by showing that $\Omega(\overline{\Delta}(\mathcal{C}))$ queries are necessary. A well-known fact from coding theory (see [28, Chap. 1, Thm. 10]) states the following: for every linear code $\mathcal{C}$ whose dual code has distance $\overline{\Delta}(\mathcal{C})$, if we examine a subword having length Δ', where $\Delta' < \overline{\Delta}(\mathcal{C})$, of a uniformly selected codeword in $\mathcal{C}$, then the resulting subword is uniformly distributed in $F^{\Delta'}$. Hence it is not possible to distinguish between a random codeword in $\mathcal{C}$ and a random word in F^n (which with high probability is far from any codeword) using less than $\overline{\Delta}(\mathcal{C})$ queries.

We now turn to the case $\epsilon < \Delta(\mathcal{C})/2q^n$. To prove the lower bound here, we apply, as usual, the Yao principle by defining two distributions, one of positive instances, and the other of negative ones, and then showing that in order to distinguish between those distributions any algorithm must perform $\Omega(1/\epsilon)$ queries. The positive distribution has all its mass at the zero vector $\bar{0} = (0, \ldots, 0)$. To define the negative distribution, partition the set of all coordinates randomly into $t = 1/\epsilon$ nearly equal parts $I_1, \ldots, I_t$ and give weight $1/t$ to each of the characteristic vectors w_i of I_i, $i = 1, \ldots, t$. (Observe that indeed $\bar{0} \in \mathcal{C}$ due to linearity, and $\mathrm{dist}(w_i, \mathcal{C}) = \epsilon$ due to the assumption on the minimum distance of $\mathcal{C}$.) Finally, a random instance is generated by first choosing one of the distributions with probability $1/2$ and then generating a vector according to the chosen distribution.

Consider the two distributions that were defined. Let A be a deterministic testing algorithm with query complexity s (where s is a function of ϵ). We need to show that if A gives an incorrect answer with probability at most $1/3$, it must be that $s > 1/(3\epsilon)$. If A is incorrect on $\bar{0}$ (that is, it does not accept it), then it is already incorrect with probability at least $1/2$. Otherwise A should accept the input if all the s queried bits are 0. Therefore it accepts as well at least $t - s$ (where $t = 1/\epsilon$ is as defined above) of the inputs w_i. This shows that A gives an incorrect answer with probability at least $(t - s)/2t$. For this to be smaller than $1/3$ it must be the case that $s > 1/(3\epsilon)$. □

Acknowledgments. We are greatly indebted to Madhu Sudan who suggested the unifying view for testing polynomials over finite fields that we apply in this work. We would also like to thank Simon Litsyn, Alex Samorodnitsky, and Adam Smith for helpful discussions. Finally, we would like to thank two anonymous reviewers for helpful comments.

REFERENCES

[1] N. Alon, M. Krivelevich, T. Kaufman, S. Litsyn, and D. Ron, *Testing Reed-Muller codes*, IEEE Trans. Inform. Theory, 51 (2005), pp. 4032–4038.

[2] N. Alon, M. Krivelevich, T. Kaufman, and D. Ron, *Testing triangle-freeness in general graphs*, in Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2006.

[3] S. Arora, *Probabilistic Checking of Proofs and the Hardness of Approximation Problems*, Ph.D. thesis, UC Berkeley, Berkeley, CA, 1994.

[4] S. Arora and M. Sudan, *Improved low-degree testing and its applications*, in Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, ACM, New York, 1997, pp. 485–495.

[5] E. F. Assmus, Jr., and J. D. Key, *Designs and Their Codes*, Cambridge Tracts in Math. 103, Cambridge University Press, Cambridge, UK, 1992.

[6] L. Babai, L. Fortnow, L. Levin, and M. Szegedy, *Checking computations in polylogarithmic time*, in Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing, ACM, New York, 1991, pp. 21–31.

[7] L. Babai, L. Fortnow, and C. Lund, *Non-deterministic exponential time has two-prover interactive protocols*, Comput. Complexity, 1 (1991), pp. 3–40.

[8] L. Babai, A. Shpilka, and D. Stefankovic, *Locally testable cyclic codes*, IEEE Trans. Inform. Theory, 51 (2005), pp. 2849–2858.

[9] M. Bellare, D. Coppersmith, J. Håstad, M. Kiwi, and M. Sudan, *Linearity testing over characteristic two*, IEEE Trans. Inform. Theory, 42 (1996), pp. 1781–1795.

[10] M. Bellare, S. Goldwasser, C. Lund, and A. Russell, *Efficient probabilistically checkable proofs and applications to approximation*, in Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing, ACM, New York, 1993, pp. 294–304.

[11] M. Bellare and M. Sudan, *Improved non-approximability results*, in Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing, ACM, New York, 1994, pp. 184–193.

[12] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan, *Robust PCPs of proximity, shorter PCPs and applications to coding*, in Proceedings of the Thirty-Sixth Annual ACM Symposium on the Theory of Computing, ACM, New York, 2004, pp. 1–10.

[13] E. Ben-Sasson, P. Harsha, and S. Raskhodnikova, *Some 3CNF properties are hard to test*, SIAM J. Comput., 35 (2005), pp. 1–21.

[14] E. Ben-Sasson, M. Sudan, S. Vadhan, and A. Wigderson, *Derandomizing low degree tests via epsilon-biased spaces*, in Proceedings of the Thirty-Fifth Annual ACM Symposium on the Theory of Computing, ACM, New York, 2003, pp. 612–621.

[15] M. Blum, M. Luby, and R. Rubinfeld, *Self-testing/correcting with applications to numerical problems*, J. Comput. System Sci., 47 (1993), pp. 549–595.

[16] P. Ding and J. D. Key, *Minimum-weight codewords as generators of generalized Reed-Muller codes*, IEEE Trans. Inform. Theory, 46 (2000), pp. 2152–2157.

[17] I. Dinur, *The PCP theorem via gap amplification*, in Proceedings of the Thirty-Eighth Annual ACM Symposium on the Theory of Computing, ACM, New York, 2006, pp. 241–250.

[18] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, *Approximating clique is almost NP-complete*, JACM, (1996), pp. 268–292.

[19] K. Friedl and M. Sudan, *Some improvements to total degree tests*, in Proceedings of the 3rd Annual Israel Symposium on Theory of Computing and Systems, Tel Aviv, Israel, 1995, pp. 190–198; corrected version available online at http://theory.lcs.mit.edu/~madhu/papers/friedl.ps.

[20] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson, *Self-testing/correcting for polynomials and for approximate functions*, in Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing, ACM, New York, 1991, pp. 32–42.

[21] O. Goldreich, S. Goldwasser, and D. Ron, *Property testing and its connection to learning and approximation*, JACM, 45 (1998), pp. 653–750.
[22] O. Goldreich and D. Ron, *A sublinear bipartite tester for bounded degree graphs*, Combinatorica, 19 (1999), pp. 335–373.
[23] O. Goldreich and D. Ron, *Property testing in bounded degree graphs*, Algorithmica, 32 (2002), pp. 302–343.
[24] O. Goldreich and M. Sudan, *Locally testable codes and PCPs of almost-linear length*, in Proceedings of the Forty-Third Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2002, pp. 13–22.
[25] C. S. Jutla, A. C. Patthak, A. Rudra, and D. Zuckerman, *Testing low-degree polynomials over prime fields*, in Proceedings of the Forty-Fifth Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2004, pp. 423–432.
[26] T. Kaufman, M. Krivelevich, and D. Ron, *Tight bounds for testing bipartiteness in general graphs*, SIAM J. Comput., 33 (2004), pp. 1441–1483.
[27] T. Kaufman and S. Litsyn, *Almost orthogonal linear codes are locally testable*, in Proceedings of the Forty-Sixth Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2005, pp. 317–326.
[28] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*, North–Holland, Amsterdam, 1977.
[29] A. Polishchuk and D. Spielman, *Nearly-linear size holographic proofs*, in Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, ACM, New York, 1994, pp. 194–203.
[30] R. Rubinfeld and M. Sudan, *Robust characterization of polynomials with applications to program testing*, SIAM J. Comput., 25 (1996), pp. 252–271.
[31] A. Shpilka and A. Wigderson, *Derandomizing homomorphism testing in general groups*, in Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing, ACM, New York, 2004, pp. 427–435.
[32] M. Sudan, *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems*, Lecture Notes in Comput. Sci. 1001, Springer-Verlag, Berlin, 1996.

SIAM J. COMPUT.
Vol. 36, No. 3, pp. 803–814

EXPONENTIAL DETERMINIZATION FOR ω-AUTOMATA WITH A STRONG FAIRNESS ACCEPTANCE CONDITION*

SHMUEL SAFRA†

Abstract. In [S. Safra, *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science*, 1988, pp. 319–327] an exponential determinization procedure for Büchi automata was shown, yielding tight bounds for decision procedures of some logics (see [A. E. Emerson and C. Jutla, *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science*, 1988, pp. 328–337; Safra (1988); S. Safra and M. Y. Vardi, *Proceedings of the 21st ACM Symposium on Theory of Computing*, 1989, pp. 127–137; and D. Kozen and J. Tiuryn, *Logics of program*, in Handbook of Theoretical Computer Science, Elsevier, Amsterdam, 1990, pp. 789–840]). In Safra and Vardi (1989) the complexity of determinization and complementation of ω-automata was further investigated, leaving as an open question the complexity of the determinization of a single class of ω-automata. For this class of ω-automata with strong fairness as an acceptance condition (*Streett automata*), Safra and Vardi (1989) managed to show an exponential complementation procedure; however, the blow-up of translating these automata—to any of the classes known to admit exponential determinization—is inherently exponential. This might suggest that the blow-up of the determinization of Streett automata is inherently doubly exponential. This paper shows an exponential determinization construction for Streett automata. In fact, the complexity of our construction is roughly the same as the complexity achieved in Safra (1988) for Büchi automata. Moreover, a simple observation extends this upper bound to the complementation problem. Since any ω-automaton that admits exponential determinization can be easily converted into a Streett automaton, we have obtained a single procedure that can be used for all of these conversions. Furthermore, this construction is optimal (up to a constant factor in the exponent) for all of these conversions. Our results imply that Streett automata (with strong fairness as an acceptance condition) can be used instead of Büchi automata (with the weaker acceptance condition) without any loss of efficiency.

Key words. ω-automata, verification, reactive systems

AMS subject classification. 68Q45

DOI. 10.1137/S0097539798332518

1. Introduction.

Finite automata on infinite words (ω-automata), despite their seemingly fantastic definition, have quite an earthly role in the formal analysis of ongoing (reactive) systems. A *reactive system* is one whose goal is to continuously interact with its environment, as opposed to computing a function on an input and terminating. Take, for example, a file editor; it is not computing a function of a preset input, and its execution should not terminate, unless the environment so insists. (Other examples of such systems are control programs of a robot or an unmanned spacecraft.) Suppose one would like to make sure that a reactive system functions properly. For systems that compute functions, we need to verify that the system always terminates and computes the correct value of the function; what would be the reactive system's equivalent?

First, one needs to make sure that every reaction produced by the system is proper (safety), and second, that every anticipated reaction is eventually produced (liveness). In our file editor example, once an editor command is given, it must eventually be

*Received by the editors January 6, 1998; accepted for publication (in revised form) February 4, 2004; published electronically November 3, 2006. Part of this work was carried out while the author was at MIT and was supported in part by a Weizmann fellowship and NSF grant CCR-8912586. Part of this work was also carried out while the author was at IBM Almaden and Stanford University.
http://www.siam.org/journals/sicomp/36-3/33251.html
†Computer Science Department, Tel-Aviv University, 69978 Tel Aviv, Israel (safra@math.tau.ac.il).

carried out. This demonstrates the notion usually referred to as *weak fairness*:

- Weak fairness: Any continuously enabled action is eventually carried out.

Now suppose we are working on a paper[1] on an operating system that can run several file editors concurrently, but there is only one display on which we can see how the paper will look once printed. So, every now and then, while we continue to work on the paper, we try to display it, but each time there is someone else's paper already on display. The system might be "weakly fair" and yet not display the paper, since this action is not continuously enabled. Still, some might find it unfair if they try again and again to display their file but never get the chance. This demonstrates the stronger notion of fairness usually referred to as *strong fairness*:

- Strong fairness: Any action that is repeatedly enabled is eventually carried out.

Notice a problem here; suppose some action is enabled every now and then, and the computation ends without the action having been carried out. Just by observing a finite computation, how can one distinguish between the two cases (a) the system is not strongly fair, and (b) the system is slow and whoever wanted the action taken eventually gave up. (Decided to print the paper?)

Our solution is to interpret reactive systems over infinite computations; it does not mean we actually run infinite computations,[2] rather that we *analyze the fairness* of the system on infinite computations. On such computations we can distinguish between the case of a slow system and an unfair one, using algorithms that run in *finite* time. The two fairness conditions above look as follows:

- Weak fairness: A computation is unfair if there is an action that is enabled continuously from some point on but is carried out only finitely many times.
- Strong fairness: A computation is unfair if there is an action that is enabled infinitely often but is carried out only finitely many times.

Therefore, our computations are infinite objects (an infinite sequence for linear time, and an infinite tree for branching time), and the formal meaning (semantics) of a system is the set of computations it may produce. The specification of the system is given in some specification language (logic) over these infinite objects. In order to verify that the system functions properly, we check that the set of computations produced by the system is a subset of the computations that meet the specification.

For a complete exposition of the above subjects and related ones, the reader is referred to [HP85, Fra86, MP91].

Finite memory systems. We now restrict our attention to systems that can be described as finite-state machines (at least for the purpose of the formal analysis). It turns out that any reasonable logic for specification in the finite-state case describes a set of computations acceptable by a finite automaton over infinite objects (described below). Moreover, the most efficient decision procedures for these logics are usually obtained by translating a formula in the logic to an automaton and checking emptiness of the language this automaton accepts [VW86]. The most efficient procedures for the problem of model checking (checking that a program meets some specification) are also usually obtained using automata. A finite-state program can be viewed as a finite-state machine, and in order to check that it meets some specification, it is enough to check the *containment* of the language accepted by this finite-state machine in the language accepted by the specification automaton [VW86a].

[1] at the latest possible time to meet the deadline, quite naturally.

[2] if anyone has doubts.

There are two basic automata conversions that come up in these procedures: *complementation* and *determinization*.

This type of procedure was first suggested by Büchi [Büc62] in his original paper introducing ω-automata, in order to show that the validity of S1S (the monadic second order theory of one successor) is decidable. Büchi showed that ω-automata are closed under complementation; however, the blow-up of the complementation procedure he suggested is doubly exponential. McNaughton [McN66] showed that ω-automata can be determinized (into a deterministic automaton with a stronger acceptance condition than the one Büchi suggested). The blow-up of his determinization construction, however, is also doubly exponential. Rabin introduced tree automata and used McNaughton's result to show that these automata are closed under complementation. He could then give a decision procedure for a stronger logic—S2S (the monadic second order theory of many successors).

The decision of these logics is known to be nonelementary [Mey75], and thus there is no hope of achieving a reasonable complexity. However, when considering simpler logics and attempting to obtain more efficient procedures, the blow-up of the above constructions was prohibitive.

Sistla, Vardi, and Wolper [SVW87] showed an exponential complementation procedure for Büchi automata and utilized this result to obtain tight bounds for various logics. The exponential determinization of Büchi automata [Saf88], which also improves on [SVW87], was used [EJ88] to show a tight bound for the complexity of the decision procedure of various logics, which allow quantification over time-paths and thus require translation to tree automata (e.g., CTL*, Δ-PDL, μ-calculus, etc.). An exponential complementation for Streett automata was shown [SV89] and was utilized so as to improve the upper bound for the decision of linear-time logics that are translatable more efficiently to automata with strong fairness as an acceptance condition.

Finite automata over infinite objects. Automata on infinite words (ω-automata) are the same as automata on finite words except that, since a run over a word does not have a final state, the acceptance condition is on the set of states visited infinitely often in the run. The simplest acceptance condition was suggested by Büchi [Büc62], in which some of the states are designated as accepting, and a run is accepting if it visits infinitely many times the accepting set of states.

Muller [Mul63] suggested deterministic ω-automata, with a different acceptance condition, as a means of describing the behavior of nonstabilizing circuits. The acceptance condition he suggested is to specify explicitly all the "good" infinity sets (the *infinity set* of a run ξ is the set of states that occur infinitely many times in ξ). A run is accepting if its infinity set is one of the designated accepting sets. When we consider acceptance conditions based on the infinity set, this is obviously the most expressive condition.

A Rabin acceptance condition is, syntactically, a set of pairs of subsets of the states, $\{(L_i, U_i)\}_i$. A run ξ is accepting if, for one of the pairs i, ξ visits infinitely many times some states in L_i (the "good" states), and only finitely often the states in U_i (the "bad" states).

Streett [Str82] suggested the complementary condition to Rabin's condition, which is syntactically the same: a set of pairs of subsets of the states. A run ξ is accepting according to Streett's condition if, for all pairs i, if the run visits infinitely many times L_i it also visits infinitely many times U_i.

We may write Rabin's condition as $\bigvee_i L_i \wedge \neg U_i$ and Streett's condition as $\bigwedge_i L_i \rightarrow U_i$. Streett's condition corresponds to strong fairness (as defined above) since for each

event e the acceptance condition could contain a pair $\langle L_i, U_i \rangle$, in which L_i is the set of states in which e is enabled and U_i is the set of states in which e is taken (weak fairness can be expressed by Büchi automata).

Previous best results on determinization and complementation. In [Saf88, SV89, Kla91] the complexities of determinization and complementation of different classes of ω-automata were studied, and they were solved in full except for the complexity of determinization of Streett automata. An exponential complementation procedure was shown for Streett automata in [SV89] and with a better exponent in [Kla91]. It was shown [SV89] that the blow-up of the translation of Streett automata to any of the classes of ω-automata that were known to admit exponential determinization is inherently exponential.

Our results. Our main result (Theorem 1) is a new determinization construction for Streett automata. Given a Streett automaton with n state and h accepting pairs, we construct a deterministic Rabin automaton with $2^{\mathrm{O}(nh \log nh)}$ states and nh accepting pairs. Using the small number of accepting pairs in the determinized Rabin automaton, and a simple complementation construction for deterministic Rabin automata, which is exponential only in the number of accepting pairs (Lemma 3), we show that nondeterministic Streett automata can be converted into deterministic Streett automata with the same (exponential) blow-up (Corollary 4). Since the same deterministic automaton interpreted as a Streett or Rabin automaton accepts two complementary languages, this implies that Streett automata can be simultaneously complemented and determinized (codeterminized) into both Streett or Rabin automata with only an exponential blow-up.

The exact complexity of the complementation procedures obtained in this way matches the complexity of the complementation procedure of [Kla91].

The results reported herein were first published as an extended abstract [Saf92]. It is worthwhile noting that an independent work reported in [Wa93] may share some of these methods in a somewhat different setting.

2. Basic definitions. An *ω-automaton* over an *alphabet* Σ, $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, C \rangle$, consists of a finite set of *states* Q, an *initial state* $q_0 \in Q$, a *transition relation* $\delta \colon Q \times \Sigma \to 2^Q$, and an *acceptance condition* C. We extend δ to sets of states and sequences of letters in the usual way.

A sequence of states, $\xi \in Q^\omega$, is an *$\mathcal{A}$-run* over a word $\sigma \in \Sigma^\omega$ if $\xi_0 = q_0$ and, for every i, ξ_{i+1} is a σ_i successor of ξ_i, i.e., $\xi_{i+1} \in \delta(\xi_i, \sigma_i)$.

The *infinity set* of a sequence of letters (or states) σ, $\inf(\sigma)$, is the set of letters that appear infinitely many times in σ (i.e., $\inf(\sigma) = \{a \text{ s.t. } |\{i \text{ s.t. } \sigma_i = a\}| = \infty\}$).

An infinite word $\sigma \in \Sigma^\omega$ is *accepted* by an automaton $\mathcal{A}$ if there exists an accepting $\mathcal{A}$-run over σ. The *language* accepted by an automaton is the set of all words accepted by it.

An automaton is *deterministic* if for all $a \in \Sigma$, $q \in Q$, $|\delta(q, a)| = 1$, i.e., δ is a function into Q. Obviously, any word has exactly one run in a deterministic automaton.

We define classes of automata corresponding to the different acceptance conditions. We write N for nondeterministic and D for deterministic, and B, M, R, S for Büchi, Muller, Rabin, and Streett, respectively.

The acceptance conditions are summarized in the following table:

	Syntax	Semantics
B	$F \subseteq Q$	$\inf(\xi) \cap F \neq \phi$
M	$\mathbf{F} \subseteq 2^Q$	$\inf(\xi) \in \mathbf{F}$
R	$\bigvee_i L_i \wedge \neg U_i$	$\exists i\colon \ \inf(\xi) \cap L_i \neq \phi \ \wedge$ $\inf(\xi) \cap U_i = \phi$
S	$\bigwedge_i L_i \rightarrow U_i$	$\forall i\colon \ \inf(\xi) \cap L_i \neq \phi \ \rightarrow$ $\inf(\xi) \cap U_i \neq \phi$

Concerning the *size* of an automaton, we denote both the number of states and the size of the acceptance condition (except for Büchi automata, where the acceptance condition may be neglected). For example, our main result can be written as $\mathrm{NS}(n,h) \rightarrow \mathrm{DR}(2^{\mathrm{O}(nh\log(nh))}, nh)$.

3. Determinization of NS.

THEOREM 1. $\mathrm{NS}(n,h) \rightarrow \mathrm{DR}(2^{O(nh\log nh)}, nh)$*; i.e., for any* NS *automaton* $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \bigwedge_{0<i\leq h} L_i \rightarrow U_i\rangle$ *with* n *states and* h *acceptance pairs, there exists an equivalent* DR *automaton* $\mathcal{D} = \langle \Sigma, \widetilde{Q}, \widetilde{q}_0, \widetilde{\delta}, \bigvee_{0<i\leq nh} G_i \wedge \neg B_i\rangle$*, with* $2^{\mathrm{O}(nh\log(nh))}$ *states and* nh *acceptance pairs.*

Proof of Theorem 1. Throughout this proof we denote by H the set of indexes $[1..h]$.

Intuition. It is easier to look at the deterministic Rabin automaton $\mathcal{D}$ as a program with bounded memory and some infinitary acceptance condition. This program reads the input one letter at a time and changes its memory accordingly. The corresponding finite ω-automaton has a different state for each of the possible states of the program's memory. The infinite string is accepted if the set of memory states visited infinitely often satisfies the acceptance condition. We now describe $\mathcal{D}$ informally.

An accepting $\mathcal{A}$-run ξ has a *witness set* $J \subseteq H$ for which ξ visits infinitely many times each U_j for $j \in J$ and only finitely many time any L_j for $j \notin J$.

Given a witness set J, one can construct a small nondeterministic Büchi automaton that accepts all strings for which there is an accepting run ξ with witness set J. This automaton consists of two parts; the first one is a copy of $\mathcal{A}$ (without the acceptance condition). Each run at each point can nondeterministically guess that no state in any of the sets L_j, for $j \notin J$, will be visited from now on, and choose to move to the second part. The second part consists of $|J|+1$ copies of $\mathcal{A}$, in which the run can move to the next copy only after visiting the set U_j corresponding to the current copy. A run is accepting if it cycles infinitely through all the copies. All states $q \in L_j$ for $j \notin J$ are removed from all the copies of $\mathcal{A}$ in the second part. Hence an accepting run visits only finitely many times copies of $q \in L_j$ for $j \notin J$, and infinitely many times copies of $q \in U_j$ for each $j \in J$.

This automaton can be determinized with only an exponential blow-up [Saf88]. However, since the number of possible witness sets is exponential, a construction of an automaton that deterministically considers all the witness sets results in a doubly exponential blow-up.

The determinization construction suggested here may be viewed as a deterministic dynamic process that at each point in time considers only a polynomial number of witness sets.

The deterministic automaton $\mathcal{D}$, while maintaining the subset of $\mathcal{A}$-states reached by reading the prefix of the input, starts by assuming that the witness set of the accepting run (if one exists) is H, i.e., $\mathcal{D}$ tries, during each run, to cycle through all the U_j's. Whenever a run is waiting to visit some U_{j_1}, $\mathcal{D}$, assuming (the worst) that the run will never again visit U_{j_1}, spawns off a parallel construction, with possibly a

smaller subset of the $\mathcal{A}$-states, and with the witness set $J' = J \setminus \{j_1\}$ (disallowing any state $q \in L_{j_1}$ in the subprocess). Any run that eventually visits U_{j_1} is advanced to the next index. In the subprocess, if again a run is waiting for U_{j_2}, $\mathcal{D}$ branches off recursively with a smaller witness set. An important observation is that, for each such parallel process and for each state that appears in the subset maintained by the process, one needs to consider only one index—the most advanced one; hence the subset of the $\mathcal{A}$-states maintained by the process is partitioned among the different indexes. In the good case, in which eventually all $\mathcal{A}$-states have runs that completed a cycle, all the subprocesses are killed and the process is restarted. If any of these processes is restarted infinitely many times, $\mathcal{D}$ accepts.

However, suppose that some runs completed a cycle, but some other runs are stuck waiting for some U_{j_1}. The latter runs prevent the former runs from restarting the process. Therefore, for all runs that completed a cycle through U_j for every $j \in J$, $\mathcal{D}$ spawns off a parallel process with a smaller set of $\mathcal{A}$-states (this is similar to the determinization construction of [Saf88]). In addition (again following [Saf88]) $\mathcal{D}$ maintains an order among the subprocesses according to which subprocess was spawned first. Whenever a state appears in more than one subprocess (with the same index; otherwise, as mentioned above, the more advanced index takes priority) it is removed from all but the one spawned first.

Since for each state in each process one needs to consider only one U_j it is waiting for, the number of witness sets we need to try in parallel at any given time is polynomial.

We now return to the formal proof of Theorem 1. We start with some definitions we need for the construction of the set $\widetilde{Q}$ of states of $\mathcal{D}$.

Let $V = [1..2nh]$ be the set of *names* (these are used by $\mathcal{D}$ to preserve the identity of different parallel applications of some basic construction).

For $S \subseteq Q$, let an *S-atom* be $\langle v, S\rangle$, where $v \in V$.

For $S \subseteq Q$ and $J \subseteq H$, we give a recursive definition of an *(S, J)-decomposition*:

1. An S-atom is an (S, J)-decomposition.
2. Let $v \in V$.
   Let $S_1, \ldots, S_l$ be a partition of S (i.e., $\bigcup_i S_i = S$ and for any $i \neq j \in [1..l]$, $S_i \cap S_j = \phi$).
   Let $j_1, \ldots, j_l \in J \cup \{0\}$, where at least one of the j's is nonzero. Denote by J_i, for each $i \in [1..l]$, the set $J \setminus \{j_i\}$.
   Let $\Pi_1, \ldots, \Pi_l$ be such that for each $i \in [1..l]$, Π_i is an (S_i, J_i)-decomposition.
   Then $\langle v, (\Pi_1, j_1), \ldots, (\Pi_l, j_l)\rangle$ is an (S, J)-decomposition.

For an (S, J)-decomposition we refer to the decompositions used in the recursion of this definition as *subdecompositions*.

An (S, J)-decomposition has a *good name* if the names assigned to each of the subdecompositions are all different. We consider, from now on, only decompositions with a good name.

Note that the recursion in the definition of an (S, J)-decomposition is finite, since not all the indexes j can be 0, and thus either the set of states S or the set of indexes J decreases each level down the recursion.

We can even give a more specific bound on the size and number of decompositions, as follows.

LEMMA 2. *The number of subdecompositions in an (S, H)-decomposition ($S \subseteq Q$) is at most nh, and the total number of (S, H)-decompositions ($S \subseteq Q$) is at most $2^{O(nh \log(nh))}$.*

Proof. For an (S, J)-decomposition Π we say that a pair (q, j), for $q \in Q$, $j \in H$, is

special for a (S', J')-subdecomposition Π' of Π if the following three conditions hold:

- $q \in S'$;
- the index set of the immediate subdecomposition of Π' that contains q is a strict subset of J' (i.e., this subdecomposition is not indexed by 0);
- if a subdecomposition has index set $J'' \neq J'$ such that $J' \subseteq J''$, then $j \in J''$.

Now, each pair (q, j) is special for a subdecomposition, and each subdecomposition Π' has a pair (q, j) which is special for it. Therefore, Π can be represented as a partial function from the set of pairs (q, j) to the set of names V. □

The construction of $\mathcal{D}$.

The set of states. $\widetilde{Q}$ is the set of all (S, H)-decompositions for a subset of the states $S \subseteq Q$.

The initial state. $\widetilde{q}_0 = \langle 1, \{q_0\}\rangle$, i.e., the $\{q_0\}$-atom with 1 (arbitrarily) as its name.

The transition function. For each $\mathcal{D}$-state $\widetilde{q} \in \widetilde{Q}$ and a letter $a \in \Sigma$, $\widetilde{\delta}(\widetilde{q}, a)$ is the result of applying the following sequence of operations to $\widetilde{q}$:

1. Replace each atom $\langle v, S\rangle$ in $\widetilde{q}$ by $\langle v, \delta(S, a)\rangle$.
   This results in some structure that may violate the requirement, in the definition of an (S, J)-decomposition above, that the sets $S_1, \ldots, S_l$ be disjoint. At the end of the following several steps, this requirement is restored.
2. For any nonatomic (S, J)-subdecomposition in $\widetilde{q}$, let $j_1, \ldots, j_l$ be the indexes as in the definition of a decomposition above, and let $S'_1, \ldots, S'_l$ be its sets of $\mathcal{A}$-states after step 1. For each $\mathcal{A}$-state q and i such that $q \in S'_i$,
   - if $q \in L_{j_i}$, then remove q from S'_i and append to the list of subdecompositions an atom $\langle v, \{q\}\rangle$ with index $j = \max\{J\}$;
   - otherwise, if $q \in U_{j_i}$, then append an atom $\langle v, \{q\}\rangle$ with index $j = \max\{(J \cup \{0\}) \cap \{0, \ldots, j_i - 1\}\}$.

   In both cases $v \in V$ is an unused name in $\widetilde{q}$, and each new atom is assigned a different name. This is possible since, by Lemma 2, only nh out of $2nh$ names in V are used in $\widetilde{q}$.
   For atomic (S, J)-subdecompositions, we follow the same procedure, assuming S is a one-item list with the maximal index in J as its index.
3. For a nonatomic subdecomposition in $\widetilde{q}$ for which, after the previous steps, an $\mathcal{A}$-state q appears in a set S'_i with index j and a set $S'_{i'}$ with index $j' > j$, remove q from $S'_{i'}$.
4. For a nonatomic subdecomposition in $\widetilde{q}$ for which, after the previous steps, an $\mathcal{A}$-state q appears in a set S'_i and a set $S'_{i'}$, where $i' > i$, remove q from $S'_{i'}$.
5. Remove any empty set from any list.
6. Replace any nonatomic (S, J)-subdecomposition whose name is v, in which after the previous steps all indexes are 0, by an atom $\langle v, S\rangle$.

The acceptance condition. For each name $v \in V$, let G_v be the set of states $\widetilde{q}$ in which v is the name of an atom, and let B_v be the set of states $\widetilde{q}$ in which v is not used in the decomposition.

Correctness. $L(\mathcal{D}) \subseteq L(\mathcal{A})$: Given that there is a name v, which in the $\mathcal{D}$-run ξ over a word σ is used (in the decompositions ξ_i) continuously from some point on, and is the name of an atom infinitely many times, we prove that there is an accepting $\mathcal{A}$-run over σ.

For two positions $0 \leq l < k$, we denote by $\sigma[l, k)$ the finite word $\sigma_l, \ldots, \sigma_{k-1}$.

Let l be the largest such that v is not used in the decomposition ξ_l, and let S_i, for $i > l$, be the set such that v is the name of an (S_i, J)-subdecomposition of ξ_i. Let

$l_1 < l_2 < \cdots$ (where $l_1 > l$) be the positions at which v is the name of an atom in ξ_{l_k}. By the construction, $S_{l_1} \subseteq \delta(q_0, \sigma[1, l_1))$. The condition to make a subdecomposition become an atom (step 6) and the conditions to create new subdecompositions and maintain them (steps 2 and 1) ensure that for each $q \in S_{l_{k+1}}$, for $k > 0$, there exists some $q' \in S_{l_k}$, as well as an $\mathcal{A}$-run over $\sigma[l_k, l_{k+1})$ which leads from q' to q while visiting all U_j for $j \in J$ and no L_j for $j \notin J$.

Intending to use König's lemma, we construct a tree whose nodes are all the pairs of the form (q, k) for $q \in S_{l_k}$. As the parent of a node $(q, k+1)$ we pick one of the pairs (q', k) such that $q' \in S_{l_k}$ and there exists an $\mathcal{A}$-run from q' to q, as described above. The root of the tree is $(q_0, 0)$.

By König's lemma, since there are infinitely many pairs, and the number of pairs at each level of the tree is bounded, there is an infinite path, $(q_0, 0), (q_1, 1), \ldots$, in the tree. By the construction of this tree, for each $k > 0$ there is an $\mathcal{A}$-run, as described above, from q_k to q_{k+1}, over $\sigma[l_k, l_{k+1})$. The infinite concatenation of these segments gives an $\mathcal{A}$-run over σ which visits each $\mathcal{A}$-state in U_j for $j \in J$ infinitely many times, and visits any state in L_j for $j \notin J$ only at the first segment. This $\mathcal{A}$-run is accepting.

$L(\mathcal{A}) \subseteq L(\mathcal{D})$: Given that, for a string σ, there is an accepting $\mathcal{A}$-run, ξ, we prove that there is a name v which in the $\mathcal{D}$-run over σ is used continuously from some point on and is the name of an atom infinitely many times.

Let l be the length of the finitary prefix of ξ; i.e., every state ξ_k for $k > l$ appears infinitely many times in ξ. Let the ith state $\widetilde{q}_i$ of the $\widetilde{\mathcal{D}}$-run over σ be an (S_i, H)-decomposition; then it must be that $\xi_i \in S_i$. Therefore, S_i is never empty and its name v_1 remains fixed in all $\widetilde{q}_i$. If $\widetilde{q}_i$ becomes an atom infinitely many times, we are done. Otherwise, let i_1 be the largest such that $\widetilde{q}_i$ is an atom. For $i > i_1$, as i increases, the $\mathcal{A}$-state ξ_i appears in subdecompositions with monotonically nonincreasing index, and thus its index is eventually fixed. Thereafter, ξ_i can move only closer to the beginning of the sequence of immediate subdecompositions. Hence, there is an i'_1 such that for all $i > i'_1$, ξ_i appears in a subdecomposition with a fixed name v_2. We can now repeat the argument and show that if v_2 is not a name of an atom infinitely many times, then eventually the state of ξ appears one level down in the decomposition. Since the depth of the decomposition is finite, there must be a subdecomposition which becomes an atom infinitely many times.

This completes the proof of our main theorem. □

4. Complementation of DR. In order to see how to codeterminize (construct a deterministic automaton that accepts the complement) Streett automata, we need the following lemma.[3]

LEMMA 3. $\mathrm{DS}(n, h) \to \mathrm{DR}(n \cdot 2^{h \log h}, h + 1)$*; i.e., for any deterministic Streett automaton with* n *states and* h *accepting pairs, there exists an equivalent deterministic Rabin automaton with* $n \cdot 2^{h \log h}$ *states and* $h + 1$ *accepting pairs.*

One should comment here that one can quite easily complement such automata while leaving the set of states fixed, translating the acceptance condition from Rabin's to Streett's. Such a translation, however, would incur an exponential blow-up in the size of the acceptance condition. The above lemma is useful when applying it to the main theorem, as the size of the acceptance condition remains polynomial in the size of the original automaton, while the number of states becomes exponential.

Proof. We show an explicit construction, given a DS automaton, $\mathcal{D} = \langle \Sigma, Q, q_0, \delta,$

[3]This lemma appears in [Saf88] but only in the journal version and is repeated here for the 21st century reader.

$\bigwedge_{1\le i\le h} L_i \to U_i\rangle$, of a DR automaton, $\widetilde{\mathcal{D}} = \langle \Sigma, \widetilde{Q}, \widetilde{q}_0, \widetilde{\delta}, \bigvee_{1\le i\le h+1} \neg\widetilde{L}_i \wedge \widetilde{U}_i\rangle$.

Intuition. Again let us think of the automaton $\widetilde{\mathcal{D}}$ as a program with bounded memory; the states of $\widetilde{\mathcal{D}}$ will be all possible data states that this program's memory can be in.

$\widetilde{\mathcal{D}}$ maintains, aside from the state $\mathcal{D}$ reaches after reading the prefix of the input, a permutation of the set of indexes $[1..h]$. Whenever a state in some U_j is visited, the index j is moved to the end of the permutation (if several are visited, all of their indexes are moved to the end with no particular order). Hence, for every accepting run, let J be its witness set and let $i = h - |J|$; then eventually the first i elements of the permutation are fixed, and for each index j in the suffix of the permutation, U_j is visited infinitely many times. The Rabin acceptance condition contains, for each $i \in [0..h]$, a pair in which the "bad" set contains all states in which some L_j for j in the first i elements in the permutation is visited, and the "good" set contains all states in which U_j is visited for j being the $(i+1)$th element in the permutation.

We now give the formal proof of the lemma.

The construction. The states of $\widetilde{\mathcal{D}}, \widetilde{Q}$ have the form of a tuple, (q, π, r, g), where $q \in Q$, π is a permutation of $[1..h]$, and $r, g \in [1..h+1]$. The initial state $\widetilde{q}_0 = (q_0, \langle 1, \ldots, h\rangle, h+1, h+1)$.

Consider a state $\widetilde{q} = (q, \pi, r, g)$, where $\pi = \langle i_1, \ldots, i_h\rangle$. For a letter $a \in \Sigma$ define $\widetilde{\delta}(\widetilde{q}, a)$ to be the state $\widetilde{q}' = (q', \pi', r', g')$ as follows:

- $q' = \delta(q, a)$.
- g' is the minimal index i such that $q' \in U_{j_i}$, if it exists, and otherwise $g' = h+1$.
- r' is the minimal index i such that $q' \in L_{j_i}$, if it exists, and otherwise $r' = h+1$.
- $\pi' = \langle j_1, \ldots, j_{g'-1}, j_{g'+1}, \ldots, j_h, j_{g'}\rangle$ if $g' \le h$; otherwise, $\pi' = \pi$.

For i, $1 \le i \le h+1$, $\widetilde{L}_i$ consists of all the states $\widetilde{q}$ in which $r < i$, and $\widetilde{U}_i$ consists of all the states $\widetilde{q}$ in which $g = i$.

Note that after reading a finite prefix of the input, there is a part to the left of π which is fixed from then on, and that contains all the indexes j such that U_j is visited only finitely many times. If that prefix is of length i, then from then on $g > i$.

$L(\widetilde{\mathcal{D}}) \subseteq L(\mathcal{D})$: Assume that for some i, $r \ge i$ from some point on, and $g = i$ infinitely many times. Since $g = i$ infinitely many times, for every j, if U_j is visited only finitely many times, eventually j is placed in π with an index smaller than i ($j = j_k$ for $k < i$), and by the construction, from then on, L_j is never visited.

$L(\mathcal{D}) \subseteq L(\widetilde{\mathcal{D}})$: Assume there is an accepting $\mathcal{D}$-run; then there exists a maximal set $J \subseteq [1..h]$ such that for $k \in J$, U_{j_k} is visited infinitely many times, and for $k \notin J$, neither L_{j_k} nor U_{j_k} is visited from some point on. There exists a further point, after which $[1..h] \setminus J$ occupies the leftmost positions in π, and none of its indexes changes its place. Let $i = h - |J|$. Obviously, $r \ge j$ beyond this point. We claim that $g = i$ infinitely many times. At any point, let $j_i = k$. Since the run visits U_k infinitely many times, on the next visit to U_k, g will be i.

Complexity. The number of states is $n \cdot h! \cdot (h+1)^2 < n \cdot 2^{h \log h}$ (for $h > 5$). ∎

Since in their deterministic versions the same automaton interpreted as a Rabin automaton and as a Streett automaton accept two complementary languages, we can conclude the following.

COROLLARY 4. $\overline{\mathrm{NS}(n,h)} \to \mathrm{DR}(2^{\mathrm{O}(nh\log(nh))}, nh+1)$. ∎

Hence, NS can be translated to DS with this complexity.

5. Complementation of Streett tree automata. Rabin [Rab69, Rab70] introduced automata on infinite trees. The input of such an automaton is an infinite binary tree, whose nodes are labeled by letters from some finite alphabet Σ, $T\colon \{0,1\}^* \to \Sigma$. A Σ-tree automaton, $\mathcal{T} = \langle \Sigma, Q, q_0, \delta, C\rangle$, is similar to an ω-automaton except that the transition function specifies, for a state $q \in Q$ and a letter $a \in \Sigma$, a set of pairs of states containing both a left successor state q_l and a right successor state q_r (i.e., $\delta\colon Q \times \Sigma \to 2^{Q\times Q}$). A $\mathcal{T}$-run is a Q-tree, $\Gamma\colon \{0,1\}^* \to Q$, in which the root node is q_0, and all the nodes satisfy δ, i.e., for each node $p \in \{0,1\}^*$, $(\Gamma(p0), \Gamma(p1)) \in \delta(\Gamma(p), T(p))$. The acceptance condition C is any ω-condition, such as those of Büchi, Rabin, Muller, or Streett. A $\mathcal{T}$-run Γ is defined to be accepting if the infinity set of *every* infinite path in Γ satisfies C.

Given a tree automaton $\mathcal{A}$, the complementary tree automaton $\bar{\mathcal{A}}$ may be viewed as supplying a proof, given some input tree, that every nondeterministic $\mathcal{A}$-run has a path that does not satisfy $\mathcal{A}$'s acceptance condition. Following the procedure of Gurevich and Harrington [GH82] the proof is supplied by a finite memory strategy. A finite memory strategy is one that can be represented by a set of finite tables, for each node in the tree, giving for each state and some finite information about the history of the run so far either a left direction or a right direction. Such a strategy serves as a proof that the input tree is not accepted by $\mathcal{A}$ if, for any choice of nondeterministic moves, and following the direction the strategy supplies for each move, the resulting infinite sequence of states is not accepted according to $\mathcal{A}$'s acceptance condition.

Using the techniques of [EJ91] it can be shown [Jutla] that the complement of a Streett tree automaton has a memoryless strategy; i.e., $\bar{\mathcal{A}}$, for each node in the input tree, needs to guess a table showing for each *state* (i.e., no information at all on the history of the run) the direction to go for a nonaccepting path. Using the exponential determinization for Streett automata shown here this implies an exponential complementation procedure for Streett tree automata. A different proof for this theorem appeared first in [Kla92].

Discussion. This paper showed that ω-automata with a strong fairness acceptance condition can replace Büchi automata in all applications without loss of efficiency. This should have further applications than those shown here, in the formal analysis of finite-state systems, and, in general, for a better understanding of the notion of fairness.

The main result of this paper has applications with regards to the complementation of tree automata: Streett tree automata were shown to have exponential complementation construction [Kla92]; the results reported herein imply a simple such exponential complementation procedure for these automata. The main open problem left in this area is the complexity of complementation of Rabin tree automata; is it doubly exponential, or can one show an upper bound similar to the one for Streett tree automata?

Acknowledgments. I would like to thank Moshe Vardi and Amir Pnueli for many most insightful discussions on this problem, and the unnamed referees for most constructive comments.

REFERENCES

[BL69] J. R. Büchi and L. H. Landweber, *Solving sequential conditions by finite-state strategies*, Trans. Amer. Math. Soc., 138 (1969), pp. 295–311.

[Büc62] J. R. Büchi, *On a decision method in restricted second order arithmetics*, in Proceedings of the International Congr. on Logic, Method. and Phil. of Sci., 1960, E. Nagel et al., eds., Stanford University Press, Stanford, CA, 1962, pp. 1–12.

[EJ88] A. E. Emerson and C. Jutla, *The complexity of tree automata and logics of programs*, in Proceedings of the 29th IEEE Symposium on Foundations of Computer Science, 1988, pp. 328–337.

[EJ91] A. E. Emerson and C. Jutla, *Tree automata mu-calculus and determinacy*, in Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science, 1991, pp. 368–377.

[ES84] A. E. Emerson and P. A. Sistla, *Deciding full branching time logic*, Inform. and Control, 61 (1984), pp. 175–201.

[Fra86] N. Francez, *Fairness*, Springer-Verlag, New York, 1986.

[GH82] Y. Gurevich and L. Harrington, *Trees, automata, and games*, in Proceedings of the 14th ACM Symposium on Theory of Computing, 1982, pp. 60–65.

[HP85] D. Harel and A. Pnueli, *On the development of reactive systems*, in Logics and Models of Concurrent Systems, K. R. Apt, ed., Springer, Berlin, 1985, pp. 477–498.

[Jutla] C. Jutla, *personal communication.*

[Kla91] N. Klarlund, *Progress measures for complementation of ω-automata with application to temporal logic*, in Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science, 1991, pp. 358–367.

[Kla92] N. Klarlund, *Progress measures, immediate determinacy, and a subset construction for tree automata*, in Proceedings of the 7th IEEE Symposium on Logic in Computer Science, 1992, pp. 382–393.

[KT90] D. Kozen and J. Tiuryn, *Logics of programs*, in Handbook of Theoretical Computer Science, Vol. B, J. Van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 789–840.

[MP91] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification*, Springer, New York, 1991.

[McN66] R. McNaughton, *Testing and generating infinite sequences by a finite automaton*, Inform. and Control, 9 (1966), pp. 521–530.

[Mey75] A. R. Meyer, *Weak monadic second order theory of successor is not elementary recursive*, in Proceedings of the Boston University Logic Colloquium, 1973, Lecture Notes in Math. 453, Springer, Berlin, 1975, pp. 132–154.

[Mul63] D. E. Muller, *Infinite sequences and finite machines*, in Proceedings of the 4th IEEE Symposium on Switching Circuit Theory and Logical Design, 1963, pp. 3–16.

[Pec86] J. P. Pecuchet, *On the complementation of Büchi automata*, Theoret. Comput. Sci., 47 (1986), pp. 95–98.

[Rab69] M. O. Rabin, *Decidability of second-order theories and automata on infinite trees*, Trans. Amer. Math. Soc., 141 (1969), pp. 1–35.

[Rab70] M. O. Rabin, *Weakly definable relations and special automata*, in Proceedings of the Symposium on Mathematical Logic and Foundation of Set Theory, Y. Bar-Hillel, ed., North-Holland, Amsterdam, 1970, pp. 1–23.

[Saf88] S. Safra, *On the complexity of ω-automata*, in Proceedings of the 29th IEEE Symposium on Foundations of Computer Science, 1988, pp. 319–327. An extended version to appear in J. Comput. System Sci.

[Saf92] S. Safra, *Exponential determinization for ω-automata with strong-fairness acceptance condition (extended abstract)*, in Proceedings of the 24th ACM Symposium on Theory of Computing, 1992, pp. 275–282.

[SV89] S. Safra and M. Y. Vardi, *On ω-automata and temporal logic*, in Proceedings of the 21st ACM Symposium on Theory of Computing, 1989, pp. 127–137.

[SVW87] A. P. Sistla, M. Y. Vardi, and P. Wolper, *The complementation problem for Büchi automata with application to temporal logic*, Theoret. Comput. Sci., 49 (1987), pp. 217–237.

[Str82] R. S. Streett, *Propositional dynamic logic of looping and converse is elementary decidable*, Inform. and Control, 54 (1982), pp. 121–141.

[Var85] M. Y. Vardi, *Automatic verification of probabilistic concurrent finite-state programs*, in Proceedings of the 26th IEEE Symposium on Foundations of Computer Science, 1985, pp. 327–338.

[VS85] M. Y. Vardi and L. Stockmeyer, *Improved upper and lower bounds for modal logics of program*, in Proceedings of the 17th ACM Symposium on Theory of Computing, 1985, pp. 240–251.

[VW86] M. Y. Vardi and P. Wolper, *Automata theoretic techniques for modal logics of programs*, J. Comput. System Sci., 32 (1986), pp. 183–221.

[VW86a] M. Y. Vardi and P. Wolper, *An automata-theoretic approach to automatic program verification*, in Proceedings of the 1st IEEE Symposium on Logic in Computer Science, 1986, pp. 332–344.

[Wa93] I. Walukiewicz, *A complete deductive system for the μ-calculus*, in Proceedings of the 8th IEEE Symposium on Logic in Computer Science, 1993, pp. 136–147.

SIAM J. COMPUT.
Vol. 36, No. 3, pp. 815–834

COMPUTING MAXIMALLY SEPARATED SETS IN THE PLANE*

PANKAJ K. AGARWAL†, MARK OVERMARS‡, AND MICHA SHARIR§

Abstract. Let S be a set of n points in $\mathbb{R}^2$. Given an integer $1 \le k \le n$, we wish to find a *maximally separated subset* $I \subseteq S$ of size k; this is a subset for which the minimum among the $\binom{k}{2}$ pairwise distances between its points is as large as possible. The decision problem associated with this problem is to determine whether there exists $I \subseteq S$, $|I| = k$, so that all $\binom{k}{2}$ pairwise distances in I are at least 2. This problem can also be formulated in terms of disk-intersection graphs: Let D be the set of unit disks centered at the points of S. The *disk-intersection* graph G of D has as edges all pairs of disks with nonempty intersection. Any set I with the above properties is then the set of centers of disks that form an independent set in the graph G. This problem is known to be NP-complete if k is part of the input. In this paper we first present a linear-time ε-approximation algorithm for any constant k. Next we give exact algorithms for the cases $k = 3$ and $k = 4$ that run in time $O(n^{4/3}\text{polylog}(n))$. We also present a simpler $n^{O(\sqrt{k})}$-time exact algorithm (as compared with the recent algorithm in [J. Alber and J. Fiala, *J. Algorithms*, 52 (2004), pp. 134–151]) for arbitrary values of k.

Key words. disk-intersection graphs, independent set, geometric optimization

AMS subject classifications. 68Q25, 68U05, 68W25

DOI. 10.1137/S0097539704446591

1. Introduction.

Let S be a set of n points in the plane. We are interested in finding a small subset I of S such that all the pairwise distances between points in I are large. To be more precise, let I be a subset of S of cardinality k, for $1 \le k \le n$. We define the *separation distance* $d_{\text{sep}}(I)$ to be the minimum among the $\binom{k}{2}$ pairwise distances between its k points. We call I *δ-separated* if $d_{\text{sep}}(I) \ge \delta$. We call I a *maximally separated* subset of S if $d_{\text{sep}}(I) \ge d_{\text{sep}}(I')$ for all subsets $I' \subseteq S$ of size k. Let $d^k_{\text{sep}}(S) = \max_{I \subseteq S,\ |I|=k} d_{\text{sep}}(I)$.

In this paper we study algorithms for computing such maximally separated subsets. We consider small (constant) values of k, but we also address the general case. For the case $k = 2$ the problem is equivalent to finding a diametral pair of S and thus can be solved (exactly) in $O(n \log n)$ time [10], and can be ε-approximated in linear time (see, e.g., [1]). For larger k, the problem becomes considerably more complicated, and is known to be NP-complete if k is part of the input [9].

*Received by the editors November 29, 2004; accepted for publication (in revised form) April 6, 2006; published electronically November 3, 2006. The work of P.A. and M.S. was supported by a grant from the U.S.-Israeli Binational Science Foundation. Work by P.A. was also supported by NSF under grants CCR-00-86013, EIA-98-70724, EIA-99-72879, EIA-01-31905, and CCR-02-04118. Work by M.S. was also supported by NSF grants CCR-97-32101 and CCR-00-98246, by a grant from the Israel Science Fund (for a Center of Excellence in Geometric Computing), and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University. Part of the research was done during the 2003 Bellairs Workshop on Computational Geometry, organized by Godfried Toussaint, and the 2003 Dagstuhl Workshop on Computational Geometry. A preliminary version of this paper has appeared in *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, New Orleans, 2004, pp. 509–518.

http://www.siam.org/journals/sicomp/36-3/44659.html

†Department of Computer Science, Duke University, Durham, NC 27708-0129 (pankaj@cs.duke.edu).

‡Department of Computer Science, Utrecht University, Utrecht, The Netherlands (markov@cs.uu.nl).

§School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012 (michas@post.tau.ac.il).

Finding small well-separated subsets is important in certain pattern-matching problems, where the points in the subset form a representation of the total set of points. For example, Vleugels and Veltkamp [23] describe a method for fast indexing of multimedia databases using so-called *vantage objects*. These vantage objects are points in the feature space for the matching problem. It has been observed that, for the application at hand, the chosen vantage objects best be well-separated.

The decision problem associated with the problem of computing a maximally separated subset of size k calls for determining whether a δ-separated subset I of size k exists for a given $\delta > 0$. This problem can also be formulated in terms of disk-intersection graphs: Let D be the set of disks of radius $\delta/2$ centered at the points of S. The *disk-intersection* graph G of D has the disks as nodes and two disks are connected by an edge if they intersect. Clearly, a δ-separated subset I is the set of centers of an independent set in G (and vice versa). So the decision problem is equivalent to the problem of finding an independent set of size k in the disk-intersection graph G. Recently, the problem of computing the maximum independent set in intersection graphs has attracted considerable attention because of its application in geographic information systems (GIS); see [2, 11, 12] and references therein.

Related work. The problem of computing an independent set in a graph is one of the earliest problems known to be NP-complete [13]. In fact, for a general graph with n vertices, there cannot be a polynomial-time algorithm with approximation ratio better than $n^{1-\varepsilon}$, for any $\varepsilon > 0$, unless $NP = ZPP$ [15]. The best known polynomial-time algorithm finds an independent set of size $\Omega((\kappa \log^2 n)/n)$, where κ is the size of the maximum independent set in the graph [6]. However, better approximation algorithms are known for intersection graphs of geometric objects. The maximum independent set in the intersection graph of intervals on a line can be computed in polynomial time, but the problem remains NP-complete for intersection graphs of orthogonal segments, unit disks, and unit squares [9]. Still, ε-approximation algorithms have been proposed for intersection graphs of unit disks, unit squares, arbitrary disks, and fat objects [7, 9, 11, 17, 19, 20, 22], and an $O(\log n)$-approximation algorithm is known for intersection graphs of rectangles [2].

Little is known about computing maximally separated sets. Formann and Wagner [12] developed a 2-approximation algorithm under the L_∞-metric. Alber and Fiala [5] present an algorithm that computes, in time $n^{O(\sqrt{k})}$, an independent set of cardinality k in the intersection graph of a set of disks. Their algorithm, however, is rather complicated, and they do not consider cases involving small values of k. Moreover, since they compute all pairwise distances between input points, their algorithm takes $\Omega(n^2)$ time even for small values of k.

Our results. In this paper we consider both the general problem and special instances of it that involve small values of k, and develop exact and approximation algorithms for these problems. The paper contains four main results:

(i) For any constant k, we present in Section 2 a simple, linear-time algorithm that returns a subset I of size k such that $d_{\text{sep}}(I) \geq (1-\varepsilon)d^k_{\text{sep}}(S)$. Such an approximation algorithm is suitable for the pattern-matching application mentioned above.

(ii, iii) We present in Sections 3 and 4 $O(n^{4/3}\text{polylog}(n))$-time algorithms for computing (exactly) maximally separated subsets of size 3 and 4, respectively.

(iv) We also present, in Section 5, a simpler $n^{O(\sqrt{k})}$-time exact algorithm (as compared with the algorithm in [5]) for arbitrary values of k.

2. An ε-approximation algorithm. In this section we show that, for any constant k and for any constant $0 < \varepsilon < 1$, we can find in linear time a subset I of S of cardinality k such that $d_{\text{sep}}(I) \geq (1-\varepsilon)d^k_{\text{sep}}(S)$. The running time is exponential in k. We refer to such an I as an *ε-approximation* of the optimal solution.

As a warm-up exercise let us consider the case $k = 2$. We want to find an ε-approximation of the diameter of the set S in linear time. This is an already solved problem (see, e.g., [1]), but we sketch a solution (a) for the sake of completeness, and (b) to prepare for tackling the general case $k \geq 3$.

Let B be the axis-parallel bounding box of S. Let w be the width of B and h its height, and let us assume, without loss of generality, that $w \geq h$. Clearly, the diameter d lies between w and $\sqrt{2}w$. Choose $\delta = \varepsilon w/2\sqrt{2}$ and divide the box B into $O\left(1/\varepsilon^2\right)$ squares of size $\delta \times \delta$. In each nonempty square τ we pick a single point from $\tau \cap S$ and retain only the highest and lowest point in each row or column of the grid. So we end up with a subset S' of S with $O(1/\varepsilon)$ points. We compute the diameter d' of S' exactly, which takes $O((1/\varepsilon)\log(1/\varepsilon)) = O(1)$ time. Now it is easy to see that the actual diameter d satisfies

$$d \leq d' + 2\sqrt{2}\delta.$$

So

$$d' \geq d - 2\sqrt{2}\delta \geq d - \varepsilon w \geq (1-\varepsilon)d,$$

since $d \geq w$. Hence the diameter of the set S' is an ε-approximation for the diameter of S. As computing the bounding box and the set S' takes $O(n)$ time, this procedure computes an ε-approximate diameter of S in $O(n + (1/\varepsilon)\log(1/\varepsilon))$ time.

Let us next assume that $k \geq 3$. Our algorithm uses recursion on k. As above, we compute the smallest axis-parallel bounding box B of S, denote its width and height as w and h, respectively, and assume that $w \geq h$.

We first consider the case in which $d^k_{\text{sep}}(S) \leq w/(k+1)$. (Note that this case cannot arise for $k = 2$.) We subdivide the box B into $k+1$ vertical strips $s_0, \ldots, s_k$, each of width $w/(k+1)$, and set $S_i := S \cap s_i$, for $i = 0, \ldots, k$. Any solution will use points from at most k of these $k+1$ strips. Therefore, for each strip s_i, we compute an ε-approximation of a maximally separated set in $S \setminus S_i$. The best among those $k+1$ solutions is the answer we are looking for.

For each $i = 0, \ldots, k$, we process $S \setminus S_i$ as follows. A crucial observation is that, for $1 \leq i \leq k-1$, we may assume that the optimal solution uses points that *lie on both sides* of s_i. Indeed, if an optimal solution I consists only of points that lie, say, to the right of s_i, replace I by $I' \cup \{p_L\}$, where p_L is the leftmost point of P (which lies on the left edge of B), and I' is an optimal solution with $k-1$ points for the subset S_R of S that lies to the right of s_i. Since $d^k_{\text{sep}}(S) \leq w/(k+1)$, it follows that the separation of the new solution, which does have points on both sides of s_i, is at least as large as that of $d_{\text{sep}}(I)$. The same argument works for $i = 0$ and for $i = k$ (in the latter case we use the rightmost point p_R of S instead of p_L). As is easily seen, these observations also carry over to ε-approximations of the optimal solution.

Hence, for $i = 0$ and $i = k$, we invoke the procedure recursively for finding an ε-approximate solution with $k-1$ points for the set $S \setminus S_i$, and then add to the solution the leftmost point of S (for $i = 0$) or the rightmost point (for $i = k$).

Consider then the case $1 \leq i \leq k-1$. Put $S_L := \bigcup_{j<i} S_j$ and $S_R := \bigcup_{j>i} S_j$. We need to guess the number t of points of the optimal solution that lie in S_L (so that $k-t$ points lie in S_R). As argued above, we may assume that $1 \leq t \leq k-1$. For

each value of t in this range, we compute recursively an ε-approximation I_L (resp., I_R) of a maximally separated set of size t in S_L (resp., of size $k-t$ in S_R). Then $I_L \cup I_R$ form an ε-approximation of the optimal solution to the whole problem. Thus, for each strip we solve $2(k-1)$ problems with size smaller than k. In total, we need to solve $O(k^2)$ subproblems. Denoting by $T_k(n,\varepsilon)$ the maximum time needed to ε-approximate $d^k_{\text{sep}}(S)$, over sets S of n points, we thus obtain a procedure that handles the case $d^k_{\text{sep}}(S) \le w/(k+1)$, with total cost of $O(n + k^2 T_{k-1}(n,\varepsilon))$.

So we are left with the case in which the maximal separation distance $d^k_{\text{sep}}(S)$ is larger than $w/(k+1)$. We proceed in a manner similar to that for the case $k=2$. Let

$$\delta = \frac{\varepsilon w}{2\sqrt{2}(k+1)}.$$

We partition the bounding box B of the set S into $O(k^2/\varepsilon^2)$ grid cells of size at most $\delta \times \delta$, choose an arbitrary single point of S from each nonempty cell of the grid, obtain a set A of $O(k^2/\varepsilon^2)$ representative points, and compute an exact maximally separated set I of size k for A, using any, potentially brute-force, method. (One possibility is to use the algorithm presented in Section 5.)

We claim that $d_{\text{sep}}(I) \ge (1-\varepsilon) d^k_{\text{sep}}(S)$. Indeed, let $\{p_1, \ldots, p_k\} \subseteq S$ be a maximally separated set of S of size k. Since $\varepsilon < 1$, these points must lie in different cells. Let $p'_i \in A$ be the representative point from the cell in which p_i lies, and let $I' = \{p'_1, \ldots, p'_k\}$. As in the case $k=2$, it is easily seen that

$$d_{\text{sep}}(I') \ge d^k_{\text{sep}}(S) - 2\sqrt{2}\delta = d^k_{\text{sep}}(S) - \frac{\varepsilon w}{k+1} > (1-\varepsilon) d^k_{\text{sep}}(S),$$

because $d^k_{\text{sep}}(S) > w/(k+1)$. Since we solve the problem exactly for A, $d_{\text{sep}}(I) \ge d_{\text{sep}}(I') > (1-\varepsilon) d^k_{\text{sep}}(S)$, as asserted. The running time bound $T_k(n,\varepsilon)$ thus satisfies the recurrence

$$T_k(n,\varepsilon) = O(n + k^2 T_{k-1}(n,\varepsilon) + C_k(O(k^2/\varepsilon^2))),$$

where $C_k(m)$ is the time needed to compute exactly a maximally separated subset of size k in a set of m points. Clearly, the solution of this recurrence is $O(n)$, for any constant k. More precisely, it is upper bounded by $c^k (k!)^2 n + b(k)/\varepsilon^{a(k)}$, for some constant $c > 0$, where $b(k)$ is exponential in k, and $a(k)$ is at most linear in k. (For example, using a brute-force solution for the case of large separation, for which $C_k(m) = O(m^k)$, we obtain $b(k) \le (c'k)^{2k}$, for some constant $c' > 0$, and $a(k) = 2k$.) Thus, we have the following theorem.

THEOREM 2.1. *For a set S of n points in $\mathbb{R}^2$ and any constants k and $0 < \varepsilon < 1$, we can compute in $k^{O(k)} n + (k/\varepsilon)^{O(k)}$ time a subset $I \subseteq S$ of size k such that $d_{\text{sep}}(I) \ge (1-\varepsilon) d^k_{\text{sep}}(S)$.*

3. Computing a maximally separated triple. Let S be a set of n points in $\mathbb{R}^2$. We wish to compute a maximally separated triple in S. Our overall approach consists of three steps. First, we perform a binary search on the pairwise distances of S, and for each distance δ that the search encounters, we determine whether S contains a δ-separated triple. Next, in order to determine the existence of a δ-separated triple, we draw a sufficiently small grid within the bounding box of S so that each point of a δ-separated triple of S lies in a distinct grid cell. We thus reduce the problem of computing a δ-separated triple to a trichromatic variant of this problem. Finally, we determine the existence of a trichomatic δ-separated triple in $O(n^{4/3} \log^2 n)$ time.

For simplicity, we describe these steps in the reverse order. That is, we first describe the decision algorithm for the trichromatic version, then we show how to reduce the original decision problem to the trichromatic problem, and finally we sketch the binary-search procedure.

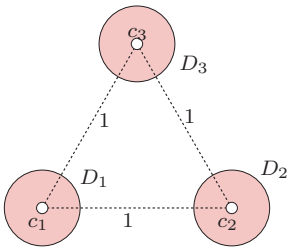


FIG. 1. *An instance of three point sets* (*contained in the shaded disks*) *with property* ($\triangle$).

We need a few notations. First, we may assume for the decision problem that $\delta = 1$. For a point $p \in \mathbb{R}^2$, let $\mathbb{D}(p)$ denote the disk of unit radius centered at p. For a set A of points in $\mathbb{R}^2$, let $K(A) = \bigcap_{p\in A} \mathbb{D}(p)$. $K(A)$ is a convex region bounded by circular arcs that lie on the boundaries of the disks $\mathbb{D}(p)$, and each disk contributes at most one such arc to $\partial K(A)$; $K(A)$ can be constructed in time $O(|A| \log |A|)$ [10].

3.1. Computing a trichromatic 1-separated triple. Let S_1, S_2, and S_3 be three sets of n points each in $\mathbb{R}^2$ that satisfy the following property.

($\triangle$) There is a constant $\delta \leq 1/6$ so that, for $i = 1, 2, 3$, S_i is contained in a disk D_i of radius δ centered at a point c_i, and $|c_1c_2| = |c_2c_3| = |c_3c_1| = 1$.

Without loss of generality, we assume that $c_1 = (0, 0), c_2 = (1, 0)$, and $c_3 = (1/2, \sqrt{3}/2)$; see Figure 1. We wish to compute a 1-separated triple in $S_1 \times S_2 \times S_3$, or to determine that no such triple exists. Clearly, no other triple of points in $S_1 \cup S_2 \cup S_3$ can be 1-separated.

Before continuing, we remark that, informally, property ($\triangle$) captures the hard case for finding a 1-separated trichromatic triple. If two of the sets S_i are too close to each other, then no trichromatic 1-separated triple exists, and if some pairs of sets are too far apart, the problem reduces to finding a diametral pair. This will be discussed in detail in section 3.2.

Let $G \subseteq S_1 \times S_2$ denote the bipartite graph

$$G = \{(p, q) \mid p \in S_1, q \in S_2;\ |pq| \geq 1\}.$$

Using the algorithm of Katz and Sharir [18], we compute, in $O(n^{4/3} \log n)$ time, a family $\mathcal{F} = \{A_1 \times B_1, \ldots, A_u \times B_u\}$, which is a partition of G into complete bipartite graphs, satisfying

$$\sum_i (|A_i| + |B_i|) = O(n^{4/3} \log n).$$

For each $1 \leq i \leq u$, let $R_i = K(A_i) \cup K(B_i)$. Set $\mathcal{R} := \bigcap_{i=1}^{u} R_i$. The following lemma is a straightforward reformulation of the original problem.

LEMMA 3.1. *There exists a* 1*-separated* (*trichromatic*) *triple in* $S_1 \times S_2 \times S_3$ *if and only if* $S_3 \not\subseteq \mathcal{R}$.

Proof. Let $p \in S_3$ be a point that does not lie in $\mathcal{R}$. Then there exists an $i \leq u$ so that $p \notin R_i = K(A_i) \cup K(B_i)$. Since $p \notin K(A_i)$, there is a point $q \in A_i$ so that

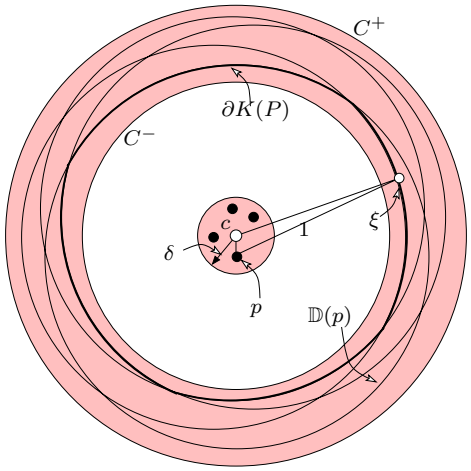


FIG. 2. *The annulus that contains* $\partial K(P)$ (*drawn as a thick curve*).

$p \notin \mathbb{D}(q)$. Similarly there is a point $r \in B_i$ so that $p \notin \mathbb{D}(r)$. Hence, $|pq|, |pr| \geq 1$. Moreover, $|qr| \geq 1$ because $(q, r) \in A_i \times B_i$, thereby implying that (q, r, p) is a 1-separated triple. The converse implication is established in a similar manner: Suppose that $q \in S_1$, $r \in S_2$, and $p \in S_3$ form a 1-separated triple. Since $|qr| \geq 1$, there exists an i such that $(q, r) \in A_i \times B_i$. Since $|pq|, |pr| \geq 1$, it follows that $p \notin K(A_i)$ and $p \notin K(B_i)$, and therefore $p \notin \mathcal{R}$. □

The following simple technical observation is important for our algorithm.

LEMMA 3.2. *Let P be a set of points in $\mathbb{R}^2$ lying in a disk of radius δ centered at a point c. Then $\partial K(P)$ lies between two concentric circles of radius $1 + \delta$ and $1 - \delta$ centered at c.*

Proof. Let C^+ (resp., C^-) denote the circle of radius $1 + \delta$ (resp., $1 - \delta$) centered at c. Fix a point $p \in P$. For any point $\xi \in \partial\mathbb{D}(p)$,

$$1 - \delta \leq |\xi p| - |pc| \leq |\xi c| \leq |\xi p| + |pc| \leq 1 + \delta.$$

Hence, $\partial\mathbb{D}(p)$ lies between C^- and C^+. Since this is true for every point $p \in P$, $\partial K(P)$ lies between C^- and C^+; see Figure 2. □

LEMMA 3.3. *For each $1 \leq i \leq u$, the upper (resp., lower) boundaries of $K(A_i)$ and $K(B_i)$ cross at exactly one point.*

Proof. Let W_1 (resp., W_2) denote the annulus bounded by the concentric circles of radii $1+\delta$ and $1-\delta$ centered at c_1 (resp., c_2). By Lemma 3.2, $\partial K(A_i)$ (resp., $\partial K(B_i)$) is contained in W_1 (resp., W_2). Therefore $\partial K(A_i) \cap \partial K(B_i) \subseteq W_1 \cap W_2$. Since $\delta < 1/6$ and $|c_1 c_2| = 1$, the inner circles of W_1 and W_2 intersect, and thus $W_1 \cap W_2$ consists of two connected components Σ^+, Σ^-, where Σ^+ lies above the x-axis and Σ^- below the x-axis; see Figure 3. An easy calculation shows that the x-coordinate of the leftmost (resp., rightmost) point of Σ^+ is $1/2 - 2\delta$ (resp., $1/2 + 2\delta$), and that the y-coordinate of the bottommost point is $\sqrt{(1-\delta)^2 - 1/4}$. Since $\delta \leq 1/6$, Σ^+ lies fully to the right of D_1, to the left of D_2, and above both these disks. This implies that, within Σ^+, the boundary of each $\mathbb{D}(p)$, for $p \in A_i$, is the graph of a strictly decreasing function, and thus $\partial K(A_i)$ is also the graph of a strictly decreasing function within Σ^+. By a fully symmetric argument, $\partial K(B_i)$ is the graph of a strictly increasing function within Σ^+. Moreover, $\partial K(A_i) \cap \Sigma^+$ is contained in the upper boundary of $K(A_i)$,

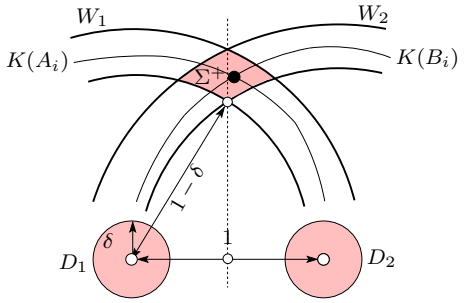


FIG. 3. *The annuli W_1, W_2 and their top intersection Σ^+. For $\delta \le 1/6$, the lowest point of Σ^+ lies above the line $y = \delta$, and its leftmost point has x-coordinate $\ge \delta$.*

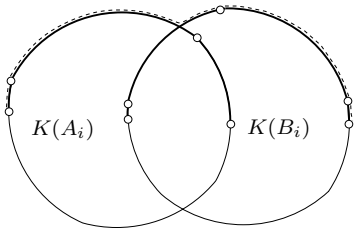


FIG. 4. *$K(A_i)$, $K(B_i)$ (whose top boundaries are drawn as thick curves), and the edges of Γ_i (drawn as dashed arcs).*

and similarly for $K(B_i)$, because Σ^+ lies above D_1 and D_2. This is easily seen to imply the assertion of the lemma. □

Lemma 3.3 implies that ∂R_i consists of a connected portion of $\partial K(A_i)$ and a connected portion of $\partial K(B_i)$. The leftmost and rightmost points of R_i partition ∂R_i into two parts, which we refer to as the upper and lower boundaries of R_i. Let Γ_i be the set of circular arcs forming the upper boundary of R_i; we have $|\Gamma_i| \le |A_i| + |B_i|$. See Figure 4. Set $\Gamma := \bigcup_{i=1}^{u} \Gamma_i$; then $|\Gamma| \le \sum_{i=1}^{u} (|A_i| + |B_i|)$. Let $\mathcal{L}_\Gamma$ denote the lower envelope of Γ.

LEMMA 3.4. *A point $p \in S_3$ lies inside $\mathcal{R}$ if and only if p lies below the lower envelope $\mathcal{L}_\Gamma$.*

Proof. If $p \in \mathcal{R}$, then it lies below the upper boundary of each R_i, thereby implying that p lies below $\mathcal{L}_\Gamma$. Conversely, suppose that p lies below $\mathcal{L}_\Gamma$. Then p lies below the upper boundary of every R_i. Let Σ^+ be the same region as in the proof of Lemma 3.3. Since $|c_1c_2| = |c_1c_3| = |c_2c_3| = 1$, and $\delta \le 1/6$, a simple calculation shows that $D_3 \subset \Sigma^+$, and thus S_3 is also contained in Σ^+. The argument in the proof of Lemma 3.3 implies that Σ^+ lies above the lower boundaries of every $K(A_i)$ and of every $K(B_i)$. Hence, if p lies below the boundary of each R_i, it lies in each R_i and thus also in $\mathcal{R}$. □

In view of Lemma 3.4, we may proceed as follows. For each i, we compute $K(A_i)$, $K(B_i)$, R_i, and Γ_i. The total time spent in this step is

$$O\left(\sum_{i=1}^{u} (|A_i| + |B_i|) \log n\right) = O\big(n^{4/3} \log^2 n\big).$$

Since each arc in Γ is a portion of the upper boundary of a unit-radius disk, two arcs of Γ intersect in at most one point. Hence, we can compute the lower envelope

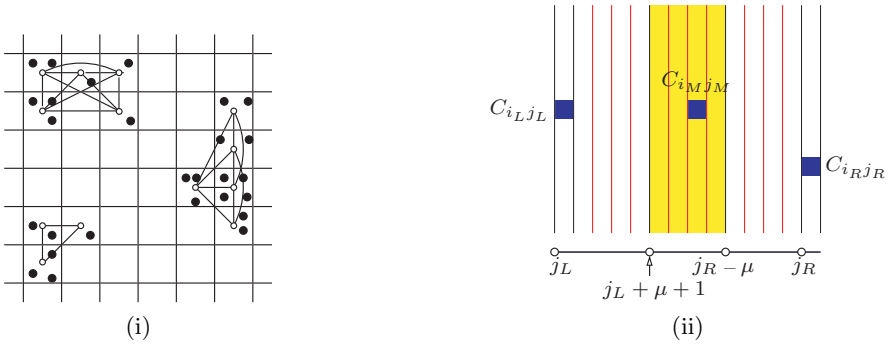


FIG. 5. (i) *Drawing a grid and the graph* $\mathcal{G}$. (ii) *The case where* $\mathcal{C}$ *spans more than* $3\mu + 1$ *columns.*

$\mathcal{L}_\Gamma$ of Γ in $O(|\Gamma| \log n)$ time, using the algorithm of Hershberger [16] (see also [21]). For each edge ξ of $\mathcal{L}_\Gamma$ we store the index j such that ξ is (a portion of) an arc in Γ_j. Finally, for each point $p \in S_3$ we determine whether p lies below or above $\mathcal{L}_\Gamma$, using a simple binary search over the arcs of $\mathcal{L}_\Gamma$. If p lies above $\mathcal{L}_\Gamma$, then the test yields an arc of $\mathcal{L}_\Gamma$ that lies below p. This arc is contained in an arc of some Γ_i, and we can thus deduce that $p \notin R_i$ (by Lemma 3.4). Then, scanning the points of $A_i \cup B_i$ in additional $O(|A_i| + |B_i|)$ time, we are certain to find a 1-separated triple $(a, b, p) \in A_i \times B_i \times S_3$. The total running time of the algorithm is $O(n^{4/3} \log^2 n)$. Hence, we obtain the following result.

THEOREM 3.5. *Let* S_1, S_2, *and* S_3 *be three finite point sets in* $\mathbb{R}^2$ *that satisfy property* $(\triangle)$, *and put* $n_i = |S_i|$, *for* $i = 1, 2, 3$. *Then one can construct, in time* $O(n^{4/3} \log^2 n)$, *a* 1*-separated triple in* $S_1 \times S_2 \times S_3$, *if one exists, or determine that no such triple exists.*

3.2. Reduction to the trichromatic case. Let S be a set of n points in $\mathbb{R}^2$. We wish to compute a 1-separated triple in S if one exists, or else to determine that no such triple exists. We fix a small constant $\varepsilon \ll 1/16$, and set $\mu = \lceil 1/\varepsilon \rceil$. We draw a square grid of size ε in the plane. For $i, j \in \mathbb{Z}$, let C_{ij} denote the grid cell $[i\varepsilon, (i+1)\varepsilon) \times [j\varepsilon, (j+1)\varepsilon)$, and let $S_{ij} = S \cap C_{ij}$. Let $\mathcal{C}$ denote the set of nonempty grid cells (i.e., those with $S_{ij} \neq \emptyset$). We construct a graph $\mathcal{G} = (\mathcal{C}, \mathcal{E})$ where $(C, C') \in \mathcal{E}$ if $\min\{|pp'| \mid p \in C, p \in C'\} < 1$; see Figure 5 (i).

LEMMA 3.6. *If* $\mathcal{G}$ *is not connected, then we can compute a* 1*-separated triple in* S (*or determine that no such triple exists*) *in* $O(n \log n)$ *time.*

Proof. First, note that if two nonempty grid cells $C_{ij}, C_{kl} \in \mathcal{C}$ lie in different connected components of $\mathcal{G}$, then for any pair $(p, q) \in S_{ij} \times S_{kl}$, $|pq| \geq 1$. If $\mathcal{G}$ has three (or more) connected components $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$, then a 1-separated triple is obtained by choosing one point of S lying in a single grid cell of each of $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$. If $\mathcal{G}$ has at least two connected components, then let $S_1 \subseteq S$ be the subset of points lying in the grid cells of one connected component, and put $S_2 := S \setminus S_1$. We test, in $O(n \log n)$ time, whether $\max\{\mathrm{diam}(S_1), \mathrm{diam}(S_2)\} \geq 1$. Suppose that p, q is a diametral pair of, say, S_1 and that $|pq| \geq 1$; then we choose an arbitrary point $r \in S_2$ and return (p, q, r). By construction, this is a 1-separated triple. If $\mathrm{diam}(S_1), \mathrm{diam}(S_2)$ are both smaller than 1, then clearly no 1-separated triple exists. Hence, if $\mathcal{G}$ is not connected, then we can construct in time $O(n \log n)$ a 1-separated triple in S if one exists, or determine that no such triple exists. ∎

LEMMA 3.7. *If $\mathcal{G}$ is connected and $\mathcal{C}$ spans more than $3\mu+1$ columns or rows of the grid, i.e., it has cells in two columns (or rows) whose indices j_L, j_R satisfy $j_R - j_L \geq 3\mu+1$, then a 1-separated triple in S exists, and can be constructed in $O(n)$ time.*

Proof. Without loss of generality, it suffices to consider the case where $\mathcal{C}$ spans more than $3\mu+1$ columns. Let $C_{i_L j_L}$ (resp., $C_{i_R j_R}$) be a grid cell of $\mathcal{C}$ in the leftmost (resp., rightmost) column, let $p_L \in S_{i_L j_L}$, and let $p_R \in S_{i_R j_R}$. By assumption, $j_R - j_L \geq 3\mu+1$; see Figure 5 (ii). We group the columns between the j_Lth and j_Rth columns (exclusive) into three pairwise-disjoint vertical strips V_1, V_2, V_3, appearing in this left-to-right order, each of width at least $\mu\varepsilon \geq 1$. It is easily seen that V_2 must contain a point of S, or else $\mathcal{G}$ would not be connected. Then p_L, p_R, and any point in $V_2 \cap S$ form a 1-separated triple. Clearly, finding these points takes linear time. ∎

By Lemmas 3.6 and 3.7, it remains to consider the case in which $\mathcal{G}$ is connected and $\mathcal{C}$ spans at most $3\mu+1$ rows and at most $3\mu+1$ columns. Clearly, in this case $|\mathcal{C}| \leq (3\mu+1)^2$. We consider all triples $C_1, C_2, C_3 \in \mathcal{C}$ and determine whether $S_1 \times S_2 \times S_3$ contains a 1-separated triple, where $S_i = C_i \cap S$, for $i = 1, 2, 3$. If the maximum distance between two of these three cells, say, C_1 and C_2, is less than 1, then no 1-separated triple in $S_1 \times S_2 \times S_3$ exists. Hence, we can assume that the maximum distance between every pair of C_1, C_2, C_3 is at least 1. There are four cases to consider, depending on the number k of edges of $\mathcal{G}$ between these three cells:

(i) $k = 0$; that is, C_1, C_2, C_3 is an independent set in $\mathcal{G}$. Then any triple in $S_1 \times S_2 \times S_3$ is 1-separated, and we return one of these triples.

(ii) $k = 1$; suppose, without loss of generality, that $(C_1, C_2) \in \mathcal{E}$ and $(C_1, C_3), (C_2, C_3) \notin \mathcal{E}$. We compute a diametral pair (p, q) of $S_1 \cup S_2$. If $|pq| \geq 1$, then we return (p, q, r), where r is any point of S_3. If $|pq| < 1$, no triple in $S_1 \times S_2 \times S_3$ is 1-separated. This step takes $O(n \log n)$ time.

(iii) $k = 2$; suppose, without loss of generality, that $(C_1, C_2), (C_1, C_3) \in \mathcal{E}$ and $(C_2, C_3) \notin \mathcal{E}$. We compute $K(S_2)$ and $K(S_3)$. If a point $p \in S_1$ lies neither in $K(S_2)$ nor in $K(S_3)$, then there exists a pair $(q, r) \in S_2 \times S_3$ so that $p \notin \mathbb{D}(q) \cup \mathbb{D}(r)$ and thus (p, q, r) is 1-separated. If $S_1 \subseteq K(S_2) \cup K(S_3)$, then, arguing as in the proof of Lemma 3.1, no triple in $S_1 \times S_2 \times S_3$ is 1-separated. This step too takes $O(n \log n)$ time.

(iv) $k = 3$; that is, $(C_1, C_2), (C_1, C_3), (C_2, C_3) \in \mathcal{E}$. In other words, for any pair $i \neq j \in \{1, 2, 3\}$ we have

$$\min\{|xy| \mid x \in C_i, y \in C_j\} < 1 \leq \max\{|xy| \mid x \in C_i, y \in C_j\}.$$

By the triangle inequality, this implies that any $x \in C_i$, $y \in C_j$ satisfy $1 - 2\sqrt{2}\varepsilon \leq |xy| \leq 1 + 2\sqrt{2}\varepsilon$. We claim that our choice of ε implies that there exist points $c_1, c_2, c_3 \in \mathbb{R}^2$ so that $|c_i c_j| = 1$ for each pair of distinct points c_i, c_j, and S_i is contained in the disk D_i of radius $\delta \leq 1/6$ centered at c_i, for $i = 1, 2, 3$. To see this, pick any pair of points $c_1 \in C_1$, $c_2 \in C_2$, such that $|c_1 c_2| = 1$. Let $c_3 \in \mathbb{R}^2$ be a point such that $\Delta c_1 c_2 c_3$ is equilateral and c_3 lies on the same side of the line through c_1 and c_2 as C_3 (our choice of ε is easily seen to imply that C_3 does not intersect such a line). By what we have just argued, C_3 is fully contained in the intersection of the two annuli

$$1 - 2\sqrt{2}\varepsilon \leq |c_1 x| \leq 1 + 2\sqrt{2}\varepsilon,$$
$$1 - 2\sqrt{2}\varepsilon \leq |c_2 x| \leq 1 + 2\sqrt{2}\varepsilon.$$

A simple calculation then shows that C_3 is fully contained in the disk of radius $1/6$ centered at c_3. In other words, in this case S_1, S_2, and S_3 satisfy property ($\triangle$), and we can therefore use Theorem 3.5 to compute a 1-separated triple in $S_1 \times S_2 \times S_3$, if one exists, or to determine that no such triple exists.

The total running time of the algorithm is dominated by the overall cost of handling case (iv), and is thus, by Theorem 3.5, $O(n^{4/3}\log^2 n)$ since $\mu = O(1)$. We thus obtain the following main result of this section.

THEOREM 3.8. *Let S be a set of n points in $\mathbb{R}^2$. We can compute, in $O(n^{4/3}\log^2 n)$ time, a 1-separated triple in S, if one exists, or determine that no such triple exists.*

Finally, we run a binary search on the $\binom{n}{2}$ pairwise distances in S. The kth smallest pairwise distance δ_k in S, for any $1 \le k \le \binom{n}{2}$, can be computed in time $O(n^{4/3}\log^2 n)$ [18], and by Theorem 3.8, we can determine whether a δ_k-separated triple exists in S within the same time bound. Hence, we obtain the following theorem.

THEOREM 3.9. *Let S be a set of n points in $\mathbb{R}^2$. We can compute, in $O(n^{4/3}\log^3 n)$ time, a maximally separated triple in S.*

4. Computing a maximally separated quadruple. Our overall approach to this problem is similar to the one in Section 3. We first consider a multicolored version of this problem, in which we are given four sets, S_1, S_2, S_3, and S_4, of points, placed "reasonably far" from each other, and we wish to determine whether there exists a 1-separated quadruple in $S_1 \times S_2 \times S_3 \times S_4$. The easy cases are when some pairs of subsets S_i, S_j are either too far from each other or too near each other. The difficult case is when these sets are arranged in a so-called *diamond* configuration, and we present in Section 4.1 below an algorithm for handling this case. We then present the overall algorithm, which, as in the previous section, runs a binary search through the pairwise distances in S, and, for each fixed distance, reduces the general problem to a constant number of multicolored instances.

4.1. The diamond configuration. Let S_1, S_2, S_3, and S_4 be sets of n points each in $\mathbb{R}^2$ that satisfy the following property:

($\diamond$) There is a constant $\delta \le 1/8$ so that each S_i, for $i = 1, \ldots, 4$, is contained in a disk D_i of radius δ centered at a point c_i, so that $|c_1c_3|, |c_2c_3|, |c_1c_4|, |c_2c_4| = 1$, and $1 \le |c_1c_2| \le 1 + 2\delta < |c_3c_4|$.

Without loss of generality assume that $c_3 = (0,0)$, c_4 lies on the x-axis to the right of c_3, and c_1 (resp., c_2) lies below (resp., above) the x-axis (in symmetric positions). The conditions on the c_i's imply that

$$1 \le |c_1c_2| \le 1 + 2\delta < \sqrt{2} \le |c_3c_4| \le \sqrt{3}.$$

See Figure 6.

Note that one helpful property of the diamond configuration is that any pair of points in $S_3 \times S_4$ is 1-separated, so only five of the six pairwise distances in a quadruple in $S_1 \times S_2 \times S_3 \times S_4$ need to be considered.

For a point $p \in S_1$, let $S_p^{(3)} = \{q \in S_3 \mid |pq| \ge 1\}$ and $S_p^{(4)} = \{q \in S_4 \mid |pq| \ge 1\}$. (We ignore for the time being the issue of efficient construction of these sets; this will be addressed later on.) We remove from S_1 any point p for which one of these sets is empty, because such a p cannot be part of a 1-separated quadruple in $S_1 \times S_2 \times S_3 \times S_4$.

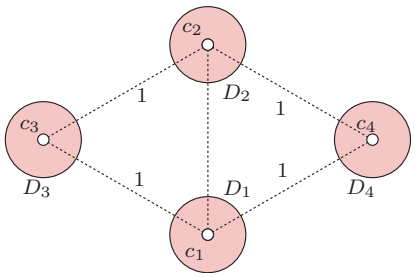


FIG. 6. *A diamond configuration. The missing edge between D_3 and D_4 indicates that every pair of points in $S_3 \times S_4$ is 1-separated.*

Set, for each remaining $p \in S_1$,

$$K_p^{(3)} := \bigcap_{q \in S_p^{(3)}} \mathbb{D}(q), \qquad K_p^{(4)} := \bigcap_{q \in S_p^{(4)}} \mathbb{D}(q),$$

$$R_p := K_p^{(3)} \cup K_p^{(4)} \cup \mathbb{D}(p), \qquad \mathcal{R} := \bigcap_{p \in S_1} R_p.$$

The following lemma is fairly straightforward (cf. Lemma 3.1).

LEMMA 4.1. *There exists a 1-separated quadruple in $S_1 \times S_2 \times S_3 \times S_4$ if and only if $S_2 \not\subseteq \mathcal{R}$.*

Proof. If there exists a point $q \in S_2 \setminus \mathcal{R}$, then there exists a point $p \in S_1$ so that $q \notin R_p = K_p^{(3)} \cup K_p^{(4)} \cup \mathbb{D}(p)$, which implies that there is a pair $(u, v) \in S_p^{(3)} \times S_p^{(4)}$ so that $q \notin \mathbb{D}(u) \cup \mathbb{D}(v) \cup \mathbb{D}(p)$. Therefore, $|pu|, |qu|, |pv|, |qv|, |pq| \geq 1$. By property $(\diamond)$, $|uv| \geq 1$ too. Hence, (p, q, u, v) is 1-separated. The converse implication is argued in the same manner. ∎

The following is a variant of Lemma 3.3, proved in a similar manner.

LEMMA 4.2.

(i) *For any $p \in S_1$, $\partial K_p^{(3)}$ and $\partial K_p^{(4)}$ intersect above the x-axis at exactly one point σ_p, which lies on the upper boundaries of both regions.*

(ii) *For any $p \in S_1$, $\partial K_p^{(3)}$ and $\partial \mathbb{D}(p)$ intersect above the x-axis exactly once, and similarly for $\partial K_p^{(4)}$ and $\partial \mathbb{D}(p)$.*

Proof. (i) Let W_3 (resp., W_4) denote the annulus bounded by the concentric circles of radii $1 + \delta$ and $1 - \delta$ centered at c_3 (resp., c_4). By Lemma 3.2, $\partial K_p^{(3)}$ (resp., $\partial K_p^{(4)}$) is contained in W_3 (resp., W_4). Therefore $\partial K_p^{(3)} \cap \partial K_p^{(4)} \subseteq W_3 \cap W_4$. Since $\delta < 1 - \sqrt{3}/2$ and $|c_3 c_4| \leq \sqrt{3}$, the inner circles of W_3 and W_4 intersect and thus $W_3 \cap W_4$ consists of two connected components Σ^+, Σ^-, where Σ^+ lies above the x-axis and Σ^- below the x-axis (as in Figure 3(i)). Moreover, by the choice of δ, Σ^+ lies fully to the right of D_3, to the left of D_4, and above both these disks, as is easily verified. This implies that, within Σ^+, the boundary of each $\mathbb{D}(q)$, for $q \in S_p^{(3)}$, is the graph of a decreasing function, and thus $\partial K_p^{(3)}$ is also the graph of a decreasing function within Σ^+. By a fully symmetric argument, $\partial K_p^{(4)}$ is the graph of an increasing function within Σ^+. Moreover, $\partial K_p^{(3)} \cap \Sigma^+$ is contained in the upper boundary of $K_p^{(3)}$, and similarly for $K_p^{(4)}$, because Σ^+ lies above D_3 and D_4. This is easily seen to imply the assertion in (i).

(ii) The center p of $\mathbb{D}(p)$ lies to the right of the center of the circle of any arc that appears on $\partial K_p^{(3)}$, and only the upper semicircle of $\mathbb{D}(p)$ can be above the x-axis.

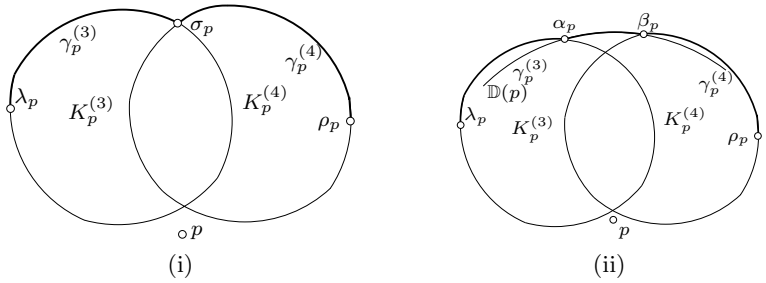


FIG. 7. (i) *Implicit representation of the upper boundary of R_p when $\mathbb{D}(p)$ does not appear on this boundary.* (ii) *Implicit representation of the upper boundary of R_p when $\mathbb{D}(p)$ appears on it. In both cases, the boundary is drawn as a thick curve.*

Hence, above the x-axis, $\partial\mathbb{D}(p)$ intersects any arc γ that bounds $\partial K_p^{(3)}$ in at most one point, and lies above γ to the right of that point. Hence $\partial\mathbb{D}(p)$ lies above $\partial K_p^{(3)}$ to the right of any intersection point between the two curves, which readily implies (ii). □

DEFINITION 4.3. *For a point $p \in S_1$, let σ_p denote, as in Lemma* 4.2*, the unique intersection point of the upper boundaries of $K_p^{(3)}$ and $K_p^{(4)}$, and let α_p (resp., β_p) denote the intersection point of the upper boundary of $K_p^{(3)}$ (resp., $K_p^{(4)}$) with $\mathbb{D}(p)$ if such a point exists.*

Consider the upper envelope of the upper boundaries of $K_p^{(3)}$, $K_p^{(4)}$, and $\mathbb{D}(p)$. The preceding analysis implies that the envelope has one of the following two structures: (a) Either $\mathbb{D}(p)$ does not appear on the envelope, and then the envelope consists of a connected portion $\gamma_p^{(3)}$ of the upper boundary of $K_p^{(3)}$ and a connected portion $\gamma_p^{(4)}$ of the upper boundary of $K_p^{(4)}$, meeting at the point σ_p (see Figure 7 (i)); or (b) $\mathbb{D}(p)$ appears on the envelope, and then the envelope consists of a connected portion $\gamma_p^{(3)}$ of the upper boundary of $K_p^{(3)}$, a connected portion δ_p of the upper boundary of $\mathbb{D}(p)$, and a connected portion $\gamma_p^{(4)}$ of the upper boundary of $K_p^{(4)}$, so that the first and second portions meet at α_p and the second and third portions meet at β_p (see Figure 7 (ii)).

Let $\Gamma^{(3)} = \{\gamma_p^{(3)} \mid p \in S_1\}$, $\Gamma^{(4)} = \{\gamma_p^{(4)} \mid p \in S_1\}$, and $\Delta = \{\delta_p \mid p \in S_1\}$. Note the difference between this notation and the one in section 3. There Γ_i was a family of circular arcs, whereas here each arc in $\Gamma^{(3)}$ or $\Gamma^{(4)}$ is a sequence of circular arcs. Let $\mathcal{L}^{(3)}$ (resp., $\mathcal{L}^{(4)}$, $\mathcal{L}^{(1)}$) denote the lower envelope of $\Gamma^{(3)}$ (resp., $\Gamma^{(4)}$, Δ).

The following corollary follows from Lemmas 3.4 and 4.2. Its proof uses the obvious observation that the lower envelope of $\mathcal{L}^{(3)}$, $\mathcal{L}^{(4)}$, and $\mathcal{L}^{(1)}$ is the same as the lower envelope of the upper boundaries of the regions R_p, for $p \in S_1$. (We follow here the convention that if a curve is undefined at some x, it is assumed to be $+\infty$ there.) Arguing in much the same way as in Section 3 and exploiting the fact that $\delta < 1/8$, one can show that a point of S_2 does not lie below the lower boundary of any $K_p^{(3)}, K_p^{(4)}$, or $\mathbb{D}(p)$. Hence, we obtain the following corollary.

COROLLARY 4.4. *A point $q \in S_2$ lies in $\mathcal{R}$ if and only if q lies below each of $\mathcal{L}^{(3)}$, $\mathcal{L}^{(4)}$, and $\mathcal{L}^{(1)}$.*

We thus compute each of the envelopes $\mathcal{L}^{(3)}$, $\mathcal{L}^{(4)}$, and $\mathcal{L}^{(1)}$ separately, and determine whether any point of S_2 lies above any of them. If the answer is yes, we can conclude that a 1-separated quadruple in $S_1 \times S_2 \times S_3 \times S_4$ exists, and we can compute

it in additional linear time. Otherwise no such quadruple exists.

However, unlike the situation in Section 3, computing these envelopes explicitly is expensive, because they consist of too many arcs, so we represent them implicitly. We first describe the implicit representation of the envelopes and of their arcs, following a similar representation used by Agarwal, Sharir, and Welzl [4], and then present the algorithm for computing and searching in the envelopes.

Implicit representation of $K_p^{(3)}, K_p^{(4)}$, and of the lower envelopes. For a subset $Q \subseteq S_1$, let $\mathcal{L}_Q^{(3)}$ denote the lower envelope of arcs in the set $\{\gamma_p^{(3)} \mid p \in Q\}$. We represent $\mathcal{L}_Q^{(3)}$ by the sequence of its *breakpoints* in increasing order of their x-coordinates. The breakpoints are defined so that each portion ξ of $\mathcal{L}_Q^{(3)}$ between two consecutive breakpoints is contained in a single $\gamma_p^{(3)}$ (such a ξ may overlap with many $\gamma_p^{(3)}$'s, but there is (at least) one point $p \in S_1$ such that ξ is fully contained in $\gamma_p^{(3)}$). We maintain ξ implicitly, by recording a point $p \in Q$ that satisfies $\xi \subseteq \gamma_p^{(3)}$. Recall that each $\gamma_p^{(3)}$ may consist of many circular arcs, bounding different disks centered at points of $S_p^{(3)}$; our implicit representation avoids the costly explicit enumeration of these arcs.

Let $\mathcal{D}_3 = \{\mathbb{D}(q) \mid q \in S_3\}$. To represent each $\gamma_p^{(3)}$ implicitly, we choose a parameter $n \le s \le n^2$, and use the result of Katz and Sharir [18], which shows that there exists a family $\mathcal{F}^{(3)} = \{\mathcal{D}^{(1)}, \ldots, \mathcal{D}^{(u)}\}$ of *canonical* subsets of $\mathcal{D}$, with a corresponding family of intersection regions $\mathcal{I}^{(i)} = \bigcap \mathcal{D}^{(i)}$, for $i = 1, \ldots, u$, such that $\sum_{i=1}^{u} |\mathcal{D}^{(i)}| = O(s \log n)$, and such that for any $p \in S_1$, $K_p^{(3)}$ can be represented as the intersection of $O((n/\sqrt{s}) \log n)$ of these canonical regions $\mathcal{I}^{(i)}$. Let $J_p^{(3)}$ denote the set of indices of these canonical regions, i.e., $K_p^{(3)} = \bigcap_{j \in J_p^{(3)}} \mathcal{I}^{(j)}$. We represent each arc $\gamma_p^{(3)}$ by its endpoints and by the set $J_p^{(3)}$. We also store the vertices of all $\mathcal{I}^{(j)}$ in a single master list Λ, sorted in increasing order of their x-coordinates.

As shown in [18], $\mathcal{F}^{(3)}$ and the sets $J_p^{(3)}$ for all $p \in S_3$ can be computed in time $O((s + n^2/\sqrt{s}) \log n)$, and we spend another $O(s \log^2 n)$ time to compute the regions $\mathcal{I}^{(j)}$. A similar representation was developed by Agarwal, Sharir, and Welzl [4], who showed that the above representation enables us to perform each of the following four operations on $\{\partial K_p^{(3)}\}_{p \in S_1}$ in $O((n/\sqrt{s}) \log^3 n)$ time.

(S1) *Leftmost and rightmost points:* Given a point $p \in S_1$, compute the leftmost and rightmost points of $K_p^{(3)}$.

(S2) *Intersection point(s) with a vertical line:* Given a vertical line ℓ and a point $p \in S_1$, determine the intersection point(s) of ℓ with $\partial K_p^{(3)}$.

(S3) *Intersection points with a unit disk:* Given a unit disk $\mathbb{D}$ and a point $p \in S_1$, determine the intersection point(s) of $\mathbb{D}$ with $\partial K_p^{(3)}$.

(S4) *Crossing point of two arcs:* Given two points $p, q \in S_1$ and an x-interval $[a, b]$ contained in the x-span of the top boundaries of $K_p^{(3)}$ and $K_q^{(3)}$, determine whether they cross in $[a, b]$. If so, return their crossing point. If they *weakly cross* in $[a, b]$, i.e., overlap over some subinterval J of $[a, b]$ and their vertical order to the right of J is the reverse of their vertical order to the left of J, then return the leftmost endpoint of their common overlap in $[a, b]$.

In a fully analogous fashion, we process S_4 in $O((s + (n^2/\sqrt{s})) \log n)$ time, to compute an implicit representation of all the arcs $\gamma_p^{(4)}$. Each of the operations (S1)–(S4) on $\{\partial K_p^{(4)}\}_{p \in S_1}$ can also be performed in $O((n/\sqrt{s}) \log^3 n)$ time. Moreover, given any $p \in S_1$, the intersection point of the top boundaries of $K_p^{(3)}$ and $K_p^{(4)}$ can also be

computed in $O((n/\sqrt{s})\log^3 n)$ time.

Computing $\mathcal{L}^{(3)}$, $\mathcal{L}^{(4)}$, and $\mathcal{L}^{(1)}$. Using the subroutine (S1), we first compute the lefmost point λ_p of $K_p^{(3)}$ and the rightmost point ρ_p of $K_p^{(4)}$, for each $p \in S_1$. Next, using (S3) and (S4), we compute the intersection point σ_p of the upper boundaries of $K_p^{(3)}$ and $K_p^{(4)}$, the intersection point α_p of the upper boundaries of $K_p^{(3)}$ and $\mathbb{D}(p)$, and the intersection point β_p of the upper boundaries of $K_p^{(4)}$ and $\mathbb{D}(p)$. By comparing the x-coordinates of these three points, we can determine whether the upper boundary of R_p is of the first kind (disjoint from $\mathbb{D}(p)$) or of the second kind (overlapping an arc of $\mathbb{D}(p)$). In the first case, $\gamma_p^{(3)}$ (resp., $\gamma_p^{(4)}$) is the portion of the upper boundary of $K_p^{(3)}$ (resp., $K_p^{(4)}$) between λ_p and σ_p (resp., σ_p and ρ_p). In the second case it is the portion of $\partial K_p^{(3)}$ (resp., $\partial K_p^{(4)}$) between λ_p and α_p (resp., β_p and ρ_p). We thus have the endpoints of γ_p and its implicit representation at our disposal.

The x-coordinate of any point $p \in S_1$ is at most $\sqrt{3}/2 + \delta$ and the x-coordinate of the rightmost point of R_p is at least $1 - \delta$. This easily implies that p lies below $K_p^{(3)}$ (i.e., the vertical ray emanating upward from p intersects $K_p^{(3)}$). Since this holds for any point $p \in S_1$, Theorem 2.8 of Agarwal, Sharir, and Welzl [4] (concerning the "pseudo-segment" property of the upper portions of $\partial K_p^{(3)}$) implies that, for any $p, q \in S_1$, $\gamma_p^{(3)}$ and $\gamma_q^{(3)}$ cross in at most one point. A similar argument proves the corresponding claim for the curves in $\Gamma^{(4)}$. Hence, each of $\Gamma^{(3)}, \Gamma^{(4)}$ is a collection of pseudo-segments. We can therefore compute the lower envelopes $\mathcal{L}^{(3)}, \mathcal{L}^{(4)}$ using the divide-and-conquer algorithm of Hershberger [16], mentioned above. In the main step of this algorithm, we have envelopes $\mathcal{L}_A^{(3)}, \mathcal{L}_B^{(3)}$ of two subsets $A, B \subseteq \Gamma^{(3)}$ at our disposal, and we need to merge these envelopes to compute $\mathcal{L}_{A\cup B}^{(3)}$. The only nontrivial part in the merge step is computing the crossing point of two arcs $\gamma_p^{(3)}$ and $\gamma_q^{(3)}$ in a given x-interval $[a, b]$ that is contained in the x-span of both $\gamma_p^{(3)}$ and $\gamma_q^{(3)}$. Using the subroutine (S4), we can compute, in $O((n/\sqrt{s})\log^3 n)$ time, the crossing point of the upper boundaries of $K_p^{(3)}$ and $K_p^{(4)}$ in the interval $[a, b]$, if it exists. If so, this is the crossing point of $\gamma_p^{(3)}$ and $\gamma_q^{(3)}$; otherwise, these arcs do not intersect over $[a, b]$. Plugging this bound into Hershberger's algorithm, we can compute an implicit representation of $\mathcal{L}^{(3)}$ in overall time $O((n^2/\sqrt{s})\log^4 n + s\log^2 n)$. Similarly, we can compute an implicit representation of $\mathcal{L}^{(4)}$ within the same time bound $O((n^2/\sqrt{s})\log^4 n + s\log^2 n)$. Computing $\mathcal{L}^{(1)}$ is easier, since no implicit representation is needed here: We simply have to compute the lower envelope of at most n upper unit circular arcs that behave as pseudo-segments, so their envelope can be computed in $O(n \log n)$ time (as in [16]). Finally, for each point $q \in S_2$, we determine in $O((n/\sqrt{s})\log^3 n)$ time, using the subroutine (S2), whether q lies above $\mathcal{L}^{(3)}$ or above $\mathcal{L}^{(4)}$. Testing whether q lies above $\mathcal{L}^{(1)}$ is easy to accomplish in $O(\log n)$ time. The total time spent is thus $O\left((n^2/\sqrt{s})\log^4 n + s\log^2 n\right)$. By choosing $s = n^{4/3}\log^{4/3} n$, we obtain the following summary result.

THEOREM 4.5. *Let S_1, S_2, S_3, and S_4 be four sets of n points each in $\mathbb{R}^2$ that satisfy property ($\diamond$). One can determine, in time $O(n^{4/3}\log^{10/3} n)$, whether $S_1 \times S_2 \times S_3 \times S_4$ contains a 1-separated quadruple, and, if so, compute such a quadruple.*

4.2. Reduction to the multicolored case. As in section 3, we construct a square grid of size ε, for a sufficiently small constant parameter $\varepsilon > 0$. Let $C_{ij}, S_{ij}, \mathcal{C}, \mathcal{G}$, and μ be as in section 3.

LEMMA 4.6. *If $\mathcal{G}$ is not connected, then we can compute a 1-separated quadruple in S (or determine that no such quadruple exists) in $O(n^{4/3}\log^2 n)$ time.*

Proof. If $\mathcal{G}$ has at least two connected components, then let $S_1 \subseteq S$ be the subset of points lying in the grid cells of one connected component, and put $S_2 := S \setminus S_1$. If a 1-separated quadruple exists, then there also exists a 1-separated quadruple that has points in both S_1 and S_2. Indeed, if (p_1, p_2, p_3, p_4) is a 1-separated quadruple that is contained in, say, S_1, then the quadruple obtained by replacing, say, p_1 by any point of S_2 is also 1-separated. Hence it suffices to look for 1-separated quadruples that have two points in each of S_1, S_2, or have three points in one of these sets and one point in the other set. Moreover, it suffices to find the two parts of such a quadruple *independently*—putting together any pair of such parts, one contained in S_1, the other in S_2, and consisting together of four points, will form a 1-separated quadruple in S. In the former case, it suffices to check that $\min\{\mathrm{diam}(S_1), \mathrm{diam}(S_2)\} \geq 1$, and then return a pair of diametral points in each of S_1, S_2. In the latter case, we apply the decision procedure of Section 3 to S_1 and to S_2. If either of these applications yields a 1-separated triple, combining it with any point in the other set yields a 1-separated quadruple in S. If none of these steps succeeds, S has no 1-separated quadruple. The overall cost of the procedure just sketched is, by Theorem 3.8, $O(n^{4/3}\log^2 n)$. □

LEMMA 4.7. *If $\mathcal{G}$ is connected and $\mathcal{C}$ spans more than $5\mu+1$ columns or rows of the grid, i.e., it has cells in two columns (or rows) whose indices j, j' satisfy $j'-j \geq 5\mu+1$, then a 1-separated quadruple in S exists, and can be constructed in $O(n)$ time.*

Proof. Consider the case where $\mathcal{C}$ spans more than $5\mu + 1$ columns. Let $C_{i_L j_L}$ (resp., $C_{i_R j_R}$) be a grid cell of $\mathcal{C}$ in the leftmost (resp., rightmost) column, and let $p_L \in S_{i_L j_L}$, and $p_R \in S_{i_R j_R}$. By assumption, $j_R - j_L \geq 5\mu + 1$. We group the columns between the j_Lth and j_Rth columns (exclusive) into five pairwise-disjoint vertical strips $V_1, \ldots, V_5$, appearing in this left-to-right order, each of width at least $\mu\varepsilon \geq 1$. We argue that each of V_2, V_4 must contain points of S, or else $\mathcal{G}$ would not be connected. Then p_L, p_R, any point in $V_2 \cap S$, and any point in $V_4 \cap S$ form a 1-separated quadruple. Clearly, finding these points takes linear time. □

By Lemmas 4.6 and 4.7, we may therefore assume that $\mathcal{G}$ is connected and that $\mathcal{C}$ spans at most $5\mu + 1$ rows and at most $5\mu + 1$ columns. In this case, we try all quadruples $C_1, C_2, C_3, C_4 \in \mathcal{C}$ and determine whether the corresponding product $S_1 \times S_2 \times S_3 \times S_4$ contains a 1-separated quadruple. We can assume that, for each pair of cells, the maximum distance between points in these two cells is at least one, and that the subgraph induced by these four cells is connected, because if the former assumption is violated, then no 1-separated quadruple exists, and if the latter is violated, then we can find a 1-separated quadruple (or determine that none exists), proceeding as in Lemma 4.6. In other words, we may assume that, for each C_i, C_j,

$$1 \leq \max_{x \in C_i,\, y \in C_j} |xy| \leq 1 + 2\sqrt{2}\varepsilon.$$

We now proceed by case analysis, according to the structure of the edges of $\mathcal{G}$ that connect the cells $C_1, \ldots, C_4$. Figure 8 shows all the possible cases, up to symmetries. It is easily checked that the complete graph on $C_1, \ldots, C_4$ is impossible, if ε is chosen sufficiently small.

In cases (i), (ii), and (iii), there is at least one node that has degree 1 in $\mathcal{G}$. It is then easy to reduce the problem to the case of finding a 1-separated triple. Consider for example case (iii), where the only edge incident to C_2 is (C_2, C_3). We then replace

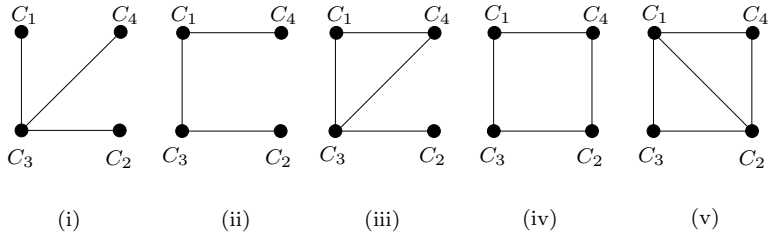


FIG. 8. *Possible graphs for $k = 4$.*

S_3 by the set

$$S_3' = \Big\{ p \in S_3 \mid \max_{q \in S_2} |pq| \geq 1 \Big\}.$$

S_3' can be computed in time $O(n \log n)$, by constructing $K(S_2)$ and choosing all points of S_3 that lie outside $K(S_2)$. We now find, in time $O(n^{4/3} \log^2 n)$, a 1-separated triple in $S_1 \times S_2 \times S_3'$, or determine that none exists. Once such a triple (p, q, r) is found, any point $s \in S_2$ for which $|rs| \geq 1$ can be added to it to form a 1-separated quadruple; s can be found in additional $O(n)$ time.

This leaves us with cases (iv) and (v). Pick points $c_i \in C_i$, for $i = 1, 2, 3, 4$, such that $|c_1c_3| = |c_2c_4| = 1$. Then

$$1 - 2\sqrt{2}\varepsilon \leq |c_2c_3|, |c_1c_4| \leq 1 + 2\sqrt{2}\varepsilon.$$

Hence, by translating c_2c_4 by distance $\leq 2\sqrt{2}\varepsilon$, we can also enforce $|c_2c_3| = 1$, and $1-4\sqrt{2}\varepsilon \leq |c_1c_4| \leq 1+4\sqrt{2}\varepsilon$. Note that $\angle c_1c_3c_2$ and $\angle c_3c_2c_4$ cannot be much smaller than $\pi/3$ each, because otherwise c_1 and c_2, or c_3 and c_4, would be too close to each other, contrary to what we are assuming. For the same reason, these angles cannot be much larger than $2\pi/3$.

This is easily seen to imply that, by slightly rotating c_4 around c_2, we can make the distance $|c_1c_4|$ also equal to 1. The new points c_i are no longer necessarily inside the respective cells C_i, for $i = 2, 3, 4$, but they remain close to these cells. If ε is chosen sufficiently small, the disk of radius $\delta = 1/8$ around c_i will fully contain C_i, for $i = 1, \ldots, 4$. Moreover, by slightly flexing the rhombus $c_1c_3c_2c_4$, we can also assume that $|c_1c_2| \geq 1$, while the containment property just mentioned continues to hold. If $|c_1c_2| \leq 1 + 2\delta = 5/4$, then S_1, S_2, S_3, S_4 satisfy property ($\diamond$). In this case, we can apply the algorithm of Theorem 4.5 to find a 1-separated quadruple in $S_1 \times S_2 \times S_3 \times S_4$, or to determine that none exists, in time $O(n^{4/3} \log^{10/3} n)$.

If $|c_1c_2| > 5/4$, any pair of points $p \in S_1$, $q \in S_2$ is 1-separated. We can then apply a simpler variant of the algorithm in section 4.1, in which we ignore any interaction between S_1 and S_2. Thus we may ignore the family Δ of disks, and only consider the intersection points σ_p and not α_p, β_p. Alternatively, we can run the algorithm as is, and the disks $\mathbb{D}(p)$, for $p \in S_1$, will never show up on the overall envelope. In either case, the running time is $O(n^{4/3} \log^{10/3} n)$.

In summary, we show the following theorem.

THEOREM 4.8. *Let S be a set of n points in $\mathbb{R}^2$. A 1-separated quadruple in S can be computed (or be determined not to exist) in time $O(n^{4/3} \log^{10/3} n)$.*

Finally, by performing a binary search on the pairwise distances in S, as in section 3, we obtain the following main result of this section.

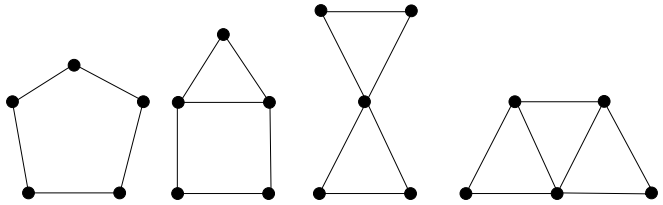


FIG. 9. *Possible graphs for* $k = 5$.

THEOREM 4.9. *Let S be a set of n points in $\mathbb{R}^2$. A maximally separated quadruple in S can be computed in $O(n^{4/3}\log^{13/3} n)$ time.*

4.3. Discussion. The technique that we have presented in sections 3 and 4 can be extended in principle to larger values of k. As above, it suffices to solve the decision problem: Determine whether a 1-separated k-tuple exists in S. Lemmas 4.6 and 4.7 can be extended in a straightforward manner, and they reduce the problem to $O(1)$ subproblems. In each subproblem we have k $\varepsilon \times \varepsilon$ square cells $C_1, \ldots, C_k$ of a grid, and subsets $S_i = S \cap C_i$, for $i = 1, \ldots, k$. Every pair of cells is such that the maximum distance between their points is at least 1, and some pairs of cells are such that the minimum distance between their points is at most 1. The collection of the pairs of the second kind constitutes the edge set of a graph $\mathcal{G}$, and the problem proceeds by case analysis, depending on the structure of $\mathcal{G}$. As above, we may assume that $\mathcal{G}$ is connected, and that the degree of each node is at least two.

For example, consider the case $k = 5$. The possible graphs $\mathcal{G}$ that need to be considered are shown in Figure 9. We leave it as an open problem to design efficient algorithms for the decision problem on each of these graphs, and thus to obtain an efficient algorithm for finding a maximally separated 5-tuple in S.

5. An exact algorithm for an arbitrary *k*. Let S be a set of n points in $\mathbb{R}^2$, and let $k \geq 2$ be an integer. We describe an $n^{O(\sqrt{k})}$-time algorithm for computing a maximally separated subset of S of size k. As in the previous sections, it suffices to focus on the decision problem: Given a set $\mathcal{D}$ of n unit disks and an integer $1 \leq k \leq n$, is there a subset $I \subseteq \mathcal{D}$ of k pairwise-disjoint disks?

Suppose that all the disks of $\mathcal{D}$ lie inside a horizontal strip W of (integer) width w. Using a sweep-line algorithm, similar to the one by Gonzalez [14] for computing a k-center of a set of points, we can compute a largest subset of pairwise-disjoint disks in $n^{O(w)}$ time, as follows.

We define the *index* of a set $A = \{D_1, \ldots, D_q\}$ of unit disks, for $q \leq n$, to be the $2n$-vector

$$\sigma(A) = (\underbrace{0, \ldots, 0, x_1, \ldots, x_q}_{n}, \underbrace{0, \ldots, 0, y_1, \ldots, y_q}_{n}),$$

where (x_i, y_i) is the center of D_i, $x_1 \leq x_2 \leq \cdots \leq x_q$, and if $x_i = x_{i+1}$, then $y_i < y_{i+1}$. We refer to the set of pairwise-disjoint disks with the maximal index in lexicographic order as the *optimal independent set*. The sweep-line algorithm computes the optimal independent set of $\mathcal{D}$, as follows.

For a subset $A \subseteq \mathcal{D}$ and a vertical line ℓ, let $\chi(A, \ell) \subseteq A$ be the set of disks in A that intersect ℓ. We sweep a vertical line ℓ from left to right, stopping at the leftmost and rightmost point of each disk in $\mathcal{D}$. At any time, the algorithm maintains a family $\mathcal{F} = \{I_1, \ldots, I_u\}$ of subsets of pairwise-disjoint disks that satisfies the following invariants:

(I.1) For every $1 \le j \le u$, no disk in $I_j \subseteq \mathcal{D}$ is contained in the (closed) halfplane lying to the right of the sweep line.

(I.2) For $a \ne b$, $\chi(I_a, \ell) \ne \chi(I_b, \ell)$.

(I.3) If there is a subset $A \subseteq \mathcal{D}$ of pairwise-disjoint disks so that no disk in A lies completely to the right of ℓ, then there is a subset $I_j \in \mathcal{F}$ so that $\chi(I_j, \ell) = \chi(A, \ell)$ ($\chi(I_j, \ell)$ may be empty) and $\sigma(A) \le_{\text{lex}} \sigma(I_j)$.

Since at most $O(w)$ pairwise-disjoint disks of $\mathcal{D}$ can intersect ℓ, invariant (I.2) implies that $|\mathcal{F}| = n^{O(w)}$ throughout the sweep. When the sweep line ℓ passes through the leftmost point of a disk $D \in \mathcal{D}$, we test, for each set $I \in \mathcal{F}$, whether D does not intersect any disk in I, and, if so, we add the set $I \cup \{D\}$ to $\mathcal{F}$ (and we also keep I in $\mathcal{F}$). When ℓ passes through the rightmost point of a disk D, we delete all the sets I_a from $\mathcal{F}$ for which there is another set $I_b \in \mathcal{F}$ with $\sigma(I_a) <_{\text{lex}} \sigma(I_b)$ and $\chi(I_a, \ell) \setminus \{D\} = \chi(I_b, \ell) \setminus \{D\}$. We spend $O(|\mathcal{F}|^2 n)$ time at each leftmost or rightmost point of a disk, so the overall running time remains $n^{O(w)}$ (with an appropriate calibration of the constant of proportionality in the exponent). The reader can easily verify that the invariants (I.1)–(I.3) ensure that the algorithm computes the optimal independent set of $\mathcal{D}$.

If $w \le 2\sqrt{k}+2$, we use the above algorithm to determine, in $n^{O(\sqrt{k})}$ time, whether $\mathcal{D}$ contains k pairwise-disjoint disks. So assume that $w > 2\sqrt{k}+2$. Let Π be the set of at most $2n$ horizontal lines tangent to disks in $\mathcal{D}$. The following packing lemma was proved by Agarwal and Procopiuc [3].

LEMMA 5.1. *Let $\mathcal{D}$ be a set of n disks in $\mathbb{R}^2$, and let $\mathcal{D}' \subseteq \mathcal{D}$ be a subset of at most k pairwise-disjoint unit disks that lie in a horizontal strip of width larger than $2\sqrt{k}+2$. Then there exists a horizontal line tangent to one of the disks in $\mathcal{D}$ that intersects at most $\sqrt{k}$ disks of $\mathcal{D}'$.*

For any $h \in \Pi$ and $I \subseteq \mathcal{D}$, we call (h, I) a *canonical pair* if $|I| \le \sqrt{k}$, the disks in I are pairwise disjoint, and all disks in I intersect h (in general, h may also intersect other disks of $\mathcal{D}$). We define a *c-strip* to be a triple $\tau = (\omega, A_1, A_2)$, where ω is a strip bounded by two lines $\ell_1, \ell_2 \in \Pi$, with ℓ_1 lying above ℓ_2, and (ℓ_1, A_1) and (ℓ_2, A_2) are canonical pairs; A_1, A_2 are not necessarily disjoint. Let $\mathcal{D}_\tau \subseteq \mathcal{D}$ be the set of disks that do not intersect ℓ_1, ℓ_2 or any disk of $A_1 \cup A_2$. We define the optimal independent set of τ, denoted by $\mathbb{I}_\tau$, to be the optimal independent set of $\mathcal{D}_\tau$, and set $\kappa_\tau := |\mathbb{I}_\tau|$. We call τ *thin* if the width of ω is at most $2\sqrt{k}+2$, and *thick* otherwise.

For a given $\tau = (\omega, A_1, A_2)$, we compute $\mathbb{I}_\tau$ as follows. If τ is thin, then we compute $\mathbb{I}_\tau$ using the sweep-line algorithm described above. So assume that τ is thick. If $\mathcal{D}_\tau \ne \emptyset$, then by Lemma 5.1, there exists a canonical pair (h, I) so that h divides ω into two strips ω^+, ω^- each of width less than that of ω. Let $\tau^+ = (\omega^+, A_1, I)$ and $\tau^- = (\omega^-, I, A_2)$. We compute $\mathbb{I}_{\tau^+}$ and $\mathbb{I}_{\tau^-}$ recursively, and output $\mathbb{I}_\tau := \mathbb{I}_{\tau^+} \cup I \cup \mathbb{I}_{\tau^-}$. Since we do not know the true canonical pair (h, I), we try all canonical pairs and choose the one for which the solution has the largest index. Moreover, instead of solving the problem recursively, we use a bottom-up approach based on dynamic programming.

In particular, we build a table, each of whose entries corresponds to a c-strip $\tau = (w, A_1, A_2)$ and stores κ_τ and $\sigma_\tau = \sigma(\mathbb{I}_\tau)$. If we ever encounter an entry with

$\kappa_\tau > k$, we can conclude that the size of the largest independent set in $\mathcal{D}$ is greater than k, and we restart the algorithm with a new larger value of k. So we assume that $\kappa_\tau \leq k$ for all entries. We fill the entries of the table as follows. If $\mathcal{D}_\tau = \emptyset$, we set $\mathbb{I}_\tau := \emptyset$, and if τ is thin, we compute $\mathbb{I}_\tau$ using the sweep-line algorithm and fill the entry. Otherwise, we compute all canonical pairs (h, I) for which h lies inside ω, and let ω^+ (resp., ω^-) be the portion of ω lying above (resp., below), and let $\tau^+ = (\omega^+, A_1, I)$ and $\tau^- = (\omega^-, I, A_2)$. Compute

$$\kappa_\tau := \max_{(h,I)} \left\{ \kappa_{\tau^+} + \kappa_{\tau^-} + |I \setminus (A_1 \cup A_2)| \right\},$$

where the maximum is taken over all canonical pairs. Let (h^*, I^*) be the canonical pair for which the maximum is attained. Then $\sigma(\tau)$ is the index of $\mathbb{I}_{\tau^*_+} \cup \mathbb{I}_{\tau^*_-} \cup I^*$, where τ^*_+, τ^*_- are the c-strips defined by the canonical pair (h^*, I^*). If the maximum value of κ_τ is attained by more than one canonical pair, we choose the one for which $\sigma(\tau)$ is maximal.

Let $(\omega_1, \ldots, \omega_M)$, for $M = O(n^2)$, be the sequence of horizontal strips determined by pairs of lines in Π, sorted in nondecreasing order of their widths. We fill out the entries of the table having ω_i as the first component of the index before filling the entries with ω_{i+1} as their first component. If $\mathcal{D}$ contains an independent set of size at most k, then the optimal solution for the c-strip $(\omega_M, \emptyset, \emptyset)$ gives the size and index of the optimal independent set of $\mathcal{D}$. Since there are $n^2 \cdot n^{O(\sqrt{k})} \cdot n^{O(\sqrt{k})}$ c-strips, and each entry can be filled in $n^{O(\sqrt{k})}$ time, the overall running time of the decision algorithm is $n^{O(\sqrt{k})}$ (again, with calibration of the constant of proportionality). We thus obtain the following theorem.

THEOREM 5.2. *Let $\mathcal{D}$ be a set of n unit disks in $\mathbb{R}^2$. The optimal independent set $\mathbb{I}$ of $\mathcal{D}$ can be computed in time $n^{O(\sqrt{k})}$, where k is the size of $\mathbb{I}$.*

Returning to the problem of computing a maximally separated subset of S of size k, we perform a binary search on the pairwise distances in S. At each step, we need to determine whether there exists a maximally separated subset $I \subseteq S$ of size k with $d_{\text{sep}}(I) \geq r$, for a given r, which reduces to determining whether there is an independent set of size k in the set $\{D(p, r) \mid p \in S\}$. Using Theorem 5.2, we thus obtain the following theorem.

THEOREM 5.3. *Let S be a set of n points in $\mathbb{R}^2$, and let $1 \leq k \leq n$ be an integer. A maximally separated subset of S of size k can be computed in $n^{O(\sqrt{k})}$ time.*

6. Conclusion. In this paper we have presented efficient exact and approximation algorithms for computing maximally separated subsets of a set of points in the plane. The approximation algorithm runs in linear time for fixed values of k, and the exact algorithm runs in time $n^{O(\sqrt{k})}$. We also presented $O(n^{4/3}\text{polylog}(n))$-time algorithms for $k = 3, 4$. As mentioned in Section 4.3, it is not clear whether our approach gives a subquadratic algorithm for $k = 5$. Another interesting open problem is whether a maximally separated subset of size k can be computed in time $n^{O(1)} + k^{O(\sqrt{k})}$.

REFERENCES

[1] P. K. AGARWAL, S. HAR-PELED, AND K. R. VARADARAJAN, *Approximating extent measures of points*, J. ACM, 51 (2004), pp. 606–635.

[2] P. K. AGARWAL, M. VAN KREVELD, AND S. SURI, *Label placement by maximum independent set in rectangles*, Comput. Geom., 11 (1998), pp. 209–218.

[3] P. K. Agarwal and C. M. Procopiuc, *Exact and approximation algorithms for clustering*, Algorithmica, 33 (2002), pp. 201–226.
[4] P. K. Agarwal, M. Sharir, and E. Welzl, *The discrete 2-center problem*, Discrete Comput. Geom., 20 (1998), pp. 287–305.
[5] J. Alber and J. Fiala, *Geometric separation and exact solutions for the parameterized independent set problem on disk graphs*, J. Algorithms, 52 (2004), pp. 134–151.
[6] R. Boppana and M. M. Halldórsson, *Approximating maximum independent sets by excluding subgraphs*, BIT, 32 (1992), pp. 180–196.
[7] T. M. Chan, *Polynomial-time approximation schemes for packing and piercing fat objects*, J. Algorithms, 46 (2003), pp. 178–189.
[8] T. M. Chan, *A note on maximum independent sets in rectangle intersection graphs*, Inform. Process. Lett., 89 (2004), pp. 19–23.
[9] B. N. Clark, C. J. Colbourn, and D. S. Johnson, *Unit disk graphs*, Discrete Math., 86 (1990), pp. 165–177.
[10] M. de Berg, M. van Kreveld, M. H. Overmars, and O. Schwarzkopf, Computational Geometry: Algorithms and Applications, 2nd ed., Springer-Verlag, Berlin, 2000.
[11] T. Erlebach, K. Jansen, and E. Seidel, *Polynomial-time approximation schemes for geometric intersection graphs*. SIAM J. Comput., 34 (2005), pp. 1302–1323.
[12] M. Formann and F. Wagner, *A packing problem with applications to lettering of maps*, in Proceedings of the 7th Annual Symposium on Computational Geometry, North Conway, NH, 1991, pp. 281–288.
[13] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.
[14] T. Gonzalez, *Clustering to minimize the maximum intercluster distance*, Theoret. Comput. Sci., 38 (1985), pp. 293–306.
[15] J. Hastad, *Clique is hard to approximate within* $n^{1-\varepsilon}$, Acta Math., 182 (1999), pp. 105–142.
[16] J. Hershberger, *Finding the upper envelope of* n *line segments in* $O(n \log n)$ *time*, Inform. Process. Lett., 33 (1989), pp. 169–174.
[17] H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns, *NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs*, J. Algorithms, 26 (1998), pp. 238–274.
[18] M. Katz and M. Sharir, *An expander-based approach to geometric optimization*, SIAM J. Comput., 26 (1997), pp. 1384–1408.
[19] M. V. Marathe, H. Beru, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz, *Simple heuristics for unit graphs*, Networks, 25 (1995), pp. 59–68.
[20] D. Marx, *Efficient approximation schemes for geometric problems?*, in Proceedings of the 13th Annual European Symposium Algorithm, Palma de Mallorca, Spain, 2005, pp. 448–459.
[21] M. Sharir and P. K. Agarwal, *Davenport-Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, Cambridge, UK, 1995.
[22] E. J. van Leeuwen, *Approximation algorithms for unit disk graphs*, Proceedings of the 31st International Workshop on Graph Theoretic Algorithms, Metz, France, 2005, pp. 351–361.
[23] J. Vleugels and R. C. Veltkamp, *Efficient image retrieval through vantage objects*, Pattern Recognition, 35 (2002), pp. 69–80.

SIAM J. COMPUT.
Vol. 36, No. 3, pp. 835–843

A PROBABILISTIC APPROACH TO THE DICHOTOMY PROBLEM*

TOMASZ ŁUCZAK† AND JAROSLAV NEŠETŘIL‡

Abstract. Let $\mathcal{R}(n,k)$ denote the random k-ary relation defined on the set $[n]=\{1,2,\dots,n\}$. We show that the probability that $([n],\mathcal{R}(n,k))$ is projective tends to one, as either n or k tends to infinity. This result implies that for most relational systems $(B,\underline{R})$ the CSP$(B,\underline{R})$ problem is NP-complete (and thus that the dichotomy conjecture holds with probability 1), and confirms a conjecture of Rosenberg [I. G. Rosenberg, *Rocky Mountain J. Math.*, 3 (1973), pp. 631–639].

Key words. CSP, projectivity, random relation

AMS subject classifications. Primary, 68Q15; Secondary, 08A05, 05D40, 68R05

DOI. 10.1137/S0097539703435492

1. Introduction.

Let $\Delta=(\delta_i)_{i\in I}$ be a finite sequence of positive integers. A *relational system of type* Δ is a pair $(A,\underline{R})$, where A is a finite set, $\underline{R}=(R_i)_{i\in I}$, and R_i is a δ_i-ary relation on A, i.e., $R_i\subseteq A^{\delta_i}$ for $i\in I$. A system $(A,(R_i)_{i\in I})$ is *loop-free* if for every $a\in A$ and $i\in I$ the δ_i-tuple $(a,\dots,a)$ is not in R_i. If $(A,\underline{R})$, $(A',\underline{R}')$ are relational systems of the same type $\Delta=(\delta_i)_{i\in I}$, then by a *homomorphism* from $(A',\underline{R}')$ to $(A'',\underline{R}'')$ we mean a map $f:A'\to A''$ such that for every $i\in I$, we have $(f(x_1),\dots f(x_{\delta_i}))\in R_i''$ whenever $(x_1,\dots,x_{\delta_i})\in R_i'$ (for a recent introduction to homomorphisms of graphs and structures, see [8].)

Relational structures and homomorphisms express various decision and counting combinatorial problems such as coloring, satisfiability, and linear algebra problems. Many of them can be reduced to special cases of a general constraint satisfaction problem CSP$(B,\underline{R})$ which, in the language of homomorphisms, can be stated as follows (cf. [6]): *given a relational system* $(A,\underline{R}')$ *decide if there exists a homomorphism* $f:(A,\underline{R}')\to(B,\underline{R})$. A number of such problems have been studied and have known complexity, e.g., when we deal with undirected graphs or the problem is restricted to small sets A (see [4, 7, 14]). However at this moment we are far from understanding the behavior of the CSP$(B,\underline{R})$ problem even for binary relations $(B,\underline{R})$ (i.e., for relational systems of type $\Delta=(2)$). It seems that "most" CSP$(B,\underline{R})$ are hard (NP-complete) problems while few exceptions from this seem to be polynomial. Thus, for instance, for undirected graphs (i.e., symmetric binary relations) CSP$(B,\underline{R})$ is always a hard problem with exactly three exceptions: when $(B,\underline{R})$ is a loop, or a single vertex (with no relation), or a symmetric edge [7]. Results for other solved cases are similar (see [4, 14]).

One of the basic unsolved questions in this area of constraint satisfaction problems is the following [6].

CONJECTURE 1 (dichotomy conjecture). *Every CSP$(B,\underline{R})$ problem is either NP-complete or polynomially solvable.*

*Received by the editors September 26, 2003; accepted for publication (in revised form) April 29, 2006; published electronically November 14, 2006.

http://www.siam.org/journals/sicomp/36-3/43549.html

†Faculty of Mathematics and Computer Science, Adam Mickiewicz University, 61-614 Poznań, Poland (tomasz@amu.edu.pl). This author was partially supported by KBN grant 2 P03A 016 23.

‡Department of Applied Mathematics, Institute of Theoretical Computer Sciences (ITI), Faculty of Mathematics and Physics, Charles University, Praha, Czech Republic (nesetril@kam.ms.mff.cuni.cz). This author was partially supported by grants LN00A56 and 1M0021620808 of the Czech Ministry of Education.

This remains widely open even for $\Delta = (2)$ (in fact, the complexity of the problem is reflected by this case). In this paper we consider a "probabilistic version" of this problem showing that $\mathrm{CSP}(B, \underline{R})$ is NP-complete for most "large" loop-free relational systems $(B, \underline{R})$.

THEOREM 2. *Let $\Delta = (\delta_i)_{i\in I}$ be such that $\max_{i\in I} \delta_i \geq 2$. Then CSP$(B, \underline{R})$ is NP-complete for almost all loop-free relational systems $(B, \underline{R})$ of type Δ.*

Note that for $(B, \underline{R})$ of type $(1, 1, \ldots, 1)$ the problem $\mathrm{CSP}(B, \underline{R})$ is trivial. We also remark that our probabilistic approach is very different from those considered by Achlioptas et al. [1] who study random instances of $\mathrm{CSP}(B, \underline{R})$ for a given (i.e., nonrandom) relational system $(B, \underline{R})$.

The paper is organized as follows. In the following section we formulate the main result of the paper, Theorem 4, which states that most relational systems are strongly rigid, as conjectured by Rosenberg [13]. We also make the formulation of Theorem 2 precise by introducing a notion of a random k-ary relation and remark that Theorem 2 follows from Theorem 4. Then, we introduce "reach" relations and show that reachness implies projectivity. Finally, in section 4, we prove Theorem 4.

2. Projective and rigid relational system. Let $(A, \underline{R})$ be a relational system of type $\Delta = (\delta_i)_{i\in I}$. By (A^ℓ, R^ℓ) we denote the relational system of type Δ such that for every $i \in I$

$$\big((a_1^1, \ldots, a_\ell^1), \ldots, (a_1^k, \ldots, a_\ell^k)\big) \in R_i^\ell$$

if and only if $(a_j^1, \ldots, a_j^k) \in R_i$ for each $j = 1, \ldots, \ell$ (thus $k = \delta_i$). An *operation* (*polymorphism* in [2, 5, 9, 13]) of a relational system (A, R) is a homomorphism $f : A^\ell \to A$ from (A^ℓ, R^ℓ) to (A, R) for some $\ell \geq 1$. Such an operation is *idempotent* if $f(a, \ldots, a) = a$ for every $a \in A$, and it is a *projection* (on the jth coordinate) if there exists j, $1 \leq j \leq \ell$, such that for every $(a_1, \ldots, a_\ell) \in A^\ell$ we have $f(a_1, \ldots, a_\ell) = a_j$. We say that a system (A, R) is *projective* if for every $\ell \geq 1$ every idempotent operation $f : A^\ell \to A$ is a projection (cf., [10, 11]). The system $(A, \underline{R})$ is *rigid* if the identity mapping is the only homomorphism from A to A and *strongly rigid* if it is both projective and rigid. It is easy to see that $(A, \underline{R})$ is strongly rigid if and only if for each $\ell \geq 1$ every operation $f : A^\ell \to A$ is a projection.

The notions of projective and rigid relational systems play an important role in investigations of complexity of $\mathrm{CSP}(B, \underline{R})$ problems. In particular, it is known that these problems are hard whenever the system $(B, \underline{R})$ is strongly rigid, i.e., the following result holds (see, for instance, [2, 3, 5, 9]).

THEOREM 3. *If a relational system $(B, \underline{R})$ is strongly rigid, then the problem CSP$(B, \underline{R})$ is* NP*-complete.*

Thus, in order to show Theorem 2, it is enough to verify that most of relational systems are strongly rigid. In order to make this statement precise, let $\mathcal{R}(n, k)$ denote a random k-ary relation defined on a set $[n] = \{1, 2, \ldots, n\}$ for which the probability that $(a_1, \ldots, a_k) \in \mathcal{R}(n, k)$ is equal to $1/2$ independently for each $(a_1, \ldots, a_k)$, where $1 \leq a_r \leq n$ for $r = 1, \ldots, k$ and not all a_i's are equal; for $a \in [n]$, we put $(a, a, \ldots, a) \notin \mathcal{R}(n, k)$. Let $([n], (\mathcal{R}(\delta_i, n)_{i\in I}))$ denote the random relational system of type $\Delta = (\delta_i)_{i\in I}$. We shall show that the probability that $([n], (\mathcal{R}(\delta_i, n)_{i\in I}))$ is strongly rigid tends to one as either n or $\max_i \delta_i$ tends to infinity. Note that a relational system $(A, \underline{R})$ is strongly rigid, provided for some $i_0 \in I$ the system (A, R_{i_0}) of type (δ_{i_0}) is strongly rigid (although the converse implication, in general, does not hold). Thus, it is enough to prove our result for "simple" relational systems which consist of just one k-ary relation.

THEOREM 4. *For a fixed $k \geq 2$,*

$$\lim_{n\to\infty} \Pr\big(([n], \mathcal{R}(k,n)) \text{ is strongly rigid}\big) = 1, \tag{1}$$

while, for a given $n \geq 2$,

$$\lim_{k\to\infty} \Pr\big(([n], \mathcal{R}(k,n)) \text{ is strongly rigid}\big) = 1. \tag{2}$$

The proof of Theorem 4 we postpone until section 4. It is based on an argument which is somewhat similar to that used by the authors in [12] to show that almost every graph (i.e., almost every binary symmetric relation) is strongly rigid.

We also remark that (2) settles in the affirmative a conjecture posed by Rosenberg [13].

3. Reach relational systems. Let R be a k-ary relation on a set A. A system (A, R) is *reach* if there exist elements $z_1, \ldots, z_{k-2} \in A$ such that for any four different elements $x_1, x_2, y_1, y_2 \in A$, there is a $w \in A$ such that for $r = 1, 2$, we have

$$(z_1, \ldots, z_{k-2}, w, x_r) \in R \text{ but } (z_1, \ldots, z_{k-2}, w, y_r) \notin R. \tag{3}$$

We shall show that reachness implies projectivity.

THEOREM 5. *Each reach system (A, R) with $|A| \geq 5$ is projective.*

Before we prove Theorem 5 we introduce some more notation. Let $f : A^\ell \to A$ be an idempotent operation from A^ℓ to A. For an ℓ-tuple $(a_1, \ldots, a_\ell) \in A^\ell$, we set

$$\Xi(a_1, \ldots, a_\ell) = \{a \in A : a = a_i \text{ for some } i = 1, 2, \ldots, \ell\},$$

by $\xi(a_1, \ldots, a_\ell) = |\Xi(a_1, \ldots, a_\ell)|$ we denote the number of different coordinates of $(a_1, \ldots, a_\ell)$, and we set

$$\Lambda_f(a_1, \ldots, a_\ell) = \{i : f(a_1, \ldots, a_\ell) = a_i\}.$$

Hence, for instance, $\Xi(a, \ldots, a) = \{a\}$, $\xi(a, \ldots, a) = 1$, and, since f is idempotent, $\Lambda_f(a, \ldots, a) = \{1, 2, \ldots, \ell\}$. Our proof of Theorem 5 is based on the following two claims.

CLAIM 1. *If (A, R) is reach, then for every idempotent operation $f : A^\ell \to A$, and every $(a_1, \ldots, a_\ell) \in A^\ell$,*

$$\Lambda_f(a_1, \ldots, a_\ell) \neq \emptyset. \tag{4}$$

Proof. We shall use induction on $\xi(a_1, \ldots, a_\ell)$. As we have already observed, the fact that f is idempotent implies that $\Lambda_f(a, \ldots, a) = \{1, 2, \ldots, \ell\}$. Let us suppose that the assertion holds for every $(a_1, \ldots, a_\ell)$ with at most m, $1 \leq m \leq \ell - 1$, different coordinates, and let $(b_1, \ldots, b_\ell)$ be such that $\Xi(b_1, \ldots, b_\ell) = \{c_1, \ldots, c_{m+1}\}$. For a contradiction let us assume that

$$f(b_1, \ldots, b_\ell) = d \notin \{c_1, \ldots, c_{m+1}\}.$$

Because R is reach one can choose from A elements e_i, $i = 1, \ldots, k-2$, and $\bar{c}_j$, $j = 1, \ldots, m$, such that for each $j = 1, \ldots, m$, we have

$$(e_1, \ldots, e_{k-2}, \bar{c}_j, c_j) \in R \quad \text{but} \quad (e_1, \ldots, e_{k-2}, \bar{c}_j, d) \notin R,$$

and

$$(e_1, \ldots, e_{k-2}, \bar{c}_m, c_{m+1}) \in R.$$

For $i = 1, 2, \ldots, \ell$, define

$$\bar{b}_i = \begin{cases} \bar{c}_j & \text{if} \quad b_i = c_j \text{ for some } j = 1, \ldots, m, \\ \bar{c}_m & \text{if} \quad b_i = c_{m+1}. \end{cases}$$

Then $\xi(\bar{b}_1, \ldots, \bar{b}_\ell) = m$ and so, by the inductive assumption, for some $s_0 = 1, \ldots, m$, we have $f(\bar{b}_1, \ldots, \bar{b}_\ell) = \bar{c}_{s_0}$. Note however that the k-tuple

$$\big((e_1, \ldots, e_1), (e_2, \ldots, e_2), \ldots, (e_{k-2}, \ldots, e_{k-2}), (\bar{b}_1, \ldots, \bar{b}_\ell), (b_1, \ldots, b_\ell)\big)$$

belongs to R^ℓ but is mapped by f into $(e_1, \ldots, e_{k-2}, \bar{c}_{s_0}, d)$, which, due to the choice of $\bar{c}_{s_0}$, does not belong to R. This contradiction shows that $\Lambda_f(b_1, \ldots, b_\ell) \neq \emptyset$. □

Claim 2. *If (A, R) is reach, then for every idempotent operation $f : A^\ell \to A$, and every pair of ℓ-tuples $(a_1, \ldots, a_\ell)$, $(b_1, \ldots, b_\ell)$, there exists t, $1 \leq t \leq \ell$, such that*

$$\{a_t, b_t\} \subseteq \{f(a_1, \ldots, a_\ell), f(b_1, \ldots, b_\ell)\}. \tag{5}$$

Proof. Claim 1 implies that $f(a_1, \ldots, a_\ell) = a_{j_1}$, $f(b_1, \ldots, b_\ell) = b_{j_2}$ for some indices j_1, j_2, $1 \leq j_1, j_2 \leq \ell$. Since (A, R) is reach one can find elements e_i, $i = 1, \ldots, k-2$, and $c_1, \ldots, c_\ell$ such that for each $s = 1, \ldots, \ell$,

$$(e_1, \ldots, e_{k-2}, c_s, a_s), (e_1, \ldots, e_{k-2}, c_s, b_s) \in R \tag{6}$$

but

$$(e_1, \ldots, e_{k-2}, c_s, w) \notin R \quad \text{for} \quad w \in \{a_{j_1}, b_{j_2}\} \setminus \{a_s, b_s\}. \tag{7}$$

From Claim 1 it follows that $f(c_1, \ldots, c_\ell) = c_t$ for some t, $1 \leq t \leq \ell$. Note also that both k-tuples

$$\big((e_1, \ldots, e_1), \ldots, (e_{k-2}, \ldots, e_{k-2}), (c_1, \ldots, c_\ell), (a_1, \ldots, a_\ell)\big)$$

and

$$\big((e_1, \ldots, e_1), \ldots, (e_{k-2}, \ldots, e_{k-2}), (c_1, \ldots, c_\ell), (b_1, \ldots, b_\ell)\big)$$

belong to R^ℓ, and so we must have

$$(e_1, \ldots, e_{k-2}, c_t, a_{j_1}), (e_1, \ldots, e_{k-2}, c_t, b_{j_2}) \in R\,.$$

This fact, together with (7), implies that $\{a_t, b_t\} \subseteq \{a_{j_1}, b_{j_2}\}$. □

Proof of Theorem 5. Let

$$\hat{A}^\ell = \{(a_1, \ldots, a_\ell) \in A^\ell : \xi(a_1, \ldots, a_\ell) \leq 3\}.$$

Let us observe that for some $(a_1, \ldots, a_\ell) \in \hat{A}^\ell$ we have $|\Lambda_f(a_1, \ldots, a_\ell)| = 1$. Indeed, take $(a_1, \ldots, a_\ell) \in \hat{A}^\ell$ for which $|\Lambda_f(a_1, \ldots, a_\ell)|$ is minimal. Since, by Claim 1,

$|\Lambda_f(a_1,\dots,a_\ell)| \geq 1$, assume that $j \in \Lambda_f(a_1,\dots,a_\ell)$. Let $b, b' \in A \setminus \Xi(a_1,\dots,a_\ell)$ and let

$$c_i = \begin{cases} b & \text{if } i = j, \\ b' & \text{if } i \in \Lambda_f(a_1,\dots,a_\ell) \setminus \{j\}, \\ a_j & \text{if } i \notin \Lambda_f(a_1,\dots,a_\ell). \end{cases}$$

Then $(c_1,\dots,c_\ell) \in \hat{A}^\ell$, but Claim 2 implies that $\Lambda_f(c_1,\dots,c_\ell)$ is a proper subset of $\Lambda_f(a_1,\dots,a_\ell)$. Hence, there is an $(a_1,\dots,a_\ell) \in A^\ell$ such that $\Lambda_f(a_1,\dots,a_\ell) = \{t\}$ for some $1 \leq t \leq \ell$.

Let us fix such $(a_1,\dots,a_\ell)$ and t, pick $b \in A \setminus \Xi(a_1,\dots,a_\ell)$, and define

$$\bar{a}_i = \begin{cases} b & \text{if } i = t, \\ a_i & \text{if } i \neq t. \end{cases}$$

Then, using again Claims 1 and 2, we infer that $\Lambda_f(\bar{a}_1,\dots,\bar{a}_\ell) = \{t\}$. A similar argument shows that $\Lambda_f(d_1,\dots,d_\ell) = \{t\}$ whenever $(d_1,\dots,d_\ell)$ belongs to the set $\tilde{A}^\ell \subset \hat{A}^\ell$ which consists of all ℓ-tuples $(d_1,\dots,d_\ell)$ such that for all $i,j \neq t$ we have $d_i = d_j \neq d_t$.

Now let $(a_1,\dots,a_\ell)$ be any ℓ-tuple of A^ℓ and suppose that $f(a_1,\dots,a_\ell) = a_s \neq a_t$ for some s, $1 \leq s \leq \ell$. Consider $(d_1,\dots,d_\ell) \in \tilde{A}^\ell$ such that

$$d_i = \begin{cases} a_s & \text{if } i = t, \\ a_t & \text{if } i \neq s. \end{cases}$$

Then, the pair $(a_1,\dots,a_\ell)$, $(d_1,\dots,d_\ell)$ contradicts Claim 2. Consequently, for every $(a_1,\dots,a_\ell)$ we have $f(a_1,\dots,a_\ell) = a_t$, and the assertion follows. □

In order to deal with relational systems defined on small sets, we introduce one more definition. Let $2 \leq i \leq k-1$ and (A,R) be a relation system with k-ary relation R. We call (A,R) *i-reach* if (A,R) is reach and there exist elements $z_1,\dots,z_{k-i-1} \in A$ such that for every two different i-tuples $(x_1,\dots,x_i)$, $(y_1,\dots,y_i) \in A^i$, there is a $w \in A$ for which

$$(z_1,\dots,z_{k-i-1},w,x_1,\dots,x_i) \in R \tag{8}$$

but

$$(z_1,\dots,z_{k-i-1},w,y_1,\dots,y_i) \notin R. \tag{9}$$

It turns out that (A,R) is projective also for $|A| \geq 2$ if the assumption of Theorem 5 is slightly strengthened.

THEOREM 6. *If (A,R) is 3-reach, then (A,R) is projective.*

The following result, which is, in a way, a generalization of Claim 1, can be shown by imitating the proof of Claim 2.

CLAIM 3. *If (A,R) is i-reach for some $i \geq 2$, then for every set of i ℓ-tuples $(a_1^r,\dots,a_\ell^r)$, $r = 1,2,\dots,i$, we have*

$$\bigcap_{r=1}^{i} \Lambda_f(a_1^r,\dots,a_\ell^r) \neq \emptyset. \tag{10}$$

Proof. Let us assume that (A,R) is i-reach, and let $(a_1^r,\dots,a_\ell^r)\in A^\ell$ for $r=1,\dots,i$. From Claim 1 it follows that there exist indices j_r, $1\le j_r\le \ell$, such that $f(a_1^r,\dots,a_\ell^r)=a_{j_r}^r$ for $r=1,\dots,i$. Since (A,R) is i-reach one can find elements e_i, $i=1,\dots,k-i-1$, and $c_1,\dots,c_\ell$, such that for each $s=1,\dots,\ell$,

$$(e_1,\dots,e_{k-i-1},c_s,a_s^1,\dots,a_s^i)\in R \tag{11}$$

but

$$(e_1,\dots,e_{k-2},c_s,a_{j_1}^1,\dots,a_{j_i}^i)\notin R \quad\text{unless}\quad (a_s^1,\dots,a_s^i)=(a_{j_1}^1,\dots,a_{j_i}^i). \tag{12}$$

Claim 1 implies that $f(c_1,\dots,c_\ell)=c_t$ for some t, $1\le t\le \ell$. Note that for each $r=1,2,\dots,i$, the k-tuple

$$\big((e_1,\dots,e_1),\dots,(e_{k-i-1},\dots,e_{k-i-1}),(c_1,\dots,c_\ell),(a_1^1,\dots,a_\ell^1),\dots,(a_1^i,\dots,a_\ell^i)\big)$$

belongs to R^ℓ, and so its image under f, equal to $(e_1,\dots,e_{k-i-1},\ c_t,\ a_{j_1}^1,\dots,a_{j_i}^i)$, belongs to R. But then (12) implies that $(a_t^1,\dots,a_t^i)=(a_{j_1}^1,\dots,a_{j_i}^i)$, i.e.,

$$t\in\bigcap_{r=1}^{i}\Lambda_f(a_1^r,\dots,a_\ell^r). \qquad \square$$

Proof of Theorem 6. As in the proof of Theorem 5 we show first that for some $(a_1,\dots,a_\ell)\in A^\ell$ we have $|\Lambda_f(a_1,\dots,a_\ell)|=1$.

Indeed, suppose that $|\Lambda_f(a_1,\dots,a_\ell)|$ is minimal, $|\Lambda_f(a_1,\dots,a_\ell)|\ge 2$. Choose $j\in\Lambda_f(a_1,\dots,a_\ell)$, and $b\in A\setminus\{a_j\}$. Define

$$c_i=\begin{cases} a_j & \text{if } i\ne j,\\ b & \text{if } i=j.\end{cases}$$

Then we have the following possibilities for $\Lambda_f(c_1,\dots,c_\ell)$:

(i) $\Lambda_f(c_1,\dots,c_\ell)=\{j\}$ (in the case $f(c_1,\dots,c_\ell)=b$),

(ii) $\Lambda_f(c_1,\dots,c_\ell)\cap\Lambda_f(a_1,\dots,a_\ell)=\emptyset$ (if $f(c_1,\dots,c_\ell)\ne a_j,b$),

(iii) $\Lambda_f(c_1,\dots,c_\ell)=\Lambda_f(a_1,\dots,a_\ell)$ (in the case $f(c_1,\dots,c_\ell)=a_j$).

In the first case we are done, the second case is impossible by Claim 3, and the last case is impossible as we have chosen $(a_1,\dots,a_\ell)$ such that $|\Lambda_f(a_1,\dots,a_\ell)|$ is minimal. Thus, for some $(a_1,\dots,a_\ell)\in A^\ell$, we have $\Lambda_f(a_1,\dots,a_\ell)=\{t\}$. Now the assertion follows from Claim 3. $\square$

4. Proof of the main result. Let us start with the following result, which, in fact, gives more than we need.

LEMMA 7. *Let $i\ge 2$ be a fixed natural number.*

(i) *For a fixed $k>i$*

$$\lim_{n\to\infty}\Pr\big(([n],\mathcal{R}(n,k))\ \textit{is } i\textit{-reach}\big)=1.$$

(ii) *For a fixed $n\ge 2$*

$$\lim_{k\to\infty}\Pr\big(([n],\mathcal{R}(n,k))\ \textit{is } i\textit{-reach}\big)=1.$$

Proof. Let $k > i \geq 2$ be fixed and let $z_1, \ldots, z_{k-i-1}$ be any given elements of $[n]$, say, $z_r = r$ for $r = 1, \ldots, k-i-1$. Then, the probability that for two given i-tuples $(x_1, \ldots, x_i)$, $(y_1, \ldots, y_i)$ for all elements w either (8) or (9) does not hold is bounded from above by $(3/4)^n$. Hence the probability that one cannot find a required w, for some of at most n^{2i} possible choices for x's and y's, is smaller than $n^{2i}(3/4)^n$ and tends to 0 as $n \to \infty$. This proves the first part of the lemma.

In order to show (ii) observe first that there are at most $4n^{2i+1}$ choices for i-tuples $(x_1, \ldots, x_i)$, $(y_1, \ldots, y_i)$ and $w \in [n]$. The probability that for all these choices and for given $z_1, \ldots, z_{k-i-1}$ each of the relations (3), (8), and (9) holds is bounded from below by $\rho = 2^{-4n^{2i+1}}$. But there are $N = n^{k-i-1}$ possible choices for z's and so the number of $(k-i-1)$-tuples $(z_1, \ldots, z_{k-i-1})$ for which (3), (8), and (9) hold is bounded from below by the random variable Z with binomial distribution $Bi(N, \rho)$. Since $N \to \infty$ as $k \to \infty$ but ρ remains bounded away from 0, with probability tending to 1 as $k \to \infty$, we have $Z > 0$. Hence, with probability tending to 1 as $k \to \infty$, there exists a choice of $z_1, \ldots, z_{k-i-1}$ for which (3), (8), and (9) hold (for all $w \in A$!) and (ii) follows. □

Let us remark that in Lemma 7(i) it would be sufficient for our purposes to prove that for every fixed $k \geq 2$ we have

$$\lim_{n\to\infty} \Pr\big(([n], \mathcal{R}(n,k)) \text{ is reach}\big) = 1.$$

This can be proved similarly to (i); nevertheless Lemma 7 gives a stronger (and more symmetric) result. However there are two exceptions: for $k = 2, 3$, where (according to Theorem 6) we have to use the reach property only.

We shall also need to know that the system $([n], \mathcal{R}(n,k))$ typically has no non-trivial endomorphisms. This is a variant on the well-known result for graphs; see, e.g., [8].

LEMMA 8.

(i) *For a fixed $k \geq 2$,*

$$\lim_{n\to\infty} \Pr\big(([n], \mathcal{R}(n,k)) \textit{ is rigid}\big) = 1.$$

(ii) *For a fixed $n \geq 2$,*

$$\lim_{k\to\infty} \Pr\big(([n], \mathcal{R}(n,k)) \textit{ is rigid}\big) = 1.$$

Proof. Let us consider first the case when k is fixed. Let $f : [n] \to [n]$ be an endomorphism of $([n], \mathcal{R}(n,k))$. Note that with probability tending to 1 as $n \to \infty$ for any $S \subseteq [n]$, $|S| = m_0 = \lfloor 2 \log n \rfloor$, we have $|f(S)| \geq 2$. Indeed, if S is mapped into a single point, then, since $\mathcal{R}(n,k)$ is loop-free, for all $a_1, \ldots, a_k \in S$ we have $(a_1, \ldots, a_k) \notin \mathcal{R}(n,k)$, and the probability that such an S exists is bounded from above by

$$\binom{n}{m_0}\Big(\frac{1}{2}\Big)^{m_0^k} \leq \Big(\frac{en}{m}\Big(\frac{1}{2}\Big)^{m_0^{k-1}}\Big)^{m_0} \to 0\,.$$

Let us denote by W the number of vertices of $[n]$ which are not fixed by f. We show that with probability $1 - o(1)$ we have $|W| \leq 17 \log^2 n$. Let us assume that this is not the case. Then, using the fact we have just proved, one can greedily construct two disjoint sets $S_1 \subseteq W$, $S_2 \subseteq f(W)$, $|S_1|, |S_2| \geq \frac{17 \log^2 n}{2 \log n + 1} \geq m_1 = \lfloor 8 \log n \rfloor$, such

that $f|S_1$ is a bijective homomorphism from S_1 to S_2. However, the probability that two such sets exist is bounded from above by

$$\sum_{m_1 \geq \log^2 n} \binom{n}{m_1}\binom{n}{m_1} m_1! \left(\frac{3}{4}\right)^{m_1^k} \leq \sum_{m_1 \geq \log^2 n} \left(\frac{e^2 n^2}{m_1}\left(\frac{3}{4}\right)^{m_1^{k-1}}\right)^{m_1} \to 0\,, \tag{13}$$

since the probability that, given $f : S_1 \to S_2$, and $a_1, \ldots, a_k \in S_1$, we have $(a_1, \ldots, a_k) \in \mathcal{R}(n,k)$ but $(f(a_1), \ldots, f(a_k)) \notin \mathcal{R}(n,k)$ is equal to $1/4$.

In order to complete the proof of (i) one can argue as in the proof of Lemma 7 that with probability $1 - o(1)$, for each pair of elements $v, w \in [n]$ and each $W \subseteq [n]$, $|W| \leq 17 \log^2 n$, there are vertices $z_1, \ldots, z_{k-1} \in [n] \setminus W$ such that $(z_1, \ldots, z_{k-1}, v) \in \mathcal{R}(n,k)$ but $(z_1, \ldots, z_{k-1}, w) \notin \mathcal{R}(n,k)$ tends to one. Hence, with probability tending to one as $n \to \infty$, we have $W = \emptyset$; i.e., each endomorphism of $([n], \mathcal{R}(n,k))$ is an identity.

In order to show (ii) observe that, since with probability tending to 1 as $k \to \infty$ the relation $\mathcal{R}(n,k)$ is nonempty (see Lemma 7), the image of each homomorphism of $([n], \mathcal{R}(n,k))$ contains at least two points. Let us take any nontrivial function $f : [n] \to [n]$ with $|f([n])| \geq 2$, and let $j, j' \in [n]$, be such that $f(j) \neq j, f(j')$. For $r = 1, \ldots, k$, $s = 1, \ldots, k$, let

$$a_r^s = \begin{cases} j & \text{if} \quad r \leq s, \\ j' & \text{if} \quad r > s. \end{cases}$$

Note that all $2k$ k-tuples $(a_1^s, \ldots, a_k^s)$, $(f(a_1^s), \ldots, f(a_k^s))$, $s = 1, \ldots, k$, are different. Furthermore, the probability that for some $s = 1, \ldots, k$, it does not happen that $(a_1^s, \ldots, a_k^s) \in \mathcal{R}(n,k)$ and $(f(a_1^s), \ldots, f(a_k^s)) \notin \mathcal{R}(n,k)$, is $3/4$. Consequently, the probability that for each of at most n^n possible nontrivial mappings $f : [n] \to [n]$ none of the pairs $(a_1^s, \ldots, a_k^s)$, $(f(a_1^s), \ldots, f(a_k^s))$, $s = 1, \ldots, k$, is a "witness" that f is not a homomorphism is bounded from above by $n^n (3/4)^k$ and tends to 0 when n is fixed and $k \to \infty$. Consequently, if $n \geq 2$ is fixed and $k \to \infty$, the probability that there exists nontrivial endomorphism of $([n], \mathcal{R}(n,k))$ tends to 0. □

Proof of Theorem 4. Theorem 4 follows immediately from Theorems 5 and 6 and Lemmas 7 and 8 (and the remark following the proof of Lemma 7). □

REFERENCES

[1] D. Achlioptas, L. M. Kirousis, E. Kranakis, D. Krizanc, M. S. O. Molloy, and Y. C. Stamatiou, *Random constraint satisfaction: A more accurate picture*, Constraints, 6 (2001), pp. 329–344.

[2] M. Bodirsky and J. Nešetřil, *Constraint satisfaction with countable homogeneous templates*, in Computer Science Logic, Lecture Notes in Comput. Sci. 2803, Springer-Verlag, Berlin, 2003, pp. 44–57.

[3] V. G. Bodnarčuk, L. A. Kaluzhnin, V. N. Kotov, and B. A. Romov, *Galois theory for Post algebras* I–II, Kibernetika, 3 (1969), pp. 1–10 and 5 (1969), pp. 1–9 (in Russian); Cybernetics, (1969), pp. 243–252, 531–539 (English version).

[4] A. Bulatov, *A dichotomy theorem for constraints on a three element set*, in Proceedings of the IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2002, pp. 649–658.

[5] A. Bulatov, A. Krokhin, and P. G. Jeavons, *The complexity of maximal constraint languages*, in Proceedings of the ACM Symposium on Theory of Computing, ACM, New York, 2001, pp. 667–674.

[6] T. Feder and M. Y. Vardi, *The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory*, SIAM J. Comput., 28 (1998), pp. 57–104.

[7] P. Hell and J. Nešetřil, *On the complexity of H-coloring*, J. Combin. Theory Ser. B, 48 (1990), pp. 92–110.
[8] P. Hell and J. Nešetřil, *Graphs and Homomorphisms*, Oxford University Press, Oxford, UK, 2004.
[9] P. G. Jeavons, *On the algebraic structure of combinatorial problems*, Theoret. Comput. Sci., 200 (1998), pp. 185–204.
[10] B. Larose and C. Tardif, *Strongly rigid graphs and projectivity*, Mult.-Valued Log., 7 (2001), pp. 339–361.
[11] B. Larose and C. Tardif, *Projectivity and independent sets in powers of graphs*, J. Graph Theory, 40 (2002), pp. 162–171.
[12] T. Łuczak and J. Nešetřil, *A note on projective graphs*, J. Graph Theory, 47 (2004), pp. 81–86.
[13] I. G. Rosenberg, *Strongly rigid relations*, Rocky Mountain J. Math., 3 (1973), pp. 631–639.
[14] T. Schaefer, *The complexity of satisfiability problems*, in Proceedings of the ACM Symposium on Theory of Computing, ACM, New York, 1978, pp. 216–226.

SPECIAL ISSUE ON RANDOMNESS AND COMPLEXITY

The idea of a SICOMP special issue on randomness and complexity occurred to us when we were in residence at the Radcliffe Institute for Advanced Study at Harvard University during the academic year 2003–2004. We were part of a science cluster in theoretical computer science at the Radcliffe Institute whose other members were Eli Ben-Sasson, Dana Ron, Ronitt Rubinfeld, and Salil Vadhan. The focus of this cluster was randomness and computation. The extensive interaction within the cluster members as well as with frequent visitors (most notably Irit Dinur, Shafi Goldwasser, and Tali Kaufman) made us more aware than ever of the richness of the area, and the idea of editing a special issue on randomness and complexity emerged naturally.

The interplay of randomness and complexity is at the heart of modern cryptography and plays a fundamental role in the design of algorithms and in complexity theory at large. Specifically, this interplay is pivotal to several intriguing notions of probabilistic proof systems (e.g., interactive proofs, zero-knowledge proofs, and probabilistically checkable proofs), is the focus of the computational approach to randomness, and is essential for various types of sub–linear time algorithms. All these areas were at the focus of extensive research in the last two decades, but each research generation brings its own new perspective (and/or focus) to them. This special issue reports some of the recent progress achieved in these related areas, where recent relates to spring 2004, when papers were invited for this issue. Following are some of the issue's main themes.

Cryptography. The paper of Applebaum, Ishai, and Kushilevitz provides strong evidence that many cryptographic primitives and tasks can be implemented at very low complexity. For example, they show that the existence of one-way functions that can be evaluated in $\mathcal{NC}1$ implies the existence of one-way functions that can be evaluated in $\mathcal{NC}0$. Whereas the former are widely believed to exist (e.g., based on the standard factoring assumption), most researchers have previously believed that the latter do not exist. We stress that evaluation in $\mathcal{NC}0$ means that each output bit only depends on a constant number of input bits. The new work further shows that dependence on *four* input bits suffices (whereas dependence on at least *three* input bits is definitely necessary).

Probabilistically checkable proofs (PCPs). Current research in the area is marked by a renewed attention to aspects such as the following:

1. Achieving constructs of almost-linear length that can be tested by very few (say constant number of) queries.
2. Obtaining a combinatorial proof of the PCP theorem.
3. Exploration of the relationship between PCP and coding theory (e.g., locally testable codes).
4. Applications of PCPs to obtaining new inapproximability results regarding long-standing problems such as min-bisection.

Specifically, the paper of Ben-Sasson et al. presents significant improvements to the trade-off between proof-length and the number of queries. The paper of Dinur and Reingold makes a major step in the project of obtaining combinatorial proofs of the PCP theorem. Both papers share a reformulation of the proof-composition paradigm, where "proximity testing" and "robustness" play a central role. Finally, Khot's paper puts forward new PCP parameters and introduces new PCP constructions that are used to provide evidence that min-bisection is not approximable up to some constant.

Randomness extraction. The construction of randomness extractors has received much attention in the last two decades. Much of the past work (especially in the 1990s) has focused on extracting randomness from a single weak source while using an auxiliary short (uniformly distributed) seed. The focus was on using the weakest possible form of a source (i.e., a min-entropy source). In contrast, the current era is marked by a focus on stronger sources while disallowing the use of an auxiliary (uniformly distributed) seed. The paper of Gabizon, Raz, and Shaltiel studies bit-fixing sources, whereas the paper of Barak, Impagliazzo, and Wigderson studies extraction from a constant number of independent sources of linear min-entropy (which may be viewed as a single source consisting of a constant number of independent blocks). Indeed, each of these papers revisits problems raised in the mid 1980s, which were neglected in the 1990s (due to the focus of that era on obtaining the best results for seed-assisted extraction from a single min-entropy source). Needless to say, we believe that the renewed interest in these problems (especially the second one) is highly justified.

We wish to seize the opportunity to say a few words regarding seed-assisted versus seedless randomness extraction. Seed-assisted randomness extraction found many applications (via direct and indirect connections to other important problems), but still one may ask what they mean for the original problem of implementing a randomized procedure using a weak source of randomness. One answer is that the seed can be obtained from an expensive high-quality auxiliary source and that one wishes to minimize the use of this source (and thus uses a cheaper low-quality random source for the bulk of the randomness required). Another answer is that if the seed is short enough, then one may afford to try all possible seeds, invoke the procedure with the corresponding randomness extracted (from the same source output and varying seeds), and rule by majority. This suggestion is adequate for the implementation of standard randomized algorithms, but not in "adversarial" settings (e.g., cryptography) in which a randomized procedure is invoked in order to protect against some (adversarial) party. Thus, seedless randomness extraction is essential in many applications.

Worst-case to average-case reductions. The question of whether worst-case to average-case reductions or even merely "hardness amplification" exist for $\mathcal{NP}$ has received much interest recently. The first part of the question is studied in the paper of Bogdanov and Trevisan which provides a negative indication, restricted to nonadaptive reductions. The second part of the question is unfortunately not represented in this special issue (and the interested reader is directed to [1]).

Zero-knowledge. Vadhan's paper presents an *unconditional* study of *computational* zero-knowledge, yielding valuable transformations between various forms of zero-knowledge (e.g., from a weak form of zero-knowledge to the standard form). This work builds on studies of *statistical* zero-knowledge that were conducted in the late 1990s, thus fulfilling a prophecy made at the time.

Low-degree tests. The celebrated low-degree tests have been revisited recently with a focus on derandomization and on low-degree tests over small finite fields. The first direction is represented by the work of Shpilka and Wigderson that seems to provide a "proof from The Book" for (a derandomized version of) the linearity test. The second direction is unfortunately not represented in this special issue (and the interested reader is directed to [2, 3]).

Acknowledgments. We are grateful to the contributing authors for accepting our invitation to publish in this special issue. In some cases, they had to decline other competitive invitations in order to do so.

We are grateful to Eva Tardos for handling the refereeing process of the paper by Ben-Sasson, Goldreich, Harsha, Sudan, and Vadhan.

REFERENCES

[1] A. Healy, S. Vadhan, and E. Viola, *Using nondeterminism to amplify hardness*, in Proceedings of the 36th Symposium on Theory of Computing, 2004, pp. 192–201.

[2] C. S. Jutla, A. C. Patthak, A. Rudra, and D. Zuckerman, *Testing low-degree polynomials over prime fields*, in Proceedings of the 45th Symposium on Foundations of Computer Science, 2004, pp. 423–432.

[3] T. Kaufman and D. Ron, *Testing polynomials over general fields*, in Proceedings of the 45th Symposium on Foundations of Computer Science, 2004, pp. 413–422.

Oded Goldreich
Madhu Sudan
guest editors

SIAM J. COMPUT.
Vol. 36, No. 4, pp. 845–888

CRYPTOGRAPHY IN NC^0*

BENNY APPLEBAUM†, YUVAL ISHAI†, AND EYAL KUSHILEVITZ†

Abstract. We study the parallel time-complexity of basic cryptographic primitives such as one-way functions (OWFs) and pseudorandom generators (PRGs). Specifically, we study the possibility of implementing instances of these primitives by NC^0 functions, namely, by functions in which each output bit depends on a constant number of input bits. Despite previous efforts in this direction, there has been no convincing theoretical evidence supporting this possibility, which was posed as an open question in several previous works.

We essentially settle this question by providing strong positive evidence for the possibility of cryptography in NC^0. Our main result is that every "moderately easy" OWF (resp., PRG), say computable in NC^1, can be compiled into a corresponding OWF (resp., "low-stretch" PRG) in which each output bit depends on at most 4 input bits. The existence of OWFs and PRGs in NC^1 is a relatively mild assumption, implied by most number-theoretic or algebraic intractability assumptions commonly used in cryptography. A similar compiler can also be obtained for other cryptographic primitives such as one-way permutations, encryption, signatures, commitment, and collision-resistant hashing.

Our techniques can also be applied to obtain (unconditional) constructions of "noncryptographic" PRGs. In particular, we obtain ϵ-biased generators and a PRG for space-bounded computation in which each output bit depends on only 3 input bits.

Our results make use of the machinery of *randomizing polynomials* [Y. Ishai and E. Kushilevitz, *Proceedings of the* 41*st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2000, pp. 294–304], which was originally motivated by questions in the domain of information-theoretic secure multiparty computation.

Key words. cryptography, constant depth circuits, NC^0, cryptographic primitives, pseudorandom generator, one-way function, randomizing polynomials

AMS subject classifications. 94A60, 68P25, 68Q15

DOI. 10.1137/S0097539705446950

1. Introduction.

The efficiency of cryptographic primitives is of both theoretical and practical interest. In this work, we consider the question of minimizing the *parallel time-complexity* of basic cryptographic primitives such as one-way functions (OWFs) and pseudorandom generators (PRGs) [11, 52]. Taking this question to an extreme, it is natural to ask if there are instances of these primitives that can be computed in *constant* parallel time. Specifically, the following fundamental question was posed in several previous works (e.g., [32, 22, 16, 41, 43]):

Are there one-way functions, or even pseudorandom generators, in NC^0?

Recall that NC^0 is the class of functions that can be computed by (a uniform family of) constant-depth circuits with bounded fan-in. In an NC^0 function each bit of the output depends on a constant number of input bits. We refer to this constant as the *output locality* of the function and denote by NC^0_c the class of NC^0 functions with locality c.

The above question is qualitatively interesting, since one might be tempted to conjecture that cryptographic hardness requires some output bits to depend on many

*Received by the editors February 2, 2005; accepted for publication (in revised form) September 8, 2005; published electronically December 15, 2006. A preliminary version of this paper appeared in *Proceedings of the* 45*th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2004. This research was supported by grant 36/03 from the Israel Science Foundation.

http://www.siam.org/journals/sicomp/36-4/44695.html

†Computer Science Department, Technion–Israel Institute of Technology, Haifa 32000, Israel (abenny@cs.technion.ac.il, yuvali@cs.technion.ac.il, eyalk@cs.technion.ac.il).

input bits. Indeed, this view is advocated by Cryan and Miltersen [16], whereas Goldreich [22] takes an opposite view and suggests a concrete candidate for OWFs in NC^0. However, despite previous efforts, there has been no convincing theoretical evidence supporting either a positive or a negative resolution of this question.

1.1. Previous work. Linial, Mansour, and Nisan show that pseudorandom *functions* cannot be computed even in AC^0 [42]. However, no such impossibility result is known for PRGs. The existence of PRGs in NC^0 has been recently studied in [16, 43]. Cryan and Miltersen [16] observe that there is no PRG in NC^0_2 and prove that there is no PRG in NC^0_3 achieving a superlinear stretch, namely, one that stretches n bits to $n + \omega(n)$ bits.[1] Mossel, Shpilka, and Trevisan [43] extend this impossibility to NC^0_4. Viola [50] shows that a PRG in AC^0 with superlinear stretch cannot be obtained from an OWF via nonadaptive black-box constructions. Negative results for other restricted computation models appear in [20, 54].

On the positive side, Impagliazzo and Naor [36] construct a (sublinear-stretch) PRG in AC^0, relying on an intractability assumption related to the subset-sum problem. PRG candidates in NC^1 (or even TC^0) are more abundant and can be based on a variety of standard cryptographic assumptions including ones related to the intractability of factoring [39, 44], discrete logarithms [11, 52, 44], and lattice problems [2, 33] (see Remark 6.6).[2]

Unlike the case of pseudorandom generators, the question of OWFs in NC^0 is relatively unexplored. The impossibility of OWFs in NC^0_2 follows from the easiness of 2-SAT [22, 16]. Håstad [32] constructs a family of permutations in NC^0 whose inverses are P-hard to compute. Cryan and Miltersen [16], improving on [1], present a circuit family in NC^0_3 whose range decision problem is NP-complete. This, however, gives no evidence of cryptographic strength. Since any PRG is also an OWF, all PRG candidates cited above are also OWF candidates. (In fact, the one-wayness of an NC^1 function often serves as the underlying cryptographic *assumption*.) Finally, Goldreich [22] suggests a candidate OWF in NC^0, whose conjectured security does not follow from any well-known assumption.

1.2. Our results. As indicated above, the possibility of implementing most cryptographic primitives in NC^0 was left wide open. We present a positive answer to this basic question, showing that, surprisingly, many cryptographic tasks can be performed in constant parallel time.

Since the existence of cryptographic primitives implies that $P \neq NP$, we cannot expect unconditional results and have to rely on some unproven assumptions.[3] However, we avoid relying on *specific* intractability assumptions. Instead, we assume the existence of cryptographic primitives in a relatively "high" complexity class and transform them to the seemingly degenerate complexity class NC^0 without substantial loss of their cryptographic strength. These transformations are inherently nonblack-box, thus providing further evidence for the usefulness of nonblack-box techniques in cryptography.

[1] From here on, we use a crude classification of PRGs into ones having sublinear, linear, or superlinear additive stretch. Note that a PRG stretching its seed by just one bit can be invoked *in parallel* (on seeds of length n^ϵ) to yield a PRG stretching its seed by $n^{1-\epsilon}$ bits for an arbitrary $\epsilon > 0$.

[2] In some of these constructions it seems necessary to allow a *collection* of NC^1 PRGs and use polynomial-time preprocessing to pick (once and for all) a random instance from this collection. This is similar to the more standard notion of an OWF collection (cf. [23, section 2.4.2]). See Appendix A for further discussion of this slightly relaxed notion of PRGs.

[3] This is not the case for noncryptographic PRGs such as ϵ-biased generators, for which we do obtain unconditional results.

We now give a more detailed account of our results.

A general compiler. Our main result is that any OWF (resp., PRG) in a relatively high complexity class, containing uniform NC^1 and even $\oplus L/poly$, can be efficiently "compiled" into a corresponding OWF (resp., sublinear-stretch PRG) in NC^0_4. (The class $\oplus L/poly$ contains the classes $L/poly$ and NC^1 and is contained in NC^2. In a nonuniform setting it also contains the class $NL/poly$ [51].) The existence of OWFs and PRGs in this class is a mild assumption, implied in particular by most number-theoretic or algebraic intractability assumptions commonly used in cryptography. Hence, the existence of OWFs and sublinear-stretch PRGs in NC^0 follows from a variety of standard assumptions and is not affected by the potential weakness of a particular algebraic structure. A similar compiler can also be obtained for other cryptographic primitives including one-way permutations, encryption, signatures, commitment, and collision-resistant hashing.

It is important to note that the PRG produced by our compiler will generally have a sublinear additive stretch even if the original PRG has a large stretch. However, one cannot do much better when insisting on an NC^0_4 PRG, as there is no PRG with superlinear stretch in NC^0_4 [43].

OWFs with optimal locality. The above results leave a small gap between the possibility of cryptography in NC^0_4 and the known impossibility of implementing even OWFs in NC^0_2. We partially close this gap by providing positive evidence for the existence of OWFs in NC^0_3. In particular, we construct such OWFs based on the intractability of decoding a random linear code.

Noncryptographic generators. Our techniques can also be applied to obtain unconditional constructions of noncryptographic PRGs. In particular, building on an ϵ-biased generator in NC^0_5 constructed by Mossel, Shpilka, and Trevisan [43], we obtain a linear-stretch ϵ-biased generator in NC^0_3. This generator has optimal locality, answering an open question posed in [43]. It is also essentially optimal with respect to stretch, since locality 3 does not allow for a superlinear stretch [16]. Our techniques also apply to other types of noncryptographic PRGs such as generators for space-bounded computation [6, 45], yielding such generators (with sublinear stretch) in NC^0_3.

1.3. Organization. In section 2 we provide an overview of our techniques, which evolve around the notion of "randomized encoding" introduced in this work. Following some preliminaries (section 3), in section 4 we formally define our notion of randomized encoding and discuss some of its variants, properties, and constructions. We then apply randomized encodings to obtain NC^0 implementations of different primitives: OWFs (section 5), cryptographic and noncryptographic PRGs (section 6), and other cryptographic primitives (section 7). In section 8 we construct OWFs with optimal locality based on specific intractability assumptions. We conclude in section 9 with some further research directions and open problems. We also call the reader's attention to Appendix A which discusses *collections* of cryptographic primitives and how they fit in the context of the current work.

2. Overview of techniques. Our key observation is that instead of computing a given "cryptographic" function $f(x)$, it might suffice to compute a function $\hat{f}(x,r)$ having the following relation to f:

1. For every fixed input x and a uniformly random choice of r, the output distribution $\hat{f}(x,r)$ forms a "randomized encoding" of $f(x)$, from which $f(x)$ can be decoded. That is, if $f(x) \neq f(x')$, then the random variables $\hat{f}(x,r)$ and $\hat{f}(x',r')$, induced by a uniform choice of r, r', should have disjoint supports.

2. The distribution of this randomized encoding depends only on the encoded value $f(x)$ and does not further depend on x. That is, if $f(x) = f(x')$, then the random variables $\hat{f}(x,r)$ and $\hat{f}(x',r')$ should be identically distributed. Furthermore, we require that the randomized encoding of an output value y be efficiently samplable given y. Intuitively, this means that the output distribution of $\hat{f}$ on input x reveals no information about x except what follows from $f(x)$.

Each of these requirements alone can be satisfied by a trivial function $\hat{f}$ (e.g., $\hat{f}(x,r) = x$ and $\hat{f}(x,r) = 0$, respectively). However, the combination of the two requirements can be viewed as a nontrivial natural relaxation of the usual notion of computing. In a sense, the function $\hat{f}$ defines an "information-theoretically equivalent" representation of f. In the following, we refer to $\hat{f}$ as a *randomized encoding* of f.

For this approach to be useful in our context, two conditions should be met. First, we need to argue that a randomized encoding $\hat{f}$ can be *securely* used as a substitute for f. Second, we hope that this relaxation is sufficiently *liberal*, in the sense that it allows us to efficiently encode relatively complex functions f by functions $\hat{f}$ in NC^0. These two issues are addressed in the following subsections.

2.1. Security of randomized encodings. To illustrate how a randomized encoding $\hat{f}$ can inherit the security features of f, we consider the case where f is an OWF. We argue that the hardness of inverting $\hat{f}$ reduces to the hardness of inverting f. Indeed, a successful algorithm A for inverting $\hat{f}$ can be used to successfully invert f as follows: Given an output y of f, apply the efficient sampling algorithm guaranteed by requirement 2 to obtain a random encoding $\hat{y}$ of y. Then, use A to obtain a preimage (x,r) of $\hat{y}$ under $\hat{f}$, and output x. It follows from requirement 1 that x is indeed a preimage of y under f. Moreover, if y is the image of a uniformly random x, then $\hat{y}$ is the image of a uniformly random pair (x,r). Hence, the success probability of inverting f is the same as that of inverting $\hat{f}$.

The above argument can tolerate some relaxations in the notion of randomized encoding. In particular, one can relax the second requirement to allow a small statistical variation of the output distribution. On the other hand, to maintain the security of other cryptographic primitives, it may be required to further strengthen this notion. For instance, when f is a PRG, the above requirements do not guarantee that the output of $\hat{f}$ is pseudorandom, or even that its output is longer than its input. However, by imposing suitable "regularity" requirements on the output encoding defined by $\hat{f}$, it can be guaranteed that if f is a PRG, then so is $\hat{f}$. Thus, different security requirements suggest different variations of the above notion of randomized encoding.

2.2. Complexity of randomized encodings. It remains to address the second issue: Can we encode a complex function f by an NC^0 function $\hat{f}$? Our best solutions to this problem rely on the machinery of *randomizing polynomials,* described below. But first, we outline a simple alternative approach[4] based on Barrington's theorem [7], combined with a randomization technique of Kilian [40].

Suppose f is a boolean function in NC^1. (Nonboolean functions are handled by repeating the following procedure for each bit of the output.) By Barrington's theorem, evaluating $f(x)$, for such a function f, reduces to computing an iterated product of polynomially many elements $s_1, \ldots, s_m$ from the symmetric group S_5, where each s_i is determined by a single bit of x (i.e., for every i there exists j such

[4]In fact, a modified version of this approach has been applied for constructing randomizing polynomials in [15].

that s_i is a function of x_j). Now, let $\hat{f}(x,r) = (s_1 r_1,\ r_1^{-1} s_2 r_2, \ldots,\ r_{m-2}^{-1} s_{m-1} r_{m-1},\ r_{m-1}^{-1} s_m)$, where the random inputs r_i are picked uniformly and independently from S_5. It is not hard to verify that the output $(t_1, \ldots, t_m)$ of $\hat{f}$ is random subject to the constraint that $t_1 t_2 \cdots t_m = s_1 s_2 \cdots s_m$, where the latter product is in one-to-one correspondence to $f(x)$. It follows that $\hat{f}$ is a randomized encoding of f. Moreover, $\hat{f}$ has constant locality when viewed as a function over the alphabet S_5, and thus yields the qualitative result we are after.

However, the above construction falls short of providing a randomized encoding in NC^0, since it is impossible to sample a uniform element of S_5 in NC^0 (even up to a negligible statistical distance).[5] Also, this $\hat{f}$ does not satisfy the extra "regularity" properties required by more "sensitive" primitives such as PRGs or one-way permutations. The solutions presented next avoid these disadvantages and, at the same time, apply to a higher complexity class than NC^1 and achieve a very small constant locality.

Randomizing polynomials. The concept of randomizing polynomials was introduced by Ishai and Kushilevitz [37] as a representation of functions by vectors of low-degree multivariate polynomials. (Interestingly, this concept was motivated by questions in the area of *information-theoretic* secure multiparty computation, which seems unrelated to the current context.) Randomizing polynomials capture the above encoding question within an algebraic framework. Specifically, a representation of $f(x)$ by randomizing polynomials is a randomized encoding $\hat{f}(x,r)$ as defined above, in which x and r are viewed as vectors over a finite field $\mathcal{F}$ and the outputs of $\hat{f}$ as multivariate polynomials in the variables x and r. In this work, we will always let $\mathcal{F} = \mathrm{GF}(2)$.

The most crucial parameter of a randomizing polynomial representation is its algebraic *degree*, defined as the maximal (total) degree of the outputs (i.e., the output multivariate polynomials) as a function of the input variables in x and r. (Note that both x and r count toward the degree.) Quite surprisingly, it is shown in [37, 38] that every boolean function $f : \{0,1\}^n \to \{0,1\}$ admits a representation by *degree*-3 randomizing polynomials whose number of inputs and outputs is at most *quadratic* in its branching program size.[6] (Moreover, this degree bound is tight in the sense that most boolean functions do not admit a degree-2 representation.) Note that a representation of a nonboolean function can be obtained by concatenating representations of its output bits, using independent blocks of random inputs. This concatenation leaves the degree unchanged.

The above positive result implies that functions whose output bits can be computed in the complexity class $\oplus$L/*poly* admit an efficient representation by degree-3 randomizing polynomials. This also holds if one requires the most stringent notion of representation required by our applications. We note, however, that different constructions from the literature [37, 38, 15] are incomparable in terms of their exact efficiency and the security-preserving features they satisfy. Hence, different constructions may be suitable for different applications. These issues are discussed in section 4.

Degree vs. locality. Combining our general methodology with the above results on randomizing polynomials already brings us close to our goal, as it enables "degree-3 cryptography." Proceeding from this point, we show that any function $f : \{0,1\}^n \to \{0,1\}^m$ of algebraic degree d admits an efficient randomized encoding $\hat{f}$ of (degree d and) locality $d+1$. That is, each output bit of $\hat{f}$ can be computed by a degree-d polynomial over GF(2) depending on at most $d+1$ inputs and random inputs.

[5] Barrington's theorem generalizes to apply over arbitrary nonsolvable groups. Unfortunately, there are no such groups whose order is a power of two.

[6] By default, the notion of "branching programs" refers here to mod-2 branching programs, which output the parity of the number of accepting paths. See section 3.

Combined with the previous results, this allows us to make the final step from degree 3 to locality 4.

3. Preliminaries.

Probability notation. Let U_n denote a random variable that is uniformly distributed over $\{0,1\}^n$. Different occurrences of U_n in the same statement refer to the same random variable (rather than independent ones). If X is a probability distribution, we write $x \leftarrow X$ to indicate that x is a sample taken from X. If S is a set, we write $x \in_R S$ to indicate that x is uniformly selected from S. The *statistical distance* between discrete probability distributions X and Y is defined as $\|X - Y\| \stackrel{\text{def}}{=} \frac{1}{2}\sum_z |\Pr[X = z] - \Pr[Y = z]|$. Equivalently, the statistical distance between X and Y may be defined as the maximum, over all boolean functions T, of the *distinguishing advantage* $|\Pr[T(X) = 1] - \Pr[T(Y) = 1]|$. A function $\varepsilon(\cdot)$ is said to be *negligible* if $\varepsilon(n) < n^{-c}$ for any $c > 0$ and sufficiently large n. For two distribution ensembles $X = \{X_n\}$ and $Y = \{Y_n\}$, we write $X \equiv Y$ if X_n and Y_n are identically distributed, and $X \stackrel{s}{\approx} Y$ if the two ensembles are *statistically indistinguishable*; namely, $\|X_n - Y_n\|$ is negligible in n.

We will rely on the following standard properties of statistical distance.

FACT 3.1. *For all distributions X, Y, Z we have $\|X - Z\| \le \|X - Y\| + \|Y - Z\|$.*

FACT 3.2. *For all distributions X, X', Y, Y' we have $\|(X \times X') - (Y \times Y')\| \le \|X - Y\| + \|X' - Y'\|$, where $A \times B$ denotes the product distribution of A, B, i.e., the joint distribution of independent samples from A and B.*

FACT 3.3. *For all distributions X, Y and every function f we have $\|f(X) - f(Y)\| \le \|X - Y\|$.*

FACT 3.4. *Let $\{X_z\}_{z\in\mathcal{Z}}$, $\{Y_z\}_{z\in\mathcal{Z}}$ be distribution ensembles. Then, for every distribution Z over $\mathcal{Z}$, we have $\|(Z, X_Z) - (Z, Y_Z)\| = E_{z\leftarrow Z}[\|X_z - Y_z\|]$. In particular, if $\|X_z - Y_z\| \le \varepsilon$ for every $z \in \mathcal{Z}$, then $\|(Z, X_Z) - (Z, Y_Z)\| \le \varepsilon$.*

Branching programs. A branching program (BP) is defined by a tuple $BP = (G, \phi, s, t)$, where $G = (V, E)$ is a directed acyclic graph, ϕ is a labeling function assigning each edge either a positive literal x_i, a negative literal $\bar{x}_i$, or the constant 1, and s, t are two distinguished nodes of G. The *size* of BP is the number of nodes in G. Each input assignment $w = (w_1, \ldots, w_n)$ naturally induces an unlabeled subgraph G_w, whose edges include all $e \in E$ such that $\phi(e)$ is satisfied by w (e.g., an edge labeled x_i is satisfied by w if $w_i = 1$). BPs may be assigned different semantics: In a *nondeterministic* BP, an input w is accepted if G_w contains at least one path from s to t; in a (*counting*) *mod-p* BP, the BP computes the number of paths from s to t modulo p. In this work, we will mostly be interested in mod-2 BPs. An example of a mod-2 BP is given in Figure 3.1.

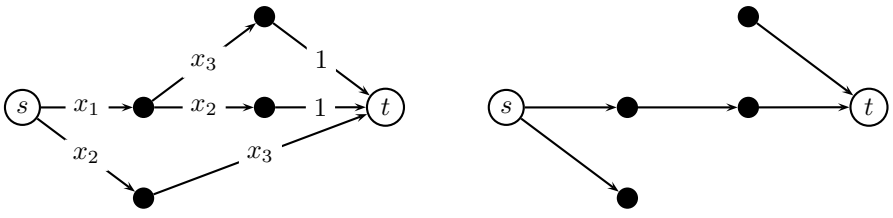


FIG. 3.1. *A mod-2 branching program computing the majority of three bits (left side), along with the graph G_{110} induced by the assignment* 110 *(right side).*

Function families and representations. We associate with a function $f : \{0,1\}^* \to \{0,1\}^*$ a function family $\{f_n\}_{n\in\mathbb{N}}$, where f_n is the restriction of f to n-bit inputs. We assume all functions to be length regular; namely, their output length depends only on their input length. Hence, we may write $f_n : \{0,1\}^n \to \{0,1\}^{l(n)}$. We will represent functions f by families of circuits, branching programs, or vectors of polynomials (where each polynomial is represented by a formal sum of monomials). Whenever f is taken from a uniform class, we assume that its representation is uniform as well. That is, the representation of f_n is generated in time poly(n) and in particular is of polynomial size. We will often abuse notation and write f instead of f_n even when referring to a function on n bits.

Locality and degree. We say that f is c-local if each of its output bits depends on at most c input bits.[7] For a constant c, the nonuniform class NC^0_c includes all c-local functions. We will sometimes view the binary alphabet as the finite field $\mathcal{F} = GF(2)$, and say that a function $f : \mathcal{F}^n \to \mathcal{F}^{l(n)}$ has degree d if each of its outputs can be expressed as a multivariate polynomial of degree (at most) d in the inputs.

Complexity classes. For brevity, we use the (somewhat nonstandard) convention that all complexity classes are polynomial-time uniform unless otherwise stated. For instance, NC^0 refers to the class of functions admitting uniform NC^0 circuits, whereas *nonuniform* NC^0 refers to the class of functions admitting nonuniform NC^0 circuits. We let NL/*poly* (resp., $\oplus$L/*poly*) denote the class of boolean functions computed by a polynomial-time uniform family of nondeterministic (resp., modulo-2) BPs. (Recall that in a uniform family of circuits or branching programs computing f, it should be possible to generate the circuit or branching program computing f_n in time poly(n).) Equivalently, the class NL/*poly* (resp., $\oplus$L/*poly*) is the class of functions computed by NL (resp., $\oplus$L) Turing machines taking a uniform advice. (The class $\oplus$L/*poly* contains the classes L/*poly* and NC^1 and is contained in NC^2. In a nonuniform setting it also contains the class NL/*poly* [51].) We extend boolean complexity classes, such as NL/*poly* and $\oplus$L/*poly*, to include nonboolean functions by letting the representation include $l(n)$ branching programs, one for each output. Uniformity requires that the $l(n)$ branching programs all be generated in time poly(n).

4. Randomized encoding of functions.

In this section we formally introduce our notion of randomized encoding. In section 4.1 we introduce several variants of randomized encoding and in section 4.2 we prove some of their useful properties. Finally, in section 4.3 we construct NC^0_4 encodings for branching programs, building on [37, 38].

4.1. Definitions.

We start by defining a randomized encoding of a finite function f. This definition will be later extended to a (uniform) family of functions.

DEFINITION 4.1 (randomized encoding). *Let $f : \{0,1\}^n \to \{0,1\}^l$ be a function. We say that a function $\hat{f} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$ is a δ-correct, ε-private* randomized encoding *of f if it satisfies the following:*

- δ-correctness. *There exists a deterministic*[8] *algorithm C, called a* decoder, *such that for every input $x \in \{0,1\}^n$, $\Pr[C(\hat{f}(x, U_m)) \neq f(x)] \leq \delta$.*

[7] A boolean function depends on the ith input bit if there exists an assignment such that flipping the ith input bit changes the value of the function.

[8] We restrict the decoder to be deterministic for simplicity. This restriction does not compromise generality, in the sense that one can transform a randomized decoder to a deterministic one by incorporating the coins of the former in the encoding itself.

- ε-privacy. *There exists a randomized algorithm S, called a* simulator*, such that for every $x \in \{0,1\}^n$, $\|S(f(x)) - \hat{f}(x, U_m)\| \leq \varepsilon$.*

We refer to the second input of $\hat{f}$ as its random input *and to m and s as the* randomness complexity *and* output complexity *of $\hat{f}$, respectively.*

Note that the above definition refers only to the *information* about x revealed by $\hat{f}(x,r)$ and does not consider the complexity of the decoder and the simulator. Intuitively, the function $\hat{f}$ defines an "information-theoretically equivalent" representation of f. The correctness property guarantees that from $\hat{y} = \hat{f}(x,r)$ it is possible to reconstruct $f(x)$ (with high probability), whereas the privacy property guarantees that by seeing $\hat{y}$ one cannot learn too much about x (in addition to $f(x)$). The encoding is δ-correct (resp., ε-private) if it is correct (resp., private) up to an "error" of δ (resp., ε). This is illustrated by the next example.

Example 4.2. Consider the function $f(x_1, \ldots, x_n) = x_1 \vee x_2 \vee \cdots \vee x_n$. We define a randomized encoding $\hat{f} : \{0,1\}^n \times \{0,1\}^{ns} \to \{0,1\}^s$ by $\hat{f}(x,r) = (\sum_{i=1}^n x_i r_{i,1}, \ldots, \sum_{i=1}^n x_i r_{i,s})$, where $x = (x_1, \ldots, x_n)$, $r = (r_{i,j})$ for $1 \leq i \leq n, 1 \leq j \leq s$, and addition is over GF(2). First, observe that the distribution of $\hat{f}(x, U_{ns})$ depends only on the value of $f(x)$. Specifically, let S be a simulator that outputs an s-tuple of zeros if $f(x) = 0$, and a uniformly chosen string in $\{0,1\}^s$ if $f(x) = 1$. It is easy to verify that $S(f(x))$ is distributed the same as $\hat{f}(x, U_{ns})$ for any $x \in \{0,1\}^n$. It follows that this randomized encoding is 0-private. Also, one can obtain an efficient decoder C that, given a sample y from the distribution $\hat{f}(x, U_{ns})$, outputs 0 if $y = 0^s$ and otherwise outputs 1. Such an algorithm will err with probability 2^{-s}; thus $\hat{f}$ is 2^{-s}-correct.

On uniform randomized encodings. The above definition naturally extends to functions $f : \{0,1\}^* \to \{0,1\}^*$. In this case, the parameters $l, m, s, \delta, \varepsilon$ are all viewed as functions of the input length n, and the algorithms C, S receive 1^n as an additional input. In our default uniform setting, we require that $\hat{f}_n$, the encoding of f_n, be computable in time poly(n) (given $x \in \{0,1\}^n$ and $r \in \{0,1\}^{m(n)}$). Thus, in this setting both $m(n)$ and $s(n)$ are polynomially bounded. We also require both the decoder and the simulator to be efficient. (This is not needed by some of the applications but is a feature of our constructions.) We formalize these requirements below.

DEFINITION 4.3 (uniform randomized encoding). *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function and $l(n)$ an output length function such that $|f(x)| = l(|x|)$ for every $x \in \{0,1\}^*$. We say that $\hat{f} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ is a $\delta(n)$-correct $\epsilon(n)$-private uniform randomized encoding of f if the following hold:*

- Length regularity. *There exist polynomially bounded and efficiently computable length functions $m(n), s(n)$ such that for every $x \in \{0,1\}^n$ and $r \in \{0,1\}^{m(n)}$, we have $|\hat{f}(x,r)| = s(n)$.*
- Efficient evaluation. *There exists a polynomial-time* evaluation algorithm *that, given $x \in \{0,1\}^*$ and $r \in \{0,1\}^{m(|x|)}$, outputs $\hat{f}(x,r)$.*
- δ-correctness. *There exists a polynomial-time* decoder *C, such that for every $x \in \{0,1\}^n$ we have $\Pr[C(1^n, \hat{f}(x, U_{m(n)})) \neq f(x)] \leq \delta(n)$.*
- ε-privacy. *There exists a probabilistic polynomial-time* simulator *S, such that for every $x \in \{0,1\}^n$ we have $\|S(1^n, f(x)) - \hat{f}(x, U_{m(n)})\| \leq \varepsilon(n)$.*

When saying that a uniform encoding $\hat{f}$ is in a (uniform) circuit complexity class, we mean that its evaluation algorithm can be implemented by circuits in this class. For instance, we say that $\hat{f}$ is in NC^0_d if there exists a polynomial-time circuit generator G such that $G(1^n)$ outputs a d-local circuit computing $\hat{f}(x,r)$ on all $x \in \{0,1\}^n$ and $r \in \{0,1\}^{m(n)}$.

From here on, a randomized encoding of an efficiently computable function is assumed to be uniform by default. Moreover, we will freely extend the above definition to apply to a uniform collection of functions $\mathcal{F} = \{f_z\}_{z\in Z}$, for some index set $Z \subseteq \{0,1\}^*$. In such a case it is required that the encoded collection $\hat{\mathcal{F}} = \{\hat{f}_z\}_{z\in Z}$ is also uniform, in the sense that the same efficient evaluation algorithm, decoder, and simulator should apply to the entire collection when given z as an additional input. (See Appendix A for a more detailed discussion of *collections* of functions and cryptographic primitives.) Finally, for the sake of simplicity we will sometimes formulate our definitions, claims, and proofs using finite functions, under the implicit understanding that they naturally extend to the uniform setting.

We move on to discuss some variants of the basic definition. Correctness (resp., privacy) can be either *perfect*, when $\delta = 0$ (resp., $\varepsilon = 0$), or *statistical*, when $\delta(n)$ (resp., $\varepsilon(n)$) is negligible. In fact, we can further relax privacy to hold only against efficient algorithms, i.e., to require that for every $x \in \{0,1\}^n$, every polynomial-time algorithm A distinguishes between the distributions $S(f(x))$ and $\hat{f}(x, U_m)$ with no more than negligible advantage. Such an encoding is referred to as *computationally* private and it suffices for the purpose of many applications discussed in this paper. (Further details and additional applications appear in [4].) However, while for some of the primitives (such as OWFs) computational privacy and statistical correctness will do, others (such as PRGs or one-way permutations) require even stronger properties than perfect correctness and privacy. One such additional property is that the simulator S, when invoked on a uniformly random string from $\{0,1\}^l$ (the output domain of f), will output a uniformly random string from $\{0,1\}^s$ (the output domain of $\hat{f}$). We call this property *balance*. Note that the balance requirement does not impose any uniformity condition on the output of f, which in fact can be concentrated on a strict subset of $\{0,1\}^l$.

DEFINITION 4.4 (balanced randomized encoding). *A randomized encoding $\hat{f} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$ of a function $f : \{0,1\}^n \to \{0,1\}^l$ is called* balanced *if it has a perfectly private simulator S such that $S(U_l) \equiv U_s$. We refer to S as a* balanced simulator.

A last useful property is a syntactic one: We sometimes want $\hat{f}$ to have the same additive stretch as f. Specifically, we say that $\hat{f}$ is *stretch-preserving* (with respect to f) if $s - (n + m) = l - n$ or, equivalently, $m = s - l$.

We are now ready to define our two main variants of randomized encoding.

DEFINITION 4.5 (statistical randomized encoding). *A* statistical randomized encoding *is a randomized encoding that is statistically correct and statistically private.*

DEFINITION 4.6 (perfect randomized encoding). *A* perfect randomized encoding *is a randomized encoding that is perfectly correct, perfectly private, balanced, and stretch-preserving.*

A combinatorial view of perfect encoding. To gain better understanding of the properties of perfect encoding, we take a closer look at the relation between a function and its encoding. Let $\hat{f} : \{0,1\}^{n+m} \to \{0,1\}^s$ be an encoding of $f : \{0,1\}^n \to \{0,1\}^l$. The following description addresses the simpler case where f is onto. Every $x \in \{0,1\}^n$ is mapped to some $y \in \{0,1\}^l$ by f, and to a 2^m-size multiset $\{\hat{f}(x,r) | r \in \{0,1\}^m\}$ which is contained in $\{0,1\}^s$. Perfect privacy means that this multiset is common to all the x's that share the same image under f; thus we have a mapping from $y \in \{0,1\}^l$ to multisets in $\{0,1\}^s$ of size 2^m (such a mapping is defined by the perfect simulator). Perfect correctness means that these multisets are mutually disjoint. However, even perfect privacy and perfect correctness together do not promise that this mapping

covers all of $\{0,1\}^s$. The balance property guarantees that the multisets form a perfect tiling of $\{0,1\}^s$; moreover it promises that each element in these multisets has the same multiplicity. If the encoding is also stretch-preserving, then the multiplicity of each element must be 1, so that the multisets are actually sets. Hence, a perfect randomized encoding guarantees the existence of a perfect simulator S whose 2^l output distributions form a perfect tiling of the space $\{0,1\}^s$ by sets of size 2^m.

Remark 4.7 (a padding convention). We will sometimes view $\hat{f}$ as a function of a single input of length $n+m(n)$ (e.g., when using it as an OWF or a PRG). In this case, we require $m(\cdot)$ to be monotone nondecreasing, so that $n+m(n)$ uniquely determines n. We apply a standard padding technique for defining $\hat{f}$ on inputs whose length is not of the form $n+m(n)$. Specifically, if $n+m(n)+t<(n+1)+m(n+1)$, we define $\hat{f}'$ on inputs of length $n+m(n)+t$ by applying $\hat{f}_n$ on the first $n+m(n)$ bits and then appending the t additional input bits to the output of $\hat{f}_n$. This convention respects the security of cryptographic primitives such as OWFs, PRGs, and collision-resistant hashing, provided that $m(n)$ is efficiently computable and is sufficiently dense (both of which are guaranteed by a uniform encoding). That is, if the unpadded function $\hat{f}$ is secure with respect to its partial domain, then its padded version $\hat{f}'$ is secure in the standard sense, i.e., over the domain of all strings.[9] (See a proof for the case of OWFs in [23, Proposition 2.2.3].) Note that the padded function $\hat{f}'$ has the same locality and degree as $\hat{f}$. Moreover, $\hat{f}'$ also preserves syntactic properties of $\hat{f}$; for example, it preserves the stretch of $\hat{f}$ and if $\hat{f}$ is a permutation, then so is $\hat{f}'$. Thus, it is enough to prove our results for the partially defined unpadded function $\hat{f}$ and keep the above conventions implicit.

Finally, we define two complexity classes that capture the power of randomized encodings in NC^0.

DEFINITION 4.8 (the classes $\mathcal{SRENC}$, $\mathcal{PRENC}$). *The class $\mathcal{SRENC}$ (resp., $\mathcal{PRENC}$) is the class of functions $f:\{0,1\}^*\to\{0,1\}^*$ admitting a statistical (resp., perfect) uniform randomized encoding in* NC^0.

4.2. Basic properties. We now put forward some useful properties of randomized encodings. We first argue that an encoding of a nonboolean function can be obtained by concatenating encodings of its output bits, using an independent random input for each bit. The resulting encoding inherits all the features of the concatenated encodings and, in particular, preserves their perfectness.

LEMMA 4.9 (concatenation). *Let $f_i:\{0,1\}^n\to\{0,1\}$, $1\le i\le l$, be the boolean functions computing the output bits of a function $f:\{0,1\}^n\to\{0,1\}^l$. If $\hat{f}_i:\{0,1\}^n\times\{0,1\}^{m_i}\to\{0,1\}^{s_i}$ is a δ-correct ε-private encoding of f_i, then the function $\hat{f}:\{0,1\}^n\times\{0,1\}^{m_1+\cdots+m_l}\to\{0,1\}^{s_1+\cdots+s_l}$ defined by $\hat{f}(x,(r_1,\ldots,r_l))\stackrel{\mathrm{def}}{=}(\hat{f}_1(x,r_1),\ldots,\hat{f}_l(x,r_l))$ is a (δl)-correct, (εl)-private encoding of f. Moreover, if all $\hat{f}_i$ are perfect, then so is $\hat{f}$.*

Proof. We start with correctness. Let C_i be a δ-correct decoder for $\hat{f}_i$. Define a decoder C for $\hat{f}$ by $C(\hat{y}_1,\ldots,\hat{y}_l)=(C_1(\hat{y}_1),\ldots,C_l(\hat{y}_l))$. By a union bound argument, C is a (δl)-correct decoder for $\hat{f}$ as required.

We turn to analyze privacy. Let S_i be an ε-private simulator for $\hat{f}_i$. An (εl)-private simulator S for $\hat{f}$ can be naturally defined by $S(y)=(S_1(y_1),\ldots,S_l(y_l))$,

[9] This can be generally explained by viewing each slice of the padded function $\hat{f}'$ (i.e., its restriction to inputs of some fixed length) as a *perfect* randomized encoding of a corresponding slice of $\hat{f}$.

where the invocations of the simulators S_i use independent coins. Indeed, for every $x \in \{0,1\}^n$ we have

$$\begin{aligned}\|S(f(x)) - \hat{f}(x,(U_{m_1},\ldots,U_{m_l}))\| &= \|(S_i(y_i))_{i=1}^l - (\hat{f}_i(x,U_{m_i}))_{i=1}^l\| \\ &\le \sum_{i=1}^{l} \|S_i(y_i) - \hat{f}_i(x,U_{m_i})\| \\ &\le \varepsilon l,\end{aligned}$$

where $y = f(x)$. The first inequality follows from Fact 3.2 and the independence of the randomness used for different i, and the second from the ε-privacy of each S_i.

Note that the simulator S described above is balanced if all S_i are balanced. Moreover, if all $\hat{f}_i$ are stretch-preserving, i.e., $s_i - 1 = m_i$, then we have $\sum_{i=1}^l s_i - l = \sum_{i=1}^l m_i$ and hence $\hat{f}$ is also stretch-preserving. It follows that if all $\hat{f}_i$ are perfect, then so is $\hat{f}$. □

We state the following uniform version of Lemma 4.9, whose proof is implicit in the above.

LEMMA 4.10 (concatenation: uniform version). *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function, viewed as a uniform collection of functions $\mathcal{F} = \{f_{n,i}\}_{n\in\mathbb{N},1\le i\le l(n)}$; that is, $f_{n,i}(x)$ outputs the ith bit of $f(x)$ for all $x \in \{0,1\}^n$. Suppose that $\hat{\mathcal{F}} = \{\hat{f}_{n,i}\}_{n\in\mathbb{N},1\le i\le l(n)}$ is a perfect (resp., statistical) uniform randomized encoding of $\mathcal{F}$. Then, the function $\hat{f} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ defined by $\hat{f}(x,(r_1,\ldots,r_{l(|x|)})) \stackrel{\text{def}}{=} (\hat{f}_{|x|,1}(x,r_1),\ldots,\hat{f}_{|x|,l(|x|)}(x,r_{l(|x|)}))$ is a perfect (resp., statistical) uniform randomized encoding of f.*

Another useful feature of randomized encodings is the following intuitive composition property: Suppose we encode f by g and then view g as a deterministic function and encode it again. Then, the resulting function (parsed appropriately) is a randomized encoding of f. Again, the resulting encoding inherits the perfectness of the encodings from which it is composed.

LEMMA 4.11 (composition). *Let $g(x,r_g)$ be a δ_g-correct, ε_g-private encoding of $f(x)$ and let $h((x,r_g),r_h)$ be a δ_h-correct, ε_h-private encoding of $g((x,r_g))$ (viewed as a single-argument function). Then, the function $\hat{f}(x,(r_g,r_h)) \stackrel{\text{def}}{=} h((x,r_g),r_h)$ is a $(\delta_g+\delta_h)$-correct, $(\varepsilon_g+\varepsilon_h)$-private encoding of f. Moreover, if g,h are perfect (resp., statistical) uniform randomized encodings, then so is $\hat{f}$.*

Proof. We start with correctness. Let C_g be a δ_g-correct decoder for g and C_h a δ_h-correct decoder for h. Define a decoder C for $\hat{f}$ by $C(\hat{y}) = C_g(C_h(\hat{y}))$. The decoder C errs only if either C_h or C_g errs. Thus, by the union bound we have for every x

$$\begin{aligned}\Pr_{r_g,r_h}[C(\hat{f}(x,(r_g,r_h))) \ne f(x)] &\le \Pr_{r_g,r_h}[C_h(h((x,r_g),r_h)) \ne g(x,r_g)] \\ &\quad + \Pr_{r_g}[C_g(g(x,r_g)) \ne f(x)] \\ &\le \delta_h + \delta_g,\end{aligned}$$

as required.

Privacy is argued similarly. Let S_g be an ε_g-private simulator for g and let S_h be an ε_h-private simulator for h. We define a simulator S for $\hat{f}$ by $S(y) = S_h(S_g(y))$.

Letting m_g, m_h denote the randomness complexity of g, h, respectively, we have for every x

$$\begin{aligned}\|S(f(x)) - \hat{f}(x,(U_{m_g},U_{m_h}))\| &= \|S_h(S_g(f(x))) - h((x,U_{m_g}),U_{m_h})\| \\ &\le \|S_h(S_g(f(x))) - S_h(g(x,U_{m_g}))\| \\ &\quad + \|S_h(g(x,U_{m_h})) - h((x,U_{m_g}),U_{m_h})\| \\ &\le \varepsilon_g + \varepsilon_h,\end{aligned}$$

where the first inequality follows from the triangle inequality (Fact 3.1), and the second from Facts 3.3 and 3.4.

It is easy to verify that if S_g and S_h are balanced, then so is S. Moreover, if g preserves the additive stretch of f and h preserves the additive stretch of g, then h (hence also $\hat{f}$) preserves the additive stretch of f. Thus $\hat{f}$ is perfect if both g, h are perfect. All of the above naturally carries over to the uniform setting, from which the last part of the lemma follows. □

Finally, we prove two useful features of a *perfect* encoding.

LEMMA 4.12 (unique randomness). *Suppose $\hat{f}$ is a perfect randomized encoding of f. Then,* (a) *$\hat{f}$ satisfies the following* unique randomness *property: For any input x, the function $\hat{f}(x,\cdot)$ is injective; namely, there are no distinct r, r' such that $\hat{f}(x,r) = \hat{f}(x,r')$. Moreover,* (b) *if f is a permutation, then so is $\hat{f}$.*

Proof. Let $f : \{0,1\}^n \to \{0,1\}^l$ and $\hat{f} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$. To prove part (a), assume toward a contradiction that $\hat{f}$ does not satisfy the unique randomness property. Then, by perfect privacy, we have $|\mathrm{Im}(\hat{f})| < |\mathrm{Im}(f)| \cdot 2^m$. On the other hand, letting S be a balanced simulator, we have

$$\begin{aligned}|\mathrm{Im}(\hat{f})| \cdot 2^{-s} &= \Pr_{y \leftarrow U_l}[S(y) \in \mathrm{Im}(\hat{f})] \\ &\ge \Pr_{y \leftarrow U_l}[S(y) \in \mathrm{Im}(\hat{f}) | y \in \mathrm{Im}(f)] \cdot \Pr_{y \leftarrow U_l}[y \in \mathrm{Im}(f)] \\ &= 1 \cdot \frac{|\mathrm{Im}(f)|}{2^l},\end{aligned}$$

where the last equality follows from perfect privacy. Since g is stretch-preserving ($s - l = m$), we get from the above that $|\mathrm{Im}(\hat{f})| \ge |\mathrm{Im}(f)| \cdot 2^m$ and derive a contradiction.

If f is a permutation, then $n = l$ and since $\hat{f}$ is stretch-preserving, we can write $\hat{f} : \{0,1\}^s \to \{0,1\}^s$. Thus, to prove part (b), it is enough to prove that $\hat{f}$ is injective. Suppose that $\hat{f}(x,r) = \hat{f}(x',r')$. Then, since f is injective and $\hat{f}$ is perfectly correct, it follows that $x = x'$; hence, by part (a), $r = r'$ and the proof follows. □

4.3. Constructions. In this section we construct randomized encodings in NC^0. We first review a construction from [38] of degree-3 randomizing polynomials based on mod-2 branching programs and analyze some of its properties. Next, we introduce a general locality reduction technique, allowing us to transform a degree-d encoding to a $(d+1)$-local encoding. Finally, we discuss extensions to other types of BPs.

Degree-3 randomizing polynomials from mod-2 branching programs [38]. Let $BP = (G, \phi, s, t)$ be a mod-2 BP of size ℓ, computing a boolean[10] function $f : \{0,1\}^n \to \{0,1\}$; that is, $f(x) = 1$ if and only if the number of paths from s to t in G_x equals 1

[10] The following construction generalizes naturally to a (counting) mod-p BP, computing a function $f : \{0,1\}^n \to Z_p$. In this work, however, we will be interested only in the case $p = 2$.

$$\begin{pmatrix} 1 & r_1^{(1)} & r_2^{(1)} & \cdot & r_{\ell-2}^{(1)} \\ 0 & 1 & \cdot & \cdot & \cdot \\ 0 & 0 & 1 & \cdot & \cdot \\ 0 & 0 & 0 & 1 & r_{\binom{\ell-1}{2}}^{(1)} \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} * & * & * & * & * \\ -1 & * & * & * & * \\ 0 & -1 & * & * & * \\ 0 & 0 & -1 & * & * \\ 0 & 0 & 0 & -1 & * \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & r_1^{(2)} \\ 0 & 1 & 0 & 0 & r_2^{(2)} \\ 0 & 0 & 1 & 0 & \cdot \\ 0 & 0 & 0 & 1 & r_{\ell-2}^{(2)} \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

FIG. 4.1. *The matrices $R_1(r^{(1)}), L(x), R_2(r^{(2)})$ (from left to right). The symbol $*$ represents a degree-*1 *polynomial in an input variable.*

modulo 2. Fix some topological ordering of the vertices of G, where the source vertex s is labeled 1 and the terminal vertex t is labeled ℓ. Let $A(x)$ be the $\ell \times \ell$ adjacency matrix of G_x viewed as a formal matrix whose entries are degree-1 polynomials in the input variables x. Specifically, the (i,j) entry of $A(x)$ contains the value of $\phi(i,j)$ on x if (i,j) is an edge in G, and 0 otherwise. (Hence, $A(x)$ contains the constant 0 on and below the main diagonal and degree-1 polynomials in the input variables above the main diagonal.) Define $L(x)$ as the submatrix of $A(x) - I$ obtained by deleting column s and row t (i.e., the first column and the last row). As before, each entry of $L(x)$ is a degree-1 polynomial in a single input variable x_i; moreover, $L(x)$ contains the constant -1 in each entry of its second diagonal (the one below the main diagonal) and the constant 0 below this diagonal. (See Figure 4.1.)

FACT 4.13 (see [38]). *$f(x) = \det(L(x))$, where the determinant is computed over* GF(2).

Proof sketch. Since G is acyclic, the number of $s-t$ paths in G_x mod 2 can be written as $(I + A(x) + A(x)^2 + \cdots + A(x)^\ell)_{s,t} = (I - A(x))^{-1}_{s,t}$, where I denotes an $\ell \times \ell$ identity matrix and all arithmetic is over GF(2). Recall that $L(x)$ is the submatrix of $A(x) - I$ obtained by deleting column s and row t. Hence, expressing $(I - A(x))^{-1}_{s,t}$ using the corresponding cofactor of $I - A(x)$, we have

$$\begin{aligned} (I - A(x))^{-1}_{s,t} &= (-1)^{s+t} \frac{\det(-L(x))}{\det(I - A(x))} \\ &= \det L(x). \qquad \square \end{aligned}$$

Let $r^{(1)}$ and $r^{(2)}$ be vectors over GF(2) of length $\sum_{i=1}^{\ell-2} i = \binom{\ell-1}{2}$ and $\ell - 2$, respectively. Let $R_1(r^{(1)})$ be an $(\ell-1) \times (\ell-1)$ matrix with 1's on the main diagonal, 0's below it, and the elements of $r^{(1)}$ in the remaining $\binom{\ell-1}{2}$ entries above the diagonal (a unique element of $r^{(1)}$ is assigned to each matrix entry). Let $R_2(r^{(2)})$ be an $(\ell - 1) \times (\ell - 1)$ matrix with 1's on the main diagonal, the elements of $r^{(2)}$ in the rightmost column, and 0's in each of the remaining entries. (See Figure 4.1.)

FACT 4.14 (see [38]). *Let M, M' be $(\ell-1) \times (\ell-1)$ matrices that contain the constant -1 in each entry of their second diagonal and the constant* 0 *below this diagonal. Then,* $\det(M) = \det(M')$ *if and only if there exist $r^{(1)}$ and $r^{(2)}$ such that $R_1(r^{(1)}) M R_2(r^{(2)}) = M'$.*

Proof sketch. Suppose that $R_1(r^{(1)}) M R_2(r^{(2)}) = M'$ for some $r^{(1)}$ and $r^{(2)}$. Then, since $\det(R_1(r^{(1)})) = \det(R_2(r^{(2)})) = 1$, it follows that $\det(M) = \det(M')$.

For the second direction assume that $\det(M) = \det(M')$. We show that there exist $r^{(1)}$ and $r^{(2)}$ such that $R_1(r^{(1)}) M R_2(r^{(2)}) = M'$. Multiplying M by a matrix $R_1(r^{(1)})$ on the left is equivalent to adding to each row of M a linear combination

of the rows below it. On the other hand, multiplying M by a matrix $R_2(r^{(2)})$ on the right is equivalent to adding to the last column of M a linear combination of the other columns. Observe that a matrix M that contains the constant -1 in each entry of its second diagonal and the constant 0 below this diagonal can be transformed, using such left and right multiplications, to a canonic matrix H_y containing -1's in its second diagonal, an arbitrary value y in its top-right entry, and 0's elsewhere. Since $\det(R_1(r^{(1)})) = \det(R_2(r^{(2)})) = 1$, we have $\det(M) = \det(H_y) = y$. Thus, when $\det(M) = \det(M') = y$ we can write $H_y = R_1(r^{(1)})MR_2(r^{(2)}) = R_1(s^{(1)})M'R_2(s^{(2)})$ for some $r^{(1)}, r^{(2)}, s^{(1)}, s^{(2)}$. Multiplying both sides by $R_1(s^{(1)})^{-1}, R_2(s^{(2)})^{-1}$ and observing that each set of matrices $R_1(\cdot)$ and $R_2(\cdot)$ forms a multiplicative group finishes the proof. □

LEMMA 4.15 (implicit in [38]). *Let BP be a mod-2 branching program computing the boolean function f. Define a degree-3 function $\hat{f}(x,(r^{(1)},r^{(2)}))$ whose outputs contain the $\binom{\ell}{2}$ entries on or above the main diagonal of the matrix $R_1(r^{(1)})L(x)R_2(r^{(2)})$. Then, $\hat{f}$ is a perfect randomized encoding of f.*

Proof. We start by showing that the encoding is stretch-preserving. The length of the random input of $\hat{f}$ is $m = \binom{\ell-1}{2} + \ell - 2 = \binom{\ell}{2} - 1$ and its output length is $s = \binom{\ell}{2}$. Thus we have $s = m + 1$, and since f is a boolean function its encoding $\hat{f}$ preserves its stretch.

We now describe the decoder and the simulator. Given an output of $\hat{f}$, representing a matrix M, the decoder C simply outputs $\det(M)$. (Note that the entries below the main diagonal of this matrix are constants and therefore are not included in the output of $\hat{f}$.) By Facts 4.13 and 4.14, $\det(M) = \det(L(x)) = f(x)$; hence the decoder is perfect.

The simulator S, on input $y \in \{0,1\}$, outputs the $\binom{\ell}{2}$ entries on and above the main diagonal of the matrix $R_1(r^{(1)})H_yR_2(r^{(2)})$, where $r^{(1)}$, $r^{(2)}$ are randomly chosen and H_y is the $(\ell-1)\times(\ell-1)$ matrix that contains -1's in its second diagonal, y in its top-right entry, and 0's elsewhere.

By Facts 4.13 and 4.14, for every $x \in \{0,1\}^n$ the supports of $\hat{f}(x,U_m)$ and of $S(f(x))$ are equal. Specifically, these supports include all strings in $\{0,1\}^s$ representing matrices with determinant $f(x)$. Since the supports of $S(0)$ and $S(1)$ form a disjoint partition of the entire space $\{0,1\}^s$ (by Fact 4.14) and since S uses $m = s-1$ random bits, it follows that $|\text{support}(S(b))| = 2^m$ for $b \in \{0,1\}$. Since both the simulator and the encoding use m random bits, it follows that both distributions, $\hat{f}(x,U_m)$ and $S(f(x))$, are uniform over their support and therefore are equivalent. Finally, since the supports of $S(0)$ and $S(1)$ halve the range of $\hat{f}$ (that is, $\{0,1\}^s$), the simulator is also balanced. □

Reducing the locality. It remains to convert the degree-3 encoding into one in NC^0. To this end, we show how to construct for any degree-d function (where d is constant) a $(d+1)$-local perfect encoding. Using the composition lemma, we can obtain an NC^0 encoding of a function by first encoding it as a constant-degree function and then applying the locality construction.

The idea for the locality construction is to represent a degree-d polynomial as a sum of monomials, each having locality d, and randomize this sum using a variant of the method for randomizing group product described in section 2.2. (A direct use of the latter method over the group Z_2 gives a $(d+2)$-local encoding instead of the $(d+1)$-local one obtained here.)

CONSTRUCTION 4.16 (locality construction). *Let $f(x) = T_1(x) + \cdots + T_k(x)$, where $f, T_1, \ldots, T_k : \text{GF}(2)^n \to \text{GF}(2)$ and summation is over* GF(2). *The local*

encoding $\hat{f} : \mathrm{GF}(2)^{n+(2k-1)} \to \mathrm{GF}(2)^{2k}$ *is defined by*

$$\hat{f}(x,(r_1,\ldots,r_k,r'_1,\ldots,r'_{k-1})) \stackrel{\text{def}}{=} (T_1(x)-r_1, T_2(x)-r_2,\ldots,T_k(x)-r_k,\\ r_1-r'_1, r'_1+r_2-r'_2,\ldots,r'_{k-2}+r_{k-1}-r'_{k-1}, r'_{k-1}+r_k).$$

For example, applying the locality construction to the polynomial $x_1x_2 + x_2x_3 + x_4$ results in the encoding $(x_1x_2 - r_1, x_2x_3 - r_2, x_4 - r_3, r_1 - r'_1, r'_1 + r_2 - r'_2, r'_2 + r_3)$.

LEMMA 4.17 (locality lemma). *Let f and $\hat{f}$ be as in Construction* 4.16. *Then, $\hat{f}$ is a perfect randomized encoding of f. In particular, if f is a degree-d polynomial written as a sum of monomials, then $\hat{f}$ is a perfect encoding of f with degree d and locality* $\max(d+1,3)$.

Proof. Since $m = 2k-1$ and $s = 2k$, the encoding $\hat{f}$ is stretch-preserving. Moreover, given $\hat{y} = \hat{f}(x,r)$, we can decode the value of $f(x)$ by summing up the bits of $\hat{y}$. It is not hard to verify that such a decoder never errs. To prove perfect privacy we define a simulator as follows. Given $y \in \{0,1\}$, the simulator S uniformly chooses $2k-1$ random bits $r_1,\ldots,r_{2k-1}$ and outputs $(r_1,\ldots,r_{2k-1}, y-(r_1+\cdots+r_{2k-1}))$. Obviously, $S(y)$ is uniformly distributed over the $2k$-length strings whose bits sum up to y over GF(2). It thus suffices to show that the outputs of $\hat{f}(x,U_m)$ are uniformly distributed subject to the constraint that they add up to $f(x)$. This follows by observing that, for any x and any assignment $w \in \{0,1\}^{2k-1}$ to the first $2k-1$ outputs of $\hat{f}(x,U_m)$, there is a unique way to set the random inputs r_i, r'_i so that the output of $\hat{f}(x,(r,r'))$ is consistent with w. Indeed, for $1 \le i \le k$, the values of x, w_i uniquely determine r_i. For $1 \le i \le k-1$, the values w_{k+i}, r_i, r'_{i-1} determine r'_i (where $r'_0 \stackrel{\text{def}}{=} 0$). Therefore, $S(f(x)) \equiv \hat{f}(x,U_m)$. Moreover, S is balanced since the supports of $S(0)$ and $S(1)$ halve $\{0,1\}^s$ and $S(y)$ is uniformly distributed over its support for $y \in \{0,1\}$. ∎

In Appendix B we describe a graph-based generalization of Construction 4.16, which in some cases can give rise to a (slightly) more compact encoding $\hat{f}$.

We now present the main theorem of this section.

THEOREM 4.18. $\oplus\mathrm{L}/poly \subseteq \mathcal{PREN}$. *Moreover, any $f \in \mathcal{PREN}$ admits a perfect randomized encoding in* NC^0_4.

Proof. The first part of the theorem is derived by combining the degree-3 construction of Lemma 4.15 together with the locality lemma (Lemma 4.17), using the composition lemma (Lemma 4.11) and the concatenation lemma (Lemma 4.10).

To prove the second part, we first encode f by a perfect encoding $\hat{f}$ in NC^0 (guaranteed by the fact that f is in $\mathcal{PREN}$). Then, since $\hat{f}$ is in $\oplus\mathrm{L}/poly$, we can use our constructions (Lemmas 4.15, 4.17, 4.11, 4.10) to perfectly encode $\hat{f}$ by a function $\hat{f}'$ in NC^0_4. By the composition lemma (Lemma 4.11), $\hat{f}'$ perfectly encodes the function f. ∎

Remark 4.19. An alternative construction of perfect randomized encodings in NC^0 can be obtained using a randomizing polynomial construction from [38, section 3], which is based on an information-theoretic variant of Yao's garbled circuit technique [53]. This construction yields an encoding with a (large) constant locality, without requiring an additional "locality reduction" step (of Construction 4.16). This construction is weaker than the current one in that it only efficiently applies to functions in NC^1 rather than $\oplus\mathrm{L}/poly$. For functions in NC^1, the complexity of this alternative (in terms of randomness and output length) is incomparable to the complexity of the current construction.

There are variants of the above construction that can handle nondeterministic branching programs as well, at the expense of losing perfectness [37, 38]. For instance, it is shown in [37] that if f is represented by a nondeterministic BP of size ℓ, then the function $\hat{f}(x,(R_1,R_2)) \stackrel{\text{def}}{=} R_1 L(x) R_2$ is a perfectly private, statistically correct encoding of f provided that R_1, R_2 are uniformly random $(\ell-1)\times(\ell-1)$ matrices over $\mathrm{GF}(p)$, where p is prime and $p > \ell^\ell$. (The matrix $L(x)$ is as defined above, except that here it is interpreted as a matrix over $\mathrm{GF}(p)$.) To obtain an encoding over a binary alphabet, we rely on the facts that one can sample an almost uniform element of $\mathrm{GF}(p)$ (up to a negligible statistical distance) as well as perform multiplications in $\mathrm{GF}(p)$ using NC^1 boolean circuits. Thus, we get a statistical *binary* encoding in NC^1, which can be converted (using Theorem 4.18 and the composition lemma (Lemma 4.11)) to a statistical encoding in NC^0_4. Based on the above, we get the following theorem.

THEOREM 4.20. *$\mathrm{NL}/poly \subseteq \mathcal{SREN}$. Moreover, any $f \in \mathcal{SREN}$ admits a statistical randomized encoding in NC^0_4.*

Note that the second part of Theorem 4.20 can be proved similarly to the second part of Theorem 4.18.

5. One-way functions in $\mathbf{NC^0}$. A *one-way function* (OWF) $f : \{0,1\}^* \to \{0,1\}^*$ is a polynomial-time computable function that is hard to invert; namely, every polynomial-time algorithm that tries to invert f on input $f(x)$, where x is picked from U_n, succeeds only with a negligible probability. Formally, we have the following definition.

DEFINITION 5.1 (one-way function). *A function $f : \{0,1\}^* \to \{0,1\}^*$ is called a* one-way function *(OWF) if it satisfies the following two properties:*

- Easy to compute: *There exists a deterministic polynomial-time algorithm computing $f(x)$.*
- Hard to invert: *For every probabilistic polynomial-time algorithm, B, the probability $\Pr_{x\leftarrow U_n}[B(1^n, f(x)) \in f^{-1}(f(x))]$ is negligible in n (where the probability is taken over a uniform choice of x and the internal coin tosses of B).*

The function f is called weakly one-way *if the second requirement is replaced with the following (weaker) one:*

- Slightly hard to invert: *There exists a polynomial $p(\cdot)$, such that for every probabilistic polynomial-time algorithm, B, and all sufficiently large n's $\Pr_{x\leftarrow U_n}[B(1^n, f(x)) \notin f^{-1}(f(x))] > \frac{1}{p(n)}$ (where the probability is taken over a uniform choice of x and the internal coin tosses of B).*

The above definition naturally extends to functions whose domain is restricted to some infinite subset $I \subset \mathbb{N}$ of the possible input lengths, such as ones defined by a randomized encoding $\hat{f}$. As argued in Remark 4.7, such a partially defined OWF can be augmented into a fully defined OWF provided that the set I is polynomially dense and efficiently recognizable (which is a feature of functions $\hat{f}$ obtained via uniform encodings).

5.1. Key lemmas. In the following we show that a perfectly correct and statistically private randomized encoding $\hat{f}$ of an OWF f is also an OWF. The idea, as described in section 2.1, is to argue that the hardness of inverting $\hat{f}$ reduces to the hardness of inverting f. The case of a statistical randomized encoding that does not enjoy perfect correctness is more involved and will be dealt with later in this section.

LEMMA 5.2. *Suppose that $f : \{0,1\}^* \to \{0,1\}^*$ is hard to invert and $\hat{f}(x,r)$ is a perfectly correct, statistically private uniform encoding of f. Then $\hat{f}$, viewed as a single-argument function, is also hard to invert.*

Proof. Let $s = s(n), m = m(n)$ be the lengths of the output and of the random input of $\hat{f}$, respectively. Note that $\hat{f}$ is defined on input lengths of the form $n+m(n)$; we prove that it is hard to invert on these inputs. Assume, toward a contradiction, that there is an efficient algorithm $\hat{B}$ inverting $\hat{f}(x,r)$ with success probability $\phi(n+m) > \frac{1}{q(n+m)}$ for some polynomial $q(\cdot)$ and infinitely many n's. We use $\hat{B}$ to construct an efficient algorithm B that inverts f with similar success. On input $(1^n, y)$, the algorithm B runs S, the statistical simulator of $\hat{f}$, on the input $(1^n, y)$ and gets a string $\hat{y}$ as the output of S. Next, B runs the inverter $\hat{B}$ on the input $(1^{n+m}, \hat{y})$, getting (x', r') as the output of $\hat{B}$ (i.e., $\hat{B}$ "claims" that $\hat{f}(x', r') = \hat{y}$). B terminates with output x'.

Complexity. Since S and $\hat{B}$ are both polynomial-time algorithms, and since $m(n)$ is polynomially bounded, it follows that B is also a polynomial-time algorithm.

Correctness. We analyze the success probability of B on input $(1^n, f(x))$, where $x \leftarrow U_n$. Let us assume for a moment that the simulator S is perfect. Observe that, by perfect correctness, if $f(x) \neq f(x')$, then the support sets of $\hat{f}(x, U_m)$ and $\hat{f}(x', U_m)$ are disjoint. Moreover, by perfect privacy the string $\hat{y}$, generated by $\hat{B}$, is always in the support of $\hat{f}(x, U_m)$. Hence, if $\hat{B}$ succeeds (that is, indeed $\hat{y} = \hat{f}(x', r')$), then so does B (namely, $f(x') = y$). Finally, observe that (by Fact 3.4) the input $\hat{y}$ on which B invokes $\hat{B}$ is distributed identically to $\hat{f}_n(U_n, U_{m(n)})$, and therefore B succeeds with probability $\geq \phi(n+m)$. Formally, we can write

$$\begin{aligned}
\Pr_{x \leftarrow U_n}[B(1^n, f(x)) \in f^{-1}(f(x))] &\geq \Pr_{x \leftarrow U_n, \hat{y} \leftarrow S(1^n, f(x))}[\hat{B}(1^{n+m}, \hat{y}) \in \hat{f}^{-1}(\hat{y})] \\
&= \Pr_{x \leftarrow U_n, r \leftarrow U_{m(n)}}[\hat{B}(1^{n+m}, \hat{f}_n(x,r)) \in \hat{f}^{-1}(\hat{f}(x,r))] \\
&\geq \phi(n+m).
\end{aligned}$$

When S is only statistically private, we lose negligible success probabilities in the first and second transitions. The first loss is due to the fact that the simulator invoked on $y = f(x)$ might output (with negligible probability) $\hat{y}$ which is not in the support of $\hat{f}(x, U_m)$. The second loss is due to the fact that the input $\hat{y}$ on which B invokes $\hat{B}$ is not distributed identically to $\hat{f}(U_n, U_m)$, on which $\hat{B}$ is guaranteed to succeed with probability $\phi(n+m)$. However, it follows from Fact 3.4 that the second loss is also negligible. Thus, if S is $\varepsilon(n)$-private for a negligible function $\varepsilon(\cdot)$, we have

$$\begin{aligned}
\Pr_{x \leftarrow U_n}[B(1^n, f(x)) \in f^{-1}(f(x))] &\geq \Pr_{x \leftarrow U_n, \hat{y} \leftarrow S(1^n, f(x))}[\hat{B}(1^{n+m}, \hat{y}) \in \hat{f}^{-1}(\hat{y})] - \varepsilon(n) \\
&\geq \Pr_{x \leftarrow U_n, r \leftarrow U_{m(n)}}[\hat{B}(1^{n+m}, \hat{f}_n(x,r)) \in \hat{f}^{-1}(\hat{f}(x,r))] \\
&\quad - \varepsilon(n) - \varepsilon(n) \\
&\geq \phi(n+m) - 2\varepsilon(n) > \frac{1}{q(n+m)} - 2\varepsilon(n) > \frac{1}{q'(n)}
\end{aligned}$$

for some polynomial $q'(\cdot)$ and infinitely many n's. It follows that f is not hard to invert, in contradiction to the hypothesis. ∎

The efficiency of the simulator S is essential for Lemma 5.2 to hold. Indeed, without this requirement one could encode any one-way permutation f by the identity function $\hat{f}(x) = x$, which is obviously not one-way. (Note that the output of $\hat{f}(x)$ can be simulated inefficiently based on $f(x)$ by inverting f.)

The perfect correctness requirement is also essential for Lemma 5.2 to hold. To see this, consider the following example. Suppose f is a one-way permutation. Consider the encoding $\hat{f}(x,r)$ which equals $f(x)$ except if r is the all-zero string, in which case $\hat{f}(x,r) = x$. This is a statistically correct and statistically private encoding, but $\hat{f}$ is easily invertible since on value $\hat{y}$ the inverter can always return $\hat{y}$ itself as a possible preimage. Still, we show below that such an $\hat{f}$ (which is only statistically correct) is a *distributionally* one-way function. We will later show how to turn a distributionally one-way function in NC^0 into an OWF in NC^0.

DEFINITION 5.3 (distributionally one-way function [35]). *A polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ is called* distributionally one-way *if there exists a positive polynomial $p(\cdot)$ such that for every probabilistic polynomial-time algorithm, B, and all sufficiently large n's, $\|(B(1^n, f(U_n)), f(U_n)) - (U_n, f(U_n))\| > \frac{1}{p(n)}$.*

Before proving that a statistical randomized encoding of an OWF is distributionally one-way, we need the following lemma.

LEMMA 5.4. *Let $f, g : \{0,1\}^* \to \{0,1\}^*$ be two functions that differ on a negligible fraction of their domain; that is, $\Pr_{x \leftarrow U_n}[f(x) \neq g(x)]$ is negligible in n. Suppose that g is slightly hard to invert (but is not necessarily computable in polynomial time) and that f is computable in polynomial time. Then, f is distributionally one-way.*

Proof. Let f_n and g_n be the restrictions of f and g to n-bit inputs, that is $f = \{f_n\}, g = \{g_n\}$, and define $\varepsilon(n) \stackrel{\text{def}}{=} \Pr_{x \leftarrow U_n}[f(x) \neq g(x)]$. Let $p(n)$ be the polynomial guaranteed by the assumption that g is slightly hard to invert. Assume, toward a contradiction, that f is not distributionally one-way. Then, there exists a polynomial-time algorithm, B, such that for infinitely many n's, $\|(B(1^n, f_n(U_n)), f_n(U_n)) - (U_n, f_n(U_n))\| \leq \frac{1}{2p(n)}$. Since $(U_n, f_n(U_n)) \equiv (x', f_n(U_n))$ where $x' \leftarrow f_n^{-1}(f_n(U_n))$, we get that for infinitely many n's, $\|(B(1^n, f_n(U_n)), f_n(U_n)) - (x', f_n(U_n))\| \leq \frac{1}{2p(n)}$. It follows that for infinitely many n's,

$$\Pr[B(1^n, f(U_n)) \in g_n^{-1}(f_n(U_n))] \geq \Pr_{x' \leftarrow f_n^{-1}(f_n(U_n))}[x' \in g_n^{-1}(f_n(U_n))] - \frac{1}{2p(n)}. \tag{5.1}$$

We show that B inverts g with probability greater than $1 - \frac{1}{p(n)}$ and derive a contradiction. Specifically, for infinitely many n's we have

$$\begin{aligned}
\Pr[B(1^n, g_n(U_n)) \in g_n^{-1}(g_n(U_n))] &\geq \Pr[B(1^n, f_n(U_n)) \in g_n^{-1}(f_n(U_n))] - \varepsilon(n) \\
&\geq \Pr_{x' \leftarrow f_n^{-1}(f_n(U_n))}[x' \in g_n^{-1}(f(U_n))] - \frac{1}{2p(n)} - \varepsilon(n) \\
&= \Pr_{x' \leftarrow f_n^{-1}(f_n(U_n))}[g_n(x') = f_n(U_n)] - \frac{1}{2p(n)} - \ \varepsilon(n) \\
&= \Pr_{x' \leftarrow f_n^{-1}(f_n(U_n))}[g_n(x') = f_n(x')] - \frac{1}{2p(n)} - \ \varepsilon(n) \\
&= 1 - \varepsilon(n) - \frac{1}{2p(n)} - \ \varepsilon(n) \\
&\geq 1 - \frac{1}{p(n)},
\end{aligned}$$

where the first inequality is due to the fact that f and g are ε-close, the second inequality uses (5.1), the second equality follows since $f(U_n) = f(x')$, the third equality is due to $x' \equiv U_n$, and the last inequality follows since ε is negligible. ∎

We now use Lemma 5.4 to prove the distributional one-wayness of a statistically correct encoding $\hat{f}$ based on the one-wayness of a related, perfectly correct, encoding g.

LEMMA 5.5. *Suppose that $f : \{0,1\}^* \to \{0,1\}^*$ is a one-way function and $\hat{f}(x,r)$ is a statistical randomized encoding of f. Then $\hat{f}$, viewed as a single-argument function, is distributionally one-way.*

Proof. Let C and S be the decoder and the simulator of $\hat{f}$. Define the function $\hat{g}(x,r)$ in the following way: If $C(\hat{f}(x,r)) \neq f(x)$, then $\hat{g}(x,r) = \hat{f}(x,r')$ for some r' such that $C(\hat{f}(x,r')) = f(x)$ (such an r' exists by the statistical correctness); otherwise, $\hat{g}(x,r) = \hat{f}(x,r)$. Obviously, $\hat{g}$ is a perfectly correct encoding of f (as C perfectly decodes $f(x)$ from $\hat{g}(x,r)$). Moreover, by the statistical correctness of C, we have that $\hat{f}(x,\cdot)$ and $\hat{g}(x,\cdot)$ differ only on a negligible fraction of the r's. It follows that $\hat{g}$ is also a statistically private encoding of f (because $\hat{g}(x,U_m) \stackrel{s}{\approx} \hat{f}(x,U_m) \stackrel{s}{\approx} S(f(x))$). Since f is hard to invert, it follows from Lemma 5.2 that $\hat{g}$ is also hard to invert. (Note that $\hat{g}$ might not be computable in polynomial time; however, the proof of Lemma 5.2 requires only that the simulator's running time and the randomness complexity of $\hat{g}$ be polynomially bounded.) Finally, it follows from Lemma 5.4 that $\hat{f}$ is distributionally one-way as required. ∎

5.2. Main results. Based on the above, we derive the main theorem of this section.

THEOREM 5.6. *If there exists an OWF in $\mathcal{SREN}$, then there exists an OWF in NC^0_4.*

Proof. Let f be an OWF in $\mathcal{SREN}$. By Lemma 5.5, we can construct a distributional OWF $\hat{f}$ in NC^0 and then apply a standard transformation (cf. [35, Lemma 1], [23, p. 96], [52]) to convert $\hat{f}$ to an OWF $\hat{f}'$ in NC^1. This transformation consists of two steps: Impagliazzo and Luby's NC^1 construction of weak OWFs from distributional OWFs [35], and Yao's NC^0 construction of a (standard) OWF from a weak OWF [52] (see [23, section 2.3]).[11] Since $NC^1 \subseteq \mathcal{PREN}$ (Theorem 4.18), we can use Lemma 5.2 to encode $\hat{f}'$ by an OWF in NC^0, in particular, by one with locality 4. ∎

Combining Lemmas 5.2 and 4.12 and Theorem 4.18, we get a similar result for one-way permutations (OWPs).

THEOREM 5.7. *If there exists an OWP in $\mathcal{PREN}$, then there exists an OWP in NC^0_4.*

In particular, using Theorems 4.18 and 4.20, we conclude that an OWF (resp., OWP) in NL/*poly* (resp., $\oplus$L/*poly*) implies an OWF (resp., OWP) in NC^0_4.

Theorem 5.7 can be extended to trapdoor permutations (TDPs) provided that the perfect encoding satisfies the following *randomness reconstruction* property: Given x and $\hat{f}(x,r)$, the randomness r can be efficiently recovered. If this is the case, then the trapdoor of f can be used to invert $\hat{f}(x,r)$ in polynomial time (but not in NC^0). First, we compute $f(x)$ from $\hat{f}(x,r)$ using the decoder; second, we use the trapdoor-inverter to compute x from $f(x)$; and, finally, we use the randomness reconstruction algorithm to compute r from x and $\hat{f}(x,r)$. The randomness reconstruction property is satisfied by the randomized encodings described in section 4.3 and is preserved under composition and concatenation. Thus, the existence of TDPs computable in NC^0_4 follows from their existence in $\oplus$L/*poly*.

[11] We will later show a degree-preserving transformation from a distributional OWF to an OWF (Lemma 8.2); however, in the current context the standard transformation suffices.

More formally, a collection of permutations $\mathcal{F} = \{f_z : D_z \to D_z\}_{z \in Z}$ is referred to as a TDP if there exist probabilistic polynomial-time algorithms (I, D, F, F^{-1}) with the following properties. Algorithm I is an index selector algorithm that on input 1^n selects an index z from Z and a corresponding trapdoor for f_z; algorithm D is a domain sampler that on input z samples an element from the domain D_z; F is a function evaluator that given an index z and x returns $f_z(x)$; and F^{-1} is a trapdoor-inverter that given an index z, a corresponding trapdoor t, and $y \in D_z$ returns $f_z^{-1}(y)$. Additionally, the collection should be hard to invert, similar to a standard collection of OWPs. (For a formal definition see [23, Definition 2.4.4].) By the above argument we derive the following theorem.

THEOREM 5.8. *If there exists a TDP $\mathcal{F}$ whose function evaluator F is in $\oplus$L/poly, then there exists a TDP $\hat{\mathcal{F}}$ whose function evaluator $\hat{F}$ is in* NC^0_4.

Remarks on Theorems 5.6, 5.7, *and* 5.8.

1. *Constructiveness.* In section 4.3, we give a constructive way of transforming a branching program representation of a function f into an NC^0 circuit computing its encoding $\hat{f}$. It follows that Theorems 5.6 and 5.7 can be made constructive in the following sense: There exists a polynomial-time *compiler* transforming a branching program representation of an OWF (resp., OWP) f into an NC^0 representation of a corresponding OWF (resp., OWP) $\hat{f}$. A similar result holds for other cryptographic primitives considered in this paper.
2. *Preservation of security: a finer look.* Loosely speaking, the main security loss in the reduction follows from the expansion of the input. (The simulator's running time has only a minor effect on the security, since it is added to the overall running time of the adversary.) Thus, to achieve a level of security similar to that achieved by applying f on n-bit inputs, one would need to apply $\hat{f}$ on $n+m(n)$ bits (the random input part of the encoding does not contribute to the security). Going through our constructions (bit-by-bit encoding of the output based on some size-$\ell(n)$ BPs, followed by the locality construction), we get $m(n) = l(n) \cdot \ell(n)^{O(1)}$, where $l(n)$ is the output length of f. If the degree of all nodes in the BPs is bounded by a constant, the complexity is $m(n) = O(l(n) \cdot \ell(n)^2)$. It is possible to further reduce the overhead of randomized encoding for specific representation models, such as balanced formulas, using constructions of randomizing polynomials from [38, 15].
3. *Generalizations.* The proofs of the above theorems carry over to OWFs whose security holds against efficient *nonuniform* adversaries (inverters). The same is true for all cryptographic primitives considered in this work. The proofs also naturally extend to the case of *collections* of OWFs and OWPs (see Appendix A for discussion).
4. *Concrete assumptions.* The existence of an OWF in $\mathcal{SREN}$ (in fact, even in NC^1) follows from the intractability of factoring and lattice problems [2]. The existence of an OWF *collection* in $\mathcal{SREN}$ follows from the intractability of the discrete logarithm problem. Thus, we get OWFs in NC^0_4 under most standard cryptographic assumptions. In the case of OWPs, we can get a collection of OWPs in NC^0_4 based on the discrete logarithm problem [11, 52] (see also Appendix A) or RSA with a small exponent [49].[12] The latter assumption is also sufficient for the construction of TDP in NC^0_4.

[12] Rabin's factoring-based OWP collection [47] seems insufficient for our purposes, as it cannot be defined over the set of *all* strings of a given length. The standard modification (cf. [24, p. 767]) does not seem to be in $\oplus$L/*poly*.

6. Pseudorandom generators in NC^0. A *pseudorandom generator* (PRG) is an efficiently computable function $G : \{0,1\}^n \to \{0,1\}^{l(n)}$ such that (1) G has a positive stretch, namely $l(n) > n$, where we refer to the function $l(n) - n$ as the *stretch* of the generator and (2) any "computationally restricted procedure" D, called a *distinguisher*, has a negligible advantage in distinguishing $G(U_n)$ from $U_{l(n)}$. That is, $|\Pr[D(1^n, G(U_n)) = 1] - \Pr[D(1^n, U_{l(n)}) = 1]|$ is negligible in n.

Different notions of PRGs differ mainly in the computational bound imposed on D. In the default case of *cryptographic* PRGs, D can be any probabilistic polynomial-time algorithm (alternatively, polynomial-size circuit family). In the case of *ϵ-biased* generators, D can only compute a linear function of the output bits, namely the exclusive-or of some subset of the bits. Other types of PRGs, e.g., for space-bounded computation, have also been considered. The reader is referred to [21, Chapter 3] for a comprehensive and unified treatment of pseudorandomness.

We start by considering cryptographic PRGs. We show that a *perfect* randomized encoding of such a PRG is also a PRG. We then obtain a similar result for other types of PRGs.

6.1. Cryptographic generators.

DEFINITION 6.1 (pseudorandom generator). *A pseudorandom generator (PRG) is a polynomial-time computable function, $G : \{0,1\}^n \to \{0,1\}^{l(n)}$, satisfying the following two conditions:*

- Expansion: *$l(n) > n$ for all $n \in \mathbb{N}$.*
- Pseudorandomness: *For every probabilistic polynomial-time algorithm, D, the distinguishing advantage $|\Pr[D(1^n, G(U_n)) = 1] - \Pr[D(1^n, U_{l(n)}) = 1]|$ is negligible in n.*

Remark 6.2 (PRGs with sublinear stretch). An NC^0 PRG, G, that stretches its input by a single bit can be transformed into another NC^0 PRG, G', with stretch $l'(n) - n = n^c$ for an arbitrary constant $c < 1$. This can be done by applying G on n^c blocks of n^{1-c} bits and concatenating the results. Since the output of any PRG is computationally indistinguishable from the uniform distribution even by a polynomial number of samples (see [23, Theorem 3.2.6]), the block generator G' is also a PRG. This PRG gains a pseudorandom bit from every block and therefore stretches $n^c n^{1-c} = n$ input bits to $n + n^c$ output bits. Obviously, G' has the same locality as G.

Remark 6.2 also applies to other types of generators considered in this section, and therefore we only use a crude classification of the stretch as being "sublinear," "linear," or "superlinear."

LEMMA 6.3. *Suppose $G : \{0,1\}^n \to \{0,1\}^{l(n)}$ is a PRG and $\hat{G} : \{0,1\}^n \times \{0,1\}^{m(n)} \to \{0,1\}^{s(n)}$ is a uniform perfect randomized encoding of G. Then $\hat{G}$, viewed as a single-argument function, is also a PRG.*

Proof. Since $\hat{G}$ is stretch-preserving, it is guaranteed to expand its seed. To prove the pseudorandomness of its output, we again use a reducibility argument. Assume, toward a contradiction, that there exists an efficient distinguisher $\hat{D}$ that distinguishes between U_s and $\hat{G}(U_n, U_m)$ with some nonnegligible advantage ϕ, i.e., ϕ such that $\phi(n+m) > \frac{1}{q(n+m)}$ for some polynomial $q(\cdot)$ and infinitely many n's. We use $\hat{D}$ to obtain a distinguisher D between U_l and $G(U_n)$ as follows. On input $y \in \{0,1\}^l$, run the balanced simulator of $\hat{G}$ on y and invoke $\hat{D}$ on the resulting $\hat{y}$. If y is taken from U_l, then the simulator, being balanced, outputs $\hat{y}$ that is distributed as U_s. On the other hand, if y is taken from $G(U_n)$, then, by Fact 3.4, the output of the simulator is distributed as $\hat{G}(U_n, U_m)$. Thus, the distinguisher D we get for

G has the same advantage as the distinguisher $\hat{D}$ for $\hat{G}$. That is, the advantage of D is $\phi'(n) = \phi(n+m)$. Since $m(n)$ is polynomial, this advantage ϕ' is not only nonnegligible in $n+m$ but also in n, in contradiction to the hypothesis. □

Remark 6.4 (the role of balance and stretch preservation). Dropping either the balance or stretch preservation requirements, Lemma 6.3 would no longer hold. To see this, consider the following two examples. Let G be a PRG, and let $\hat{G}(x,r) = G(x)$. Then, $\hat{G}$ is a perfectly correct, perfectly private, and balanced randomized encoding of G (the balanced simulator is $S(y) = y$). However, when r is sufficiently long, $\hat{G}$ does not expand its seed. On the other hand, we can define $\hat{G}(x,r) = G(x)0$, where r is a single random bit. Then, $\hat{G}$ is perfectly correct, perfectly private, and stretch-preserving, but its output is not pseudorandom.

Using Lemma 6.3 and Theorem 4.18, we get the following theorem.

THEOREM 6.5. *If there exists a PRG in $\mathcal{PREN}$ (in particular, in $\oplus$L/poly), then there exists a PRG in* NC^0_4.

As in the case of OWFs, an adversary that breaks the transformed generator $\hat{G}$ can break, in essentially the same time, the original generator G. Therefore, again, although the new PRG uses $m(n)$ extra random input bits, it is not more secure than the original generator applied to n bits. Moreover, we stress that the PRG $\hat{G}$ one gets from our construction has a sublinear stretch even if G has a large stretch. This follows from the fact that the length $m(n)$ of the random input is typically superlinear in the input length n.

Remark 6.6 (on the existence of a PRG in $\mathcal{PREN}$). The existence of PRGs in $\mathcal{PREN}$ follows from most standard concrete intractability assumptions. In particular, using Theorem 6.5 (applied to PRG collections) one can construct a collection of PRGs in NC^0_4 based on the intractability of factoring [39, 44] and the discrete logarithm problem [11, 52]. The existence of PRGs in $\mathcal{PREN}$ also follows from the existence in $\mathcal{PREN}$ of any *regular* OWF, i.e., an OWF $f = \{f_n\}$ that maps the same (polynomial-time computable) number of elements in $\{0,1\}^n$ to every element in $\mathrm{Im}(f_n)$. (This is the case, for instance, for any one-to-one OWF.) Indeed, the PRG construction from [33] (Theorem 5.4), when applied to a regular OWF f, involves only the computation of universal hash functions and hard-core bits, which can all be implemented in NC^1.[13] Thus a regular OWF in $\mathcal{PREN}$ can be transformed first into a regular OWF in NC^0 and then, using [33], into a PRG in NC^1. Combined with Theorem 6.5, this yields a PRG in NC^0_4 based on any regular OWF in $\mathcal{PREN}$.[14] This

[13] In the general case (when the OWF f is not regular) the construction of Håstad et al. (see [33, Construction 7.1]) is not in uniform NC^1, as it requires an additional nonuniform advice of logarithmic length. This (slightly) nonuniform NC^1 construction translates into a *polynomial-time* construction by applying the following steps: (1) construct a polynomial number of PRG candidates (each using a different guess for the nonuniform advice); (2) increase the stretch of each of these candidates using the standard transformation of Goldreich and Micali (cf. [23, Theorem 3.3.3]); (3) take the exclusive-or of all PRG candidates to obtain the final PRG. The second step requires polynomially many sequential applications of the PRGs, and therefore this construction is not in NC^1. (If we skip the second step, the resulting generator will not stretch its input.)

[14] In fact, the same result can be obtained under a relaxed regularity requirement. Specifically, for each n and $y \in \mathrm{Im}(f_n)$ define the value $D_{f,n}(y) = \log |f_n^{-1}(y)|$ and the random variable $R_n = D_{f,n}(f(U_n))$. The NC^1 construction of [33, Construction 7.1] needs to approximate, in $\mathrm{poly}(n)$ time, the expectations of both R_n and R_n^2. This is trivially possible when f is regular in the strict sense defined above, since in this case R_n is concentrated on a single (efficiently computable) value. Using a recent NC^1 construction from [30], only the expectation of R_n^2 needs to be efficiently approximated. We finally note that in a nonuniform computation model one can rely on [33] (which gives a nonuniform NC^1 construction of a PRG from any OWF) and get a PRG in *nonuniform* NC^0_4 from *any* OWF in $\mathcal{SREN}$.

way, for example, one can construct a (single) PRG in NC^0_4 based on the intractability of lattice problems [33, 2].

Remark 6.7 (on unconditional NC^0 reductions from PRG to OWF). Our machinery can be used to obtain an NC^0 reduction from a PRG to any regular OWF (in particular, to any one-to-one OWF), regardless of the complexity of f.[15] Moreover, this reduction only makes a *black-box* use of the underlying regular OWF f (given its regularity parameter $|\mathrm{Im}(f_n)|$). The general idea is to encode the NC^1 construction of [33, Construction 7.1] into a corresponding NC^0 construction. Specifically, suppose $G(x) = g(x, f(q_1(x)), \ldots, f(q_m(x)))$ defines a black-box construction of a PRG G from an OWF f, where g is in $\mathcal{PREN}$ and the q_i's are in NC^0. (The functions $g, q_1, \ldots, q_m$ are fixed by the reduction and do not depend on f.) Then, letting $\hat{g}((x, y_1, \ldots, y_m), r)$ be a perfect NC^0 encoding of g, the function $\hat{G}(x, r) = \hat{g}((x, f(q_1(x)), \ldots, f(q_m(x))), r)$ perfectly encodes G, and hence defines a black-box NC^0 reduction from a PRG to an OWF. The construction of [33, Construction 7.1] is of the form of $G(x)$ above,[16] assuming that f is regular. Thus, $\hat{G}$ defines an NC^0 reduction from a PRG to a regular OWF.

Comparison with lower bounds. The results of [43] rule out the existence of a superlinear-stretch cryptographic PRG in NC^0_4. Thus our NC^0_4 cryptographic PRGs are not far from optimal despite their sublinear stretch. In addition, it is easy to see that there is no PRG with degree 1 or locality 2 (since we can easily decide whether a given string is in the range of such a function). It seems likely that a cryptographic PRG with locality 3 and degree 2 can be constructed (e.g., based on its existence in a higher complexity class), but our positive result is one step short in terms of both locality and degree. (See also Table 6.1.)

6.2. ε-biased generators. The proof of Lemma 6.3 uses the balanced simulator to transform a distinguisher for a PRG G into a distinguisher for its encoding $\hat{G}$. Therefore, if this transformation can be made linear, then the security reduction goes through also in the case of ε-biased generators.

DEFINITION 6.8 (ε-biased generator). *An* ε-biased generator *is a polynomial-time computable function,* $G : \{0,1\}^n \to \{0,1\}^{l(n)}$*, satisfying the following two conditions:*

- Expansion: $l(n) > n$ *for all* $n \in \mathbb{N}$.
- ε-bias: *For every linear function* $L : \{0,1\}^{l(n)} \to \{0,1\}$ *and all sufficiently large* n*'s,*

$$|\Pr[L(G(U_n)) = 1] - \Pr[L(U_{l(n)}) = 1]| < \varepsilon(n)$$

(where a function L *is* linear *if its degree over* GF(2) *is* 1*). By default, the function* $\varepsilon(n)$ *is required to be negligible.*

LEMMA 6.9. *Let* G *be an* ε*-biased generator and* $\hat{G}$ *a perfect randomized encoding of* G*. Assume that the balanced simulator* S *of* $\hat{G}$ *is* linear *in the sense that* $S(y)$ *outputs a randomized linear transformation of* y *(which is not necessarily a linear function of the simulator's randomness). Then,* $\hat{G}$ *is also an* ε*-biased generator.*

Proof. Let $G : \{0,1\}^n \to \{0,1\}^{l(n)}$ and let $\hat{G} : \{0,1\}^n \times \{0,1\}^{m(n)} \to \{0,1\}^{s(n)}$. Assume, toward a contradiction, that $\hat{G}$ is not ε-biased; that is, for some linear function $L : \{0,1\}^{s(n)} \to \{0,1\}$ and infinitely many n's, $|\Pr[L(\hat{G}(U_{n+m})) = 1] -$

[15] Viola, in a concurrent work [50], obtains an AC^0 reduction of this type.

[16] The functions $q_1, \ldots, q_m$ are simply projections there. Interestingly, the recent NC^1 construction from [30] is not of the above form and thus we cannot encode it into an (unconditional) NC^0 construction.

$\Pr[L(U_s) = 1]| > \frac{1}{p(n+m)} > \frac{1}{p'(n)}$, where $m = m(n)$, $s = s(n)$, and $p(\cdot), p'(\cdot)$ are polynomials. Using the balance property we get

$$|\Pr[L(S(G(U_n))) = 1] - \Pr[L(S(U_l)) = 1]| = |\Pr[L(\hat{G}(U_{n+m})) = 1] - \Pr[L(U_s) = 1]| > \frac{1}{p'(n)},$$

where S is the balanced simulator of $\hat{G}$ and the probabilities are taken over the inputs as well as the randomness of S. By an averaging argument we can fix the randomness of S to some string ρ and get $|\Pr[L(S_\rho(G(U_n))) = 1] - \Pr[L(S_\rho(U_{l(n)})) = 1]| > \frac{1}{p'(n)}$, where S_ρ is the deterministic function defined by using the constant string ρ as the simulator's random input. By the linearity of the simulator, the function $S_\rho : \{0,1\}^l \to \{0,1\}^s$ is linear; therefore the composition of L and S_ρ is also linear, and thus the last inequality implies that G is not ε-biased in contradiction to the hypothesis. □

We now argue that the balanced simulators obtained in section 4.3 are all linear in the above sense. In fact, these simulators satisfy a stronger property: For every fixed random input of the simulator, each bit of the simulator's output is determined by a single bit of its input. This simple structure is due to the fact that we encode nonboolean functions by concatenating the encodings of their output bits. We state here the stronger property as it will be needed in the next subsection.

OBSERVATION 6.10. *Let S be a simulator of a randomized encoding (of a function) that is obtained by concatenating simulators (i.e., S is defined as in the proof of Lemma 4.9). Then, fixing the randomness ρ of S, the simulator's computation has the following simple form: $S_\rho(y) = \sigma_1(y_1)\sigma_2(y_2)\cdots\sigma_l(y_l)$, where each σ_i maps y_i (i.e., the ith bit of y) to one of two fixed strings. In particular, S computes a randomized degree-1 function of its input.*

Recall that the balanced simulator of the NC^0_4 encoding for functions in $\oplus\mathrm{L}/poly$ (promised by Theorem 4.18) is obtained by concatenating the simulators of boolean functions in $\oplus\mathrm{L}/poly$. By Observation 6.10, this simulator is linear. Thus, by Lemma 6.9, we can construct a sublinear-stretch ε-biased generator in NC^0_4 from any ε-biased generator in $\oplus\mathrm{L}/poly$. In fact, one can easily obtain a nontrivial ε-biased generator even in NC^0_3 by applying the locality construction to each of the bits of the degree-2 generator defined by $G(x, x') = (x, x', \langle x, x' \rangle)$, where $\langle \cdot, \cdot \rangle$ denotes inner product modulo 2. Again, the resulting encoding is obtained by concatenation and thus, by Observation 6.10 and Lemma 6.9, is also ε-biased. (This generator actually fools a much larger class of statistical tests; see section 6.3 below.) Thus, we have the following theorem.

THEOREM 6.11. *There is a (sublinear-stretch) ε-biased generator in NC^0_3.*

Building on a construction of Mossel, Shpilka, and Trevisan [43], it is in fact possible to achieve linear stretch in NC^0_3 as stated in the following theorem.

THEOREM 6.12. *There is a linear-stretch ε-biased generator in NC^0_3.*

Proof. Mossel, Shpilka, and Trevisan present an ε-biased generator in NC^0 with degree 2 and a linear stretch [43, Theorem 13].[17] Let G be their ε-biased genera-

[17] In fact, the generator of [43, Theorem 13] is in *nonuniform* NC^0_5 (and it has a slightly superlinear stretch). However, a similar construction gives an ε-biased generator in *uniform* NC^0 with degree 2 and linear stretch. (The locality of this generator is large but constant.) This can be done by replacing the probabilistic construction given in [43, Lemma 12] with a uniform construction of constant-degree bipartite expander with some "good" expansion properties—such a construction is given in [13, Theorem 7.1].

tor. We can apply the locality construction (4.16) to G (using concatenation) and get, by Lemma 6.9 and Observation 6.10, an ε-biased generator $\hat{G}$ in NC^0_3. We now relate the stretch of $\hat{G}$ to the stretch of G. Let $n, \hat{n}$ be the input complexity of $G, \hat{G}$, respectively, let $s, \hat{s}$ be the output complexity of $G, \hat{G}$, respectively, and let $c \cdot n$ be the stretch of G, where c is a constant. The generator $\hat{G}$ is stretch preserving; hence $\hat{s} - \hat{n} = s - n = c \cdot n$. Since G is in NC^0, each of its output bits can be represented as a polynomial that has a constant number of monomials and thus the locality construction adds only a constant number of random bits for each output bit of G. Therefore, the input length of $\hat{G}$ is linear in the input length of G. Hence, $\hat{s} - \hat{n} = s - n = c \cdot n = \hat{c} \cdot \hat{n}$ for some constant $\hat{c}$ and thus $\hat{G}$ has a linear stretch. ∎

Comparison with lower bounds. It is not hard to see that there is no ε-biased generator with degree 1 or locality 2.[18] In [16] it was shown that there is no superlinear-stretch ε-biased generator in NC^0_3. Thus, our linear-stretch NC^0_3 generator (building on the one from [43]) is not only optimal with respect to locality and degree but is also essentially optimal with respect to stretch.

6.3. Generators for space-bounded computation. We turn to the case of PRGs for space-bounded computation. A standard way of modeling a randomized space-bounded Turing machine is by having a random tape on which the machine can access the random bits one by one but cannot "go back" and view previous random bits (i.e., any bit that the machine wishes to remember, it must store in its limited memory). For the purpose of derandomizing such machines, it suffices to construct PRGs that fool any space-bounded distinguisher having a similar one-way access to its input. Following Babai, Nisan, and Szegedy [6], we refer to such distinguishers as *space-bounded distinguishers*.

DEFINITION 6.13 (space-bounded distinguisher [6]). *A space-$s(n)$ distinguisher is a deterministic Turing machine M, and an infinite sequence of binary strings $a = (a_1, \ldots, a_n, \ldots)$ called the advice strings, where $|a_n| = 2^{O(s(n))}$. The machine has the following tapes: read-write work tapes, a read-only advice tape, and a read-only input tape on which the tested input string, y, is given. The input tape has a one-way mechanism to access the tested string; namely, at any point it may request the next bit of y. In addition, only $s(n)$ cells of the work tapes can be used. Given an n-bit input, y, the output of the distinguisher, $M^a(y)$, is the (binary) output of M where y is given on the input tape and a_n is given on the advice tape.*

This class of distinguishers is a proper subset of the distinguishers that can be implemented by a space-$s(n)$ Turing machine with a two-way access to the input. Nevertheless, even logspace distinguishers are quite powerful, and many distinguishers fall into this category. In particular, this is true for the class of *linear* distinguishers considered in section 6.2.

DEFINITION 6.14 (PRG for space-bounded computation). *We say that a polynomial-time computable function $G : \{0,1\}^n \to \{0,1\}^{l(n)}$ is a PRG for space $s(n)$ if $l(n) > n$ and $G(U_n)$ is indistinguishable from $U_{l(n)}$ to any space-$s(n)$ distinguisher. That is, for every space-$s(n)$ distinguisher M^a, the distinguishing advantage $|\Pr[M^a(G(U_n)) = 1] - \Pr[M^a(U_{l(n)}) = 1]|$ is negligible in n.*

Several constructions of high-stretch PRGs for space-bounded computation exist in the literature (e.g., [6, 45]). In particular, a PRG for logspace computation from [6] can be computed using logarithmic space and thus, by Theorem 4.18, admits an

[18] A degree-1 generator contains more than n linear functions over n variables, which must be linearly dependent and thus biased. The nonexistence of a 2-local generator follows from the fact that every nonlinear function of two input bits is biased.

efficient perfect encoding in NC_4^0. It can be shown (see proof of Theorem 6.15) that this NC_4^0 encoding fools logspace distinguishers as well; hence, we can reduce the security of the randomized encoding to the security of the encoded generator and get an NC_4^0 PRG that fools logspace computation. However, as in the case of ε-biased generators, constructing such PRGs with a low stretch is much easier. In fact, the same "inner product" generator we used in section 6.2 can work here as well.

THEOREM 6.15. *There exists a (sublinear-stretch) PRG for sublinear-space computation in* NC_3^0.

Proof. Consider the inner product generator $G(x, x') = (x, x', \langle x, x' \rangle)$, where $x, x' \in \{0,1\}^n$. It follows from the average-case hardness of the inner product function for two-party communication complexity [14] that G fools all sublinear-space distinguishers. (Indeed, a sublinear-space distinguisher implies a sublinear-communication protocol predicting the inner product of x and x'. Specifically, the party holding x runs the distinguisher until it finishes reading x and then sends its configuration to the party holding x'.)

Applying the locality construction to G, we obtain a perfect encoding $\hat{G}$ in NC_3^0. (In fact, we can apply the locality construction only to the last bit of G and leave the other outputs as they are.) We argue that $\hat{G}$ inherits the pseudorandomness of G. As before, we would like to argue that if $\hat{M}$ is a sublinear-space distinguisher breaking $\hat{G}$ and S is the balanced simulator of the encoding, then $\hat{M}(S(\cdot))$ is a sublinear-space distinguisher breaking G. Similarly to the proof of Lemma 6.9, the fact that $\hat{M}(S(\cdot))$ can be implemented in sublinear space will follow from the simple structure of S. However, in contrast to Lemma 6.9, here it does not suffice to require S to be linear and we need to rely on the stronger property guaranteed by Observation 6.10.[19]

We now formalize the above. As argued in Observation 6.10, fixing the randomness ρ of S, the simulator's computation can be written as $S_\rho(y) = \sigma_1(y_1)\sigma_2(y_2)\cdots\sigma_l(y_l)$, where each σ_i maps a bit of y to one of two fixed strings. We can thus use S to turn a sublinear-space distinguisher $\hat{M}^a$ breaking $\hat{G}$ into a sublinear-space distinguisher $M^{a'}$ breaking G. Specifically, let the advice a' include, in addition to a, the $2l$ strings $\sigma_i(0), \sigma_i(1)$ corresponding to a "good" ρ which maintains the distinguishing advantage. (The existence of such ρ follows from an averaging argument.) The machine $M^{a'}(y)$ can now emulate the computation of $\hat{M}^a(S_\rho(y))$ using sublinear space and a one-way access to y by applying $\hat{M}^a$ in each step to the corresponding string $\sigma_i(y_i)$. ∎

6.4. Pseudorandom generators—Conclusion. We conclude this section with Table 6.1, which summarizes some of the PRGs constructed here as well as previous ones from [43] and highlights the remaining gaps.

7. Other cryptographic primitives. In this section, we describe extensions of our results to other cryptographic primitives. Aiming at NC^0 implementations, we can use our machinery in two different ways: (1) compile a primitive in a relatively high complexity class (say NC^1) into its randomized encoding and show that the encoding inherits the security properties of this primitive; or (2) use known *reductions* between cryptographic primitives, together with NC^0 primitives we already constructed (e.g.,

[19] Indeed, in the current model of (nonuniform) space-bounded computation with *one-way* access to the input (and two-way access to the advice), there exist a boolean function $\hat{M}$ computable in sublinear space and a linear function S such that the composed function $\hat{M}(S(\cdot))$ is not computable in sublinear space. For instance, let $\hat{M}(y_1, \ldots, y_{2n}) = y_1y_2 + y_3y_4 + \cdots + y_{2n-1}y_{2n}$ and $S(x_1, \ldots, x_{2n}) = (x_1, x_{n+1}, x_2, x_{n+2}, \ldots, x_n, x_{2n})$.

TABLE 6.1

Summary of known pseudorandom generators. Results of Mossel, Shpilka, and Trevisan [43] *appear in the top part and results of this paper in the bottom part. A parameter is marked as optimal (✓) if when fixing the other parameters it cannot be improved. A stretch entry is marked with ↚ if the stretch is sublinear and cannot be improved to be superlinear (but might be improved to be linear). The symbol * indicates a conditional result.*

Type	Stretch	Locality	Degree
ε-biased	superlinear	5	2 ✓
ε-biased	$n^{\Omega(\sqrt{k})}$	large k	$\Omega(\sqrt{k})$
ε-biased	$\Omega(n^2)$✓	$\Omega(n)$	2 ✓
ε-biased	linear ✓	3 ✓	2 ✓
ε-biased	sublinear ↚	3 ✓	2 ✓
Space	sublinear ↚	3 ✓	2 ✓
Cryptographic *	sublinear ↚	4	3

OWFs or PRGs), to obtain new NC^0 primitives. Of course, this approach is useful only when the reduction itself is in NC^0.[20] We mainly adopt the first approach, since most of the known reductions between primitives are not in NC^0. (An exception in the case of symmetric encryption will be discussed below.)

7.1. Collision-resistant hashing in NC^0. We start with a formal definition of collision-resistant hash-functions (CRHFs).

DEFINITION 7.1 (collision-resistant hashing). *Let $\ell, \ell' : \mathbb{N} \to \mathbb{N}$ be such that $\ell(n) > \ell'(n)$ and let $Z \subseteq \{0,1\}^*$. A collection of functions $\{h_z\}_{z \in Z}$ is said to be* collision-resistant *if the following hold:*

1. *There exists a probabilistic polynomial-time* key-generation *algorithm, G, that on input 1^n outputs an* index *$z \in Z$ (of a function h_z). The function h_z maps strings of length $\ell(n)$ to strings of length $\ell'(n)$.*
2. *There exists a polynomial-time* evaluation *algorithm that on input $z \in G(1^n)$, $x \in \{0,1\}^{\ell(n)}$ computes $h_z(x)$.*
3. *Collisions are hard to find. Formally, a pair (x, x') is called a* collision *for a function h_z if $x \neq x'$ but $h_z(x) = h_z(x')$. The collision-resistance requirement states that every probabilistic polynomial-time algorithm B that is given input $(z = G(1^n), 1^n)$ succeeds in finding a collision for h_z with a negligible probability in n (where the probability is taken over the coin tosses of both G and B).*

LEMMA 7.2. *Suppose that $\mathcal{H} = \{h_z\}_{z \in Z}$ is collision-resistant and $\hat{\mathcal{H}} = \{\hat{h}_z\}_{z \in Z}$ is a uniform perfect randomized encoding of $\mathcal{H}$. Then $\hat{\mathcal{H}}$ is also collision-resistant.*

Proof. Since $\hat{h}_z$ is stretch-preserving, it is guaranteed to shrink its input as h_z. The key-generation algorithm G of $\mathcal{H}$ is used as the key-generation algorithm of $\hat{\mathcal{H}}$. By the uniformity of the collection $\hat{\mathcal{H}}$, there exists an efficient evaluation algorithm for this collection. Finally, any collision $((x,r),(x',r'))$ under $\hat{h}_z$ (i.e., $(x,r) \neq (x',r')$ and $\hat{h}_z(x,r) = \hat{h}_z(x',r')$) defines a collision (x,x') under h_z. Indeed, perfect correctness ensures that $h_z(x) = h_z(x')$ and unique randomness (see Lemma 4.12) ensures that

[20] If the reduction is in NC^1, one can combine the two approaches: First apply the NC^1 reduction to an NC^0 primitive of type X that was already constructed (e.g., an OWF or a PRG) to obtain a new NC^1 primitive of type Y, and then use the first approach to compile the latter primitive into an NC^0 primitive (of type Y). As in the first approach, this construction requires one to prove that a randomized encoding of a primitive Y preserves its security.

$x \neq x'$. Thus, an efficient algorithm that finds collisions for $\hat{\mathcal{H}}$ with nonnegligible probability yields a similar algorithm for $\mathcal{H}$. □

By Lemma 7.2 and Theorem 4.18, we get the following theorem.

THEOREM 7.3. *If there exists a CRHF $\mathcal{H} = \{h_z\}_{z \in Z}$ such that the function $h'(z,x) \stackrel{\text{def}}{=} h_z(x)$ is in $\mathcal{PRECN}$ (in particular, in $\oplus$L/poly), then there exists a CRHF $\hat{\mathcal{H}} = \{\hat{h}_z\}_{z \in Z}$ such that the mapping $(z,y) \mapsto \hat{h}_z(y)$ is in NC^0_4.*

Using Theorem 7.3, we can construct CRHFs in NC^0 based on the intractability of factoring [17], discrete logarithm [46], or lattice problems [25, 48]. All these candidates are computable in NC^1 provided that some precomputation is done by the key-generation algorithm. Note that the key-generation algorithm of the resulting NC^0 CRHF is not in NC^0. For more details on NC^0 computation of collections of cryptographic primitives, see Appendix A.

7.2. Encryption in $\mathbf{NC^0}$. We turn to the case of encryption. Suppose that $\mathcal{E} = (G, E, D)$ is a public-key encryption scheme, where G is a key-generation algorithm, the encryption function $E(e, x, r)$ encrypts the message x using the key e and randomness r, and $D(d, y)$ decrypts the cipher y using the decryption key d. As usual, the functions G, E, D are polynomial-time computable, and the scheme provides correct decryption and satisfies indistinguishability of encryptions [29]. Let $\hat{E}$ be a randomized encoding of E, and let $\hat{D}(d, \hat{y}) \stackrel{\text{def}}{=} D(d, C(\hat{y}))$ be the composition of D with the decoder C of $\hat{E}$. We argue that the scheme $\hat{\mathcal{E}} \stackrel{\text{def}}{=} (G, \hat{E}, \hat{D})$ is also a public-key encryption scheme. The efficiency and correctness of $\hat{\mathcal{E}}$ are guaranteed by the uniformity of the encoding and its correctness. Using the efficient simulator of $\hat{E}$, we can reduce the security of $\hat{\mathcal{E}}$ to that of $\mathcal{E}$. Namely, given an efficient adversary $\hat{A}$ that distinguishes between encryptions of x and x' under $\hat{\mathcal{E}}$, we can break $\mathcal{E}$ by using the simulator to transform original ciphers into "new" ciphers, and then invoke $\hat{A}$. The same argument holds in the private-key setting. We now formalize this argument.

DEFINITION 7.4 (public-key encryption). *A* secure public-key encryption (PKE) scheme *is a triple (G, E, D) of probabilistic polynomial-time algorithms satisfying the following conditions:*

- Viability: *On input 1^n the key-generation algorithm, G, outputs a pair of keys (e, d). For every pair (e, d) such that $(e, d) \in G(1^n)$, and for every plaintext $x \in \{0,1\}^*$, the algorithms E, D satisfy*

$$\Pr[D(d, E(e, x)) \neq x] \leq \varepsilon(n),$$

  where $\varepsilon(n)$ is a negligible function and the probability is taken over the internal coin tosses of algorithms E and D.
- Security: *(Indistinguishability of encryptions of a single message.) For every (nonuniform) polynomial-time distinguisher B, every polynomial $p(\cdot)$, all sufficiently large n's, and every pair of plaintexts x, x' such that $|x| = |x'| \leq p(n)$, the distinguisher cannot distinguish between encryptions of x and x' with more than $\frac{1}{p(n)}$ advantage; namely,*

$$\left| \Pr_{(e,d) \leftarrow G(1^n)}[B(e, E(e, x)) = 1] - \Pr_{(e,d) \leftarrow G(1^n)}[B(e, E(e, x')) = 1] \right| \leq \frac{1}{p(n)},$$

  where the probabilities are taken over the coin tosses of G, E.

The definition of a *private-key* encryption scheme is similar, except that the distinguisher does not get the the encryption key e as an additional input. An extension

to multiple-message security, where the indistinguishability requirement should hold for encryptions of polynomially many messages, follows naturally (see [24, Chapter 5] for formal definitions). In the public-key case, multiple-message security is implied by single-message security as defined above, whereas in the private-key case it is a strictly stronger notion. In the following we explicitly address only the (single-message) public-key case, but the treatment easily holds for the case of private-key encryption with multiple-message security.

LEMMA 7.5. *Let $\mathcal{E} = (G, E, D)$ be a secure PKE scheme, where $E(e, x, r)$ is viewed as a polynomial-time computable function that encrypts the message x using the key e and randomness r. Let $\hat{E}((e, x), (r, s)) = \hat{E}((e, x, r), s)$ be a uniform statistical randomized encoding of E and let $\hat{D}(d, \hat{y}) \stackrel{\text{def}}{=} D(d, C(\hat{y}))$ be the composition of D with the decoder C of $\hat{E}$. Then, the scheme $\hat{\mathcal{E}} \stackrel{\text{def}}{=} (G, \hat{E}, \hat{D})$ is also a secure public-key encryption scheme.*

Proof. The uniformity of the encoding guarantees that the functions $\hat{E}$ and $\hat{D}$ can be efficiently computed. The viability of $\hat{\mathcal{E}}$ follows in a straightforward way from the correctness of the decoder C. Indeed, if (e, d) are in the support of $G(1^n)$, then for any plaintext x we have

$$\begin{aligned}\Pr_{r,s}[\hat{D}(d, \hat{E}(e, x, r, s)) \neq x] &= \Pr_{r,s}[D(d, C(\hat{E}(e, x, r, s))) \neq x] \\ &\leq \Pr_{r,s}[C(\hat{E}((e, x, r), s)) \neq E(e, x, r)] \\ &\quad + \Pr_{r}[D(d, E(e, x, r)) \neq x] \\ &\leq \varepsilon(n),\end{aligned}$$

where $\varepsilon(\cdot)$ is negligible in n and the probabilities are also taken over the coin tosses of D; the first inequality follows from the union bound and the second from the viability of $\mathcal{E}$ and the statistical correctness of $\hat{E}$.

We move on to prove the security of the construction. Assume, toward a contradiction, that $\hat{\mathcal{E}}$ is not secure. It follows that there exists an efficient (nonuniform) distinguisher $\hat{B}$ and a polynomial $p(\cdot)$, such that for infinitely many n's there exist two plaintexts x, x' such that $|x| = |x'| \leq p(n)$, and

$$\left| \Pr_{(e,d)\leftarrow G(1^n),r,s}[\hat{B}(e, \hat{E}(e, x, r, s)) = 1] - \Pr_{(e,d)\leftarrow G(1^n),r,s}[\hat{B}(e, \hat{E}(e, x', r, s)) = 1] \right| > \frac{1}{p(n)},$$

where r, s are uniformly chosen random strings of an appropriate length. We use $\hat{B}$ to construct a distinguisher B that distinguishes between encryptions of x and x' under E and derive a contradiction. Define a (nonuniform) distinguisher B by $B(e, y) \stackrel{\text{def}}{=} \hat{B}(e, S(y))$, where S is the efficient (statistical) simulator of $\hat{E}$. Then, for some negligible ε,

$$\begin{aligned}&\left| \Pr_{(e,d)\leftarrow G(1^n),r}[B(e, E(e, x, r)) = 1] - \Pr_{(e,d)\leftarrow G(1^n),r}[B(e, E(e, x', r)) = 1] \right| \\ &= \left| \Pr_{(e,d)\leftarrow G(1^n),r}[\hat{B}(e, S(E(e, x, r))) = 1] - \Pr_{(e,d)\leftarrow G(1^n),r}[B(e, S(E(e, x', r))) = 1] \right| \\ &\geq \left| \Pr_{(e,d)\leftarrow G(1^n),r,s}[\hat{B}(e, \hat{E}(e, x, r, s)) = 1] - \Pr_{(e,d)\leftarrow G(1^n),r,s}[\hat{B}(e, \hat{E}(e, x', r, s)) = 1] \right| - \varepsilon(n) \\ &> \frac{1}{p(n)} - \varepsilon(n) > \frac{1}{q(n)}\end{aligned}$$

for some polynomial $q(\cdot)$ and infinitely many n's. The first inequality is due to statistical privacy and the second follows from our hypothesis. Hence, we derive a contradiction to the security of $\mathcal{E}$, and the lemma follows. □

In particular, if the scheme $\mathcal{E} = (G, E, D)$ enables errorless decryption and the encoding $\hat{E}$ is perfectly correct, then the scheme $\hat{\mathcal{E}}$ also enables errorless decryption. Additionally, the above lemma is easily extended to the case of private-key encryption with multiple-message security. Thus we get the following theorem.

THEOREM 7.6. *If there exists a secure PKE scheme (resp., a secure private-key encryption scheme) $\mathcal{E} = (G, E, D)$, such that E is in $\mathcal{SREN}$ (in particular, in* NL*/poly), then there exists a secure PKE scheme (resp., a secure private-key encryption scheme) $\hat{\mathcal{E}} = (G, \hat{E}, \hat{D})$, such that $\hat{E}$ is in* NC^0_4.

Specifically, one can construct an NC^0 PKE based on either factoring [47, 28, 10], the Diffie–Hellman assumption [19, 28] or lattice problems [3, 48]. (These schemes enable an NC^1 encryption algorithm given a suitable representation of the key.)

On decryption in NC^0. Our construction provides an NC^0 encryption algorithm but does not promise anything regarding the parallel complexity of the decryption process. This raises the question of whether decryption can also be implemented in NC^0. In Appendix C.1, we argue that, in many settings, decryption in NC^0 is impossible regardless of the complexity of encryption. In contrast, if the scheme is restricted to a *single* message of a bounded length (even larger than the key) we can use our machinery to construct a private-key encryption scheme in which both encryption and decryption can be computed in NC^0. This can be done by using the output of an NC^0 PRG to mask the plaintext. Specifically, let $E(e, x) = G(e) \oplus x$ and $D(e, y) = y \oplus G(e)$, where e is a uniformly random key generated by the key-generation algorithm and G is a PRG. Unfortunately, the resulting scheme is severely limited by the low stretch of our PRGs. This approach can be also used to give multiple-message security, at the price of requiring the encryption and decryption algorithms to maintain a synchronized *state*. In such a stateful encryption scheme the encryption and decryption algorithms take an additional input and produce an additional output, corresponding to their state before and after the operation. The seed of the generator can be used, in this case, as the state of the scheme. In this setting, we can obtain multiple-message security by refreshing the seed of the generator in each invocation; e.g., when encrypting the current bit the encryption algorithm can randomly choose a new seed for the next session, encrypt it along with the current bit, and send this encryption to the receiver (alternatively, see [24, Construction 5.3.3]). In the resulting scheme both encryption and decryption are NC^0 functions whose inputs include the inner state of the algorithm.

Theorem 7.6 can be easily extended to stronger notions of security. In particular, randomized encoding preserves security against chosen plaintext attacks (CPA) as well as a priori chosen ciphertext attacks (CCA1). However, randomized encoding does not preserve security against a posteriori chosen ciphertext attacks (CCA2). Still, it can be shown that the encoding of a CCA2-secure scheme enjoys a relaxed security property that suffices for most applications of CCA2-security. See Appendix C.2 for further discussion.

7.3. Signatures, commitments, and zero-knowledge proofs. The construction that was used for encryption can be adapted to other cryptographic primitives including (noninteractive) commitments, signatures, message authentication schemes (MACs), and noninteractive zero-knowledge proofs (for definitions see [23, 24]). In all these cases, we can replace the sender (i.e., the encrypting party,

committing party, signer, or prover, according to the case) with its randomized encoding and let the receiver (the decrypting party or verifier) use the decoding algorithm to translate the output of the new sender to an output of the original one. The security of the resulting scheme reduces to the security of the original one by using the efficient simulator and decoder. In fact, such a construction can also be generalized to the case of interactive protocols such as zero-knowledge proofs and interactive commitments. As in the case of encryption discussed above, this transformation results in an NC^0 sender but does not promise anything regarding the parallel complexity of the receiver. An interesting feature of the case of commitment is that we can also improve the parallel complexity at the receiver's end (see below). The same holds for applications of commitment such as coin-flipping and zero-knowledge proofs. We now briefly sketch these constructions and their security proofs.

Signatures. Let $\mathcal{S} = (G, S, V)$ be a signature scheme, where G is a key-generation algorithm that generates the signing and verification keys (s, v), the signing function $S(s, \alpha, r)$ computes a signature β on the document α using the key s and randomness r, and the verification algorithm $V(v, \alpha, \beta)$ verifies that β is a valid signature on α using the verification key v. The scheme is secure (unforgeable) if it is infeasible to forge a signature in a chosen message attack. Namely, any polynomial-time adversary that gets the verification key and an oracle access to the signing process $S(s, \cdot)$ fails to produce a valid signature β on a document α (with respect to the corresponding verification key v) for which it has not requested a signature from the oracle. Let $\hat{S}$ be a statistical randomized encoding of S, and let $\hat{V}(v, \alpha, \hat{\beta}) \stackrel{\text{def}}{=} V(v, \alpha, C(\hat{\beta}))$ be the composition of V with the decoder C of the encoding $\hat{S}$. We claim that the scheme $\hat{\mathcal{S}} \stackrel{\text{def}}{=} (G, \hat{S}, \hat{V})$ is also a signature scheme. Given an adversary $\hat{A}$ that breaks $\hat{\mathcal{S}}$, we can break $\mathcal{S}$ by invoking $\hat{A}$ and emulating the oracle $\hat{S}$ using the simulator of the encoding and the signature oracle S. If the forged signature $(\alpha, \hat{\beta})$ produced by $\hat{A}$ is valid under $\hat{\mathcal{S}}$, then it is translated into a valid signature (α, β) under $\mathcal{S}$ by using the decoder, i.e., $\beta = C(\hat{\beta})$. A similar argument holds also in the private-key setting (i.e., in the case of MACs).

Commitments. A commitment scheme enables one party (a sender) to commit itself to a value while keeping it secret from another party (the receiver). Later, the sender can reveal the committed value to the receiver, and it is guaranteed that the revealed value is equal to the one determined at the commit stage. We start with the simple case of a perfectly binding, noninteractive commitment. Such a scheme can be defined by a polynomial-time computable function $\textsc{Send}(b, r)$ that outputs a commitment c to the bit b using the randomness r. We assume, without loss of generality, that the scheme has a canonical decommit stage in which the sender reveals b by sending b and r to the receiver, who verifies that $\textsc{Send}(b, r)$ is equal to the commitment c. The scheme should be both (computationally) hiding and (perfectly) binding. Hiding requires that $c = \textsc{Send}(b, r)$ keeps b computationally secret (as formalized in Definition 7.4 for the case of encryption). Binding means that it is impossible for the sender to open its commitment in two different ways; that is, there are no r_0 and r_1 such that $\textsc{Send}(0, r_0) = \textsc{Send}(1, r_1)$. Let $\hat{\textsc{Send}}(b, r, s)$ be some randomized encoding of $\textsc{Send}(b, r)$. It can be shown that if $\hat{\textsc{Send}}$ is a perfectly correct (and statistically private) encoding of $\textsc{Send}$, then $\hat{\textsc{Send}}$ defines a computationally hiding perfectly binding, noninteractive commitment: Hiding follows from the privacy of the encoding, as argued for the case of encryption in section 7.2. The binding property of $\hat{\textsc{Send}}$ follows from the perfect correctness; namely, given a cheating sender $\hat{S}^*$ for $\hat{\textsc{Send}}$ that produces ambiguous commitment $(r_0, r_0'), (r_1, r_1')$

such that $\hat{\text{SEND}}(0, r_0, s_0) = \hat{\text{SEND}}(1, r_1, s_1)$, we construct a cheating sender S^* for the original scheme that invokes $\hat{S}^*$ and outputs r_0, r_1. By perfect correctness it holds that $\text{SEND}(0, r_0) = \text{SEND}(1, r_1)$, and hence the new adversary succeeds with the same probability as the original one.[21]

Using a standard construction ([9], [23, Construction 4.4.2]), it follows that commitments in NC^0 are implied by the existence of a one-to-one OWF in $\mathcal{PREN}$. It is important to note that in contrast to the noninteractive perfectly binding primitives described so far, here we also improve the parallel complexity at the receiver's end. Indeed, on input $\hat{c}, b, r, s$ the receiver's computation consists of computing $\hat{\text{SEND}}(b, r, s)$ and comparing the result to $\hat{c}$. Assuming $\hat{\text{SEND}}$ is in NC^0, the receiver can be implemented by an NC^0 circuit augmented with a single (unbounded fan-in) AND gate. We refer to this special type of AC^0 circuit as an $\text{NC}^0[\text{AND}]$ circuit. As an immediate application, we get a 3-round protocol for flipping a coin [9] between an NC^0 circuit and an $\text{NC}^0[\text{AND}]$ circuit.

One can apply a similar transformation to other variants of commitment schemes, such as unconditionally hiding (and computationally binding) interactive commitments. Schemes of this type require some initialization phase, which typically involves a random key sent from the receiver to the sender. We can turn such a scheme into a similar scheme between an NC^0 sender and an $\text{NC}^0[\text{AND}]$ receiver, provided that it conforms to the following structure: (1) the receiver initializes the scheme by *locally* computing a random key k (say, a prime modulus and powers of two group elements for schemes based on a discrete logarithm) and sending it to the sender; (2) the sender responds with a single message computed by the commitment function $\text{SEND}(b, k, r)$ which is in $\mathcal{PREN}$ (actually, perfect correctness and statistical privacy suffice); (3) as in the previous case, the scheme has a canonical decommit stage in which the sender reveals b by sending b and r to the receiver, who verifies that $\text{SEND}(b, k, r)$ is equal to the commitment c. Using the CRHF-based commitment scheme of [18, 31], one can obtain schemes of the above type based on the intractability of factoring, discrete logarithm, and lattice problems. Given such a scheme, we replace the sender's function by its randomized encoding and get as a result an unconditionally hiding commitment scheme whose sender is in NC^0. The new scheme inherits the round complexity of the original scheme and thus consists of only two rounds of interaction. (The security proof is similar to the previous case of perfectly binding, noninteractive commitment.) If the random key k cannot be computed in $\text{NC}^0[\text{AND}]$ (as in the case of factoring- and discrete logarithm-based schemes), one can compute k once and for all during the generation of the receiver's circuit and hardwire the key to the receiver's circuit. (See Appendix A.)

Zero-knowledge proofs. We end this section by addressing the case of zero-knowledge protocols. Suppose that the prover's computations are in $\mathcal{SREN}$. Then, similarly to the case of encryption, we can compile the prover into its (statistical) randomized encoding and obtain a prover whose local computations (viewed as a function of its randomness, the common instance of the language, the private witness, and previously received messages) are in NC^0. The new verifier uses the decoder to translate the prover's encoded messages to the corresponding messages of original

[21] A modification of this scheme remains secure even if we replace SEND with a *statistical* randomized encoding. However, in this modification we cannot use the canonical decommitment stage. Instead, the receiver should verify the decommitment by applying the decoder C to $\hat{c}$ and comparing the result to the computation of the original sender; i.e., the receiver checks whether $C(\hat{c})$ equals $\text{SEND}(b, r)$. A disadvantage of this alternative decommitment is that it does not enjoy the enhanced parallelism feature discussed below.

TABLE 7.1
Sufficient properties for preserving the security of different primitives.

Primitive	Encoding	Efficient simulator	Efficient decoder
One-way function	statistical	required	—
One-way permutation	perfect	required	—
Trapdoor permutation	perfect	required	required
Pseudorandom generator	perfect	required	—
Collision-resistant hashing	perfect	—	—
Encryption (public, private)	statistical	required	required
Signatures, MAC	statistical	required	required
Commit + Decommit	perfectly correct	required	—
Zero-knowledge proof	statistical	required	required

protocol, and then invokes the original verifier. The completeness and soundness of the new protocol follow from the correctness of the encoding, and its zero-knowledge property from the privacy of the encoding. (The verifier can produce transcripts of the new protocol by composing the simulator of the encoding with the simulator of the original protocol.) A similar transformation applies to zero-knowledge *arguments*.

As before, this general approach does not parallelize the verifier; in fact, the verifier is now required to "work harder" and decode the prover's messages. However, we can improve the verifier's complexity by relying on specific, commitment-based, zero-knowledge protocols from the literature. For instance, in the constant-round protocol for graph 3-colorability of [26], the computations of the prover and the verifier consist of invoking two commitments (of both types, perfectly binding as well as statistically hiding), in addition to some AC^0 computations. Hence, we can use the parallel commitment schemes described before to construct a constant-round protocol for 3-colorability between an AC^0 prover and an AC^0 verifier. Since 3-colorability is NP complete under AC^0 reductions, we get constant-round zero-knowledge proofs in AC^0 for every language in NP.

7.4. Summary and discussion. Table 7.1 summarizes the properties of randomized encoding that suffice for encoding different cryptographic primitives. (In the case of TDPs, efficient randomness recovery is also needed.) We note that in some cases it suffices to use a *computationally private* randomized encoding, in which the simulator's output should only be computationally indistinguishable from that of the encoding. This relaxation, recently studied in [4], allows one to construct (some) primitives in NC^0 under more general assumptions.

The case of pseudorandom functions. It is natural to ask why our machinery cannot be applied to pseudorandom functions (PRFs) (assuming there exists a PRF in $\mathcal{PRE}\mathcal{N}$), as is implied from the impossibility results of Linial, Mansour, and Nisan [42]. Suppose that a PRF family $f_k(x) = f(k,x)$ is encoded by the function $\hat{f}(k,x,r)$. There are two natural ways to interpret $\hat{f}$ as a collection: (1) to incorporate the randomness into the key, i.e., $g_{k,r}(x) \stackrel{\text{def}}{=} \hat{f}(k,x,r)$; (2) to append the randomness to the argument of the collection, i.e., $h_k(x,r) \stackrel{\text{def}}{=} \hat{f}(k,x,r)$. To rule out the security of approach (1), it suffices to note that the mapping $\hat{f}(\cdot,r)$ is of degree 1 when r is fixed; thus, to distinguish $g_{k,r}$ from a truly random function, one can check whether the given function is affine (e.g., verify that $g_{k,r}(x) + g_{k,r}(y) = g_{k,r}(x+y) + g_{k,r}(0)$). The same attack applies to the function $h_k(x,r)$ obtained by the second approach, by fixing the randomness r. More generally, the privacy of a randomized encoding is guaranteed only when the randomness is secret and is freshly picked; thus our

methodology works well for cryptographic primitives which employ fresh secret randomness in each invocation. PRFs do not fit into this category: While the key contains secret randomness, it is not freshly picked in each invocation.

We finally note that by combining the positive results regarding the existence of various primitives in NC^0 with the negative results of [42] that rule out the possibility of PRFs in AC^0, one can derive a separation between PRFs and other primitives such as PRGs. In particular, we conclude that it is unlikely that a PRF is AC^0-reducible to a PRG.

8. One-way functions with optimal locality. The results presented so far leave a small gap between the strong positive evidence for cryptography in NC^0_4 and the known impossibility of even OWFs in NC^0_2. In this section we attempt to close this gap for the case of OWFs, providing positive evidence for the existence of OWFs in NC^0_3.

A natural approach for closing the gap would be to reduce the degree of our general construction of randomized encodings from 3 to 2. (Indeed, the locality construction transforms a degree-2 encoding into one in NC^0_3.) However, the results of [37] provide some evidence against the prospects of this general approach, ruling out the existence of degree-2 perfectly private encodings for most nontrivial functions. We thus take the following two alternative approaches: (1) seek *direct* constructions of degree-2 OWFs based on specific intractability assumptions; and (2) employ degree-2 randomized encodings with a weak (but nontrivial) privacy property (called *semiprivacy*), which enables the representation of general functions.

In section 8.1, we use approach (1) to construct an OWF with optimal locality based on the presumed intractability of decoding a random linear code. In section 8.2 we briefly demonstrate the usefulness of approach (2) by sketching a construction of an OWF with optimal locality based on an OWF that enjoys a certain strong "robustness" property, which is satisfied by a variant of an OWF candidate suggested in [22]. We note that neither of the above approaches yields a general result in the spirit of the results of the previous sections. Thus, we happen to pay for optimal degree and locality with the loss of generality.

8.1. OWFs in NC^0_3 from the intractability of decoding random linear codes. Several cryptographic schemes are based on hard problems from the theory of error-correcting codes. In particular, the problem of decoding random linear codes, which is a longstanding open question in coding theory, was suggested as a basis for OWFs [27]. An (n, k, δ) *binary linear code* is a k-dimensional linear subspace of $GF(2)^n$ in which the Hamming distance between each two distinct vectors (codewords) is at least δn. We refer to the ratio k/n as the *rate* of the code and to δ as its (relative) *distance*. Such a code can be defined by a $k \times n$ *generator matrix* whose rows span the space of codewords. It follows from the Gilbert–Varshamov bound that whenever $k/n < 1 - H_2(\delta) - \varepsilon$ (where H_2 is the binary entropy function and ε is an arbitrarily small positive constant), almost all $k \times n$ generator matrices form (n, k, δ)-linear codes.

Before defining our intractability assumption, imagine the following "decoding game." Let $k/n < 1 - H_2(1/3) - \varepsilon$ for some constant $\varepsilon > 0$. Pick a random $k \times n$ matrix C representing a linear code (which is with overwhelming probability an $(n, k, \frac{1}{3} + \varepsilon)$ code) and a random information word x. Encode x with C and transmit the resulting codeword $y = xC$ over a binary symmetric channel in which every bit is flipped with probability $\frac{1}{4}$. If more than $\frac{1}{3}$ of the bits were flipped, output the zero word; otherwise, output the noisy codeword $\tilde{y}$ along with the code's description C. In the

former event the adversary always wins (however, note that the probability of this event is negligible). In the latter event, the adversary's task is to find some codeword y which is at most $(n/3)$-far from $\tilde{y}$. The fact that the noise is random (rather than adversarial) guarantees, by Shannon's coding theorem, that y will be unique with overwhelming probability.

The intractability assumption on which we rely asserts that every polynomial-time adversary loses in the above game with noticeable probability. That is, roughly speaking, we assume that it is intractable to correct $n/4$ *random* errors in a random linear code of relative distance $\frac{1}{3}$. More precisely we have the following assumption.

INTRACTABILITY ASSUMPTION 8.1 (decoding a random linear code). *There exists a constant $c < 1 - H_2(\frac{1}{3})$ such that the following function f_{code} is a weak OWF:*[22]

$$f_{\text{code}}(C, x, e) \stackrel{\text{def}}{=} \begin{cases} 0 & \textit{weight}(e_1e_2, \ldots, e_{2n-1}e_{2n}) \geq n/3, \\ (C, xC + (e_1e_2, \ldots, e_{2n-1}e_{2n})) & \textit{otherwise}, \end{cases}$$

where C is a $k \times n$ binary generator matrix with $k = \lfloor cn \rfloor$, $x \in \{0,1\}^k$, $e \in \{0,1\}^{2n}$, weight$(\cdot)$ denotes Hamming weight, and arithmetic is over GF(2).

Namely, inverting f_{code} on a uniformly chosen input corresponds to winning in the above decoding game. (Two random bits, e_i and e_{i+1}, are multiplied to emulate a noise rate of $\frac{1}{4}$.) The plausibility of Assumption 8.1 is supported by the fact that a successful inverter would imply a major breakthrough in coding theory. Similar assumptions were put forward in [27, 8, 23]. It is possible to base our construction on different variants of this assumption (e.g., one in which the number of errors is bounded by half the minimal distance, as in [27]); the above formulation is preferred for simplicity (and seems even weaker than the one in [27]).

We now construct a degree-2 OWF assuming the (weak) one-wayness of f_{code}. Consider the degree-2 function f'_{code} defined by $f'_{\text{code}}(C, x, e) \stackrel{\text{def}}{=} (C, xC + (e_1e_2, \ldots, e_{2n-1}e_{2n}))$. The function f'_{code} by itself is not one-way; indeed, as there is no restriction on the choice of e, an inverter can arbitrarily pick x and then fix e to be consistent with C, x, and $\tilde{y}$. However, f'_{code} is still distributionally one-way. This follows by noting that f'_{code} differs from f_{code} only on a negligible fraction of their domain and by using Lemma 5.4. To conclude the proof we need the following lemma.

LEMMA 8.2. *A degree-2 distributional OWF implies a degree-2 OWF in* NC^0_3.

Proof. First observe that a degree-2 weak OWF can be transformed into a degree-2 (standard) OWF (cf. [52], [23, Theorem 2.3.2]). Combined with the locality construction, we get that the existence of a degree-2 weak OWF implies the existence of a degree-2 OWF in NC^0_3. Hence it is enough to show how to transform a degree-2 distributional OWF into a degree-2 weak OWF.

Let f be a degree-2 distributional OWF. Consider the function $F(x, i, h) = (f(x), h_i(x), i, h)$, where $x \in \{0,1\}^n$, $i \in \{1, \ldots, n\}$, $h : \{0,1\}^n \to \{0,1\}^n$ is a pairwise independent hash function, and h_i denotes the i-bit-long prefix of $h(x)$. This function was defined by Impagliazzo and Luby [35], who showed that in this case F is weakly one-way (see also [23, p. 96]). Note that $h(x)$ can be computed as a degree-2 function of x and (the representation of) h by using the hash family $h_{M,v}(x) = xM + v$, where M is an $n \times n$ matrix and v is a vector of length n. However, $h_i(x)$ is not of degree 2 when considered as a function of h, x, and i, since "chopping" the last $n - i$ bits of

[22] In fact, it seems likely that the function f_{code} is even strongly one-way.

$h(x)$ raises the degree of the function when i is not fixed. We get around this problem by applying n copies of F on independent inputs, where each copy uses a different i. Namely, we define the function $F'((x^{(i)}, h^{(i)})_{i=1}^n) \stackrel{\text{def}}{=} (F(x^{(i)}, i, h^{(i)}))_{i=1}^n$. Since each of the i's is now fixed, the resulting function F' can be computed by degree-2 polynomials over GF(2). Moreover, it is not hard to verify that F' is weakly one-way if F is weakly one-way. We briefly sketch the argument. Given an efficient inverting algorithm B for F', one can invert $y = F(x, i, h) = (f(x), h_i(x), i, h)$ as follows. For every $j \neq i$, uniformly and independently choose $x^{(j)}, h^{(j)}$, set $z_j = F(x^{(j)}, j, h^{(j)})$ and $z_i = y$, and then invoke B on $(z_j)_{j=1}^n$ and output the ith block of the answer. This inversion algorithm for F has the same success probability as B on a polynomially related input. □

Applying Lemma 8.2 to f'_{code} we get the following theorem.

THEOREM 8.3. *If Assumption* 8.1 *holds, there is a degree-*2 *OWF in* NC_3^0.

8.2. OWFs in NC_3^0 using semiprivate encoding. In this section we briefly address the possibility of obtaining optimal locality for OWFs (i.e., locality 3 rather than 4) by relaxing the privacy requirement of the encoding. Further details appear in [5].

We start by sketching an alternative approach for constructing OWFs in NC_3^0 based on Assumption 8.1. The basic idea is the following. Consider the degree-2 function f'_{code} defined above. This function is not one-way. However, it is possible to augment it to a (weakly) one-way function by appending to its output a single bit, $\phi(e)$, indicating whether the error vector e exceeds the weight threshold. That is, $\phi(e) = 1$ if and only if $weight(e_1e_2, \ldots, e_{2n-1}e_{2n}) \geq n/3$. (This ensures that, with high probability, the inverter will be forced to pick a low-weight error.) While we cannot encode the predicate $\phi(e)$ using degree-2 polynomials, it turns out that we can achieve this using the following type of *semiprivate* encoding. Specifically, we relax the simulation requirement to hold only when $\phi(e) = 0$. Thus, the encoding $\hat{\phi}(e, r)$ keeps e private only when $\phi(e) = 0$, i.e., when e defines a low-weight error vector. It is possible to efficiently construct such a degree-2 semiprivate encoding from the branching program representation of ϕ. (This can be done by using a variant of the BP construction described in section 4.3.) Hence, under Assumption 8.1, the degree-2 encoding $\hat{f}_{\text{code}}((C, x, e), r) \stackrel{\text{def}}{=} (f'_{\text{code}}(C, x, e), \hat{\phi}(e, r))$ is weakly one-way.

Given any OWF f, one could attempt to apply a semiprivate encoding as described above to every output bit of f, obtaining a degree-2 function $\hat{f}$. However, $\hat{f}$ will typically not be one-way: Every output bit of f that evaluates to 1 might reveal the entire input (through the corresponding block in the output of $\hat{f}$). This motivates the following notion of a *robust* OWF. Loosely speaking, an OWF f is said to be robust if it remains (slightly) hard to invert even if a random subset of its output bits are "exposed," in the sense that all input bits leading to these outputs are revealed. Intuitively, the purpose of the robustness requirement is to guarantee that the information leaked by the semiprivate encoding leaves enough uncertainty about the input to make inversion difficult. It can be shown that (1) every robust OWF with a low locality (say, logarithmic in the number of inputs) can be turned into an OWF in NC_3^0; and (2) a variant of an OWF candidate from [22] satisfies the latter property, assuming that it is indeed one-way. Thus, an intractability assumption of the flavor of the one suggested in [22] implies the existence of OWFs in NC_3^0.

9. Conclusions and open problems. Our results provide strong evidence for the possibility of cryptography in NC^0. They are also close to optimal in terms of the

exact locality that can be achieved. Still, several questions are left for further study, in particular, the following:

- What are the minimal assumptions required for cryptography in NC^0? For instance, does the existence of an arbitrary OWF imply the existence of an OWF in NC^0? We show that an OWF in NL/*poly* implies an OWF in NC^0.
- Is there a PRG with linear stretch or even superlinear stretch in NC^0? In particular, is there a PRG with linear stretch in NC^0_4? (The possibility of a PRG with superlinear stretch in NC^0_4 is ruled out in [43].) We show that there exists a PRG with *sublinear* stretch in NC^0_4, assuming the existence of a PRG in $\oplus$L/*poly*.
- Can the existence of an OWF (or PRG) in NC^0_3 be based on more general assumptions? We construct such an OWF under the intractability of decoding a random linear code.
- Is it possible to obtain constant *input* locality, i.e., construct primitives in which each input influences only a constant number of outputs? (A candidate OWF of this type is given in [22].) Note that the results of this work only address the case of a constant *output* locality, which does not imply a constant input locality.
- Can our paradigm for achieving better parallelism be of any practical use?

The above questions motivate a closer study of the complexity of randomized encodings, which so far was only motivated by questions in the domain of secure multiparty computation. In [4] we continue this study by considering a relaxed variant of randomized encoding referred to as *computationally private* encoding. We show that, under relatively mild assumptions, one can encode every polynomial-time computable function by a computationally private encoding in NC^0. This gives new sufficient conditions for cryptography in NC^0, as well as new NC^0 reductions between different cryptographic primitives.

Appendix A. On collections of cryptographic primitives.

In most cases, we view a cryptographic primitive (e.g., an OWF or a PRG) as a single function $f : \{0,1\}^* \to \{0,1\}^*$. However, it is often useful to consider more general variants of such primitives, defined by a *collection* of functions $\{f_z\}_{z\in Z}$, where $Z \subseteq \{0,1\}^*$ and each f_z is defined over a finite domain D_z. The full specification of such a collection usually consists of a probabilistic polynomial-time key-generation algorithm that chooses an index z of a function (given a security parameter 1^n), a domain sampler algorithm that samples a random element from D_z given z, and a function evaluation algorithm that computes $f_z(x)$ given z and $x \in D_z$. The primitive should be secure with respect to the distribution defined by the key-generation and the domain sampler. (See a formal definition for the case of OWFs in [23, Definition 2.4.3].)

Collections of primitives arise naturally in the context of parallel cryptography, as they allow one to shift "nonparallelizable" operations such as prime number selection and modular exponentiations to the key-generation stage (cf. [44]). They also fit naturally into the setting of P-uniform circuits, since the key-generation algorithm can be embedded in the algorithm generating the circuit. Thus, it will be convenient to assume that z is a description of a circuit computing f_z. When referring to a collection of functions from a given complexity class (e.g., NC^1, NC^0_4, or $\mathcal{PRENC}$, cf. Definition 4.8) we assume that the key-generation algorithm outputs a description of a circuit from this class. In fact, one can view collections in our context as a natural relaxation of uniformity, allowing the circuit generator to be randomized.

(The above discussion also applies to other P-uniform representation models we use, such as branching programs.)

Our usage of collections differs from the standard one in that we insist on D_z being the set of *all* strings of a given length (i.e., the set of all possible inputs for the circuit z) and restrict the domain sampler to be a trivial one which outputs a uniformly random string of the appropriate length. This convention guarantees that the primitive can indeed be invoked with the specified parallel complexity, and does not implicitly rely on a (possibly less parallel) domain sampler.[23] In most cases, it is possible to modify standard collections of primitives to conform to the above convention. We illustrate this by outlining a construction of an NC^1 collection of OWPs based on the intractability of the discrete logarithm problem. The key-generator, on input 1^n, samples a random prime p such that $2^{n-1} \leq p < 2^n$ along with a generator g of Z_p^*, and lets z be a description of an NC^1 circuit computing the function $f_{p,g}$ defined as follows. On an n-bit input x (viewed as an integer such that $0 \leq x < 2^n$) define $f_{p,g}(x) = g^x \bmod p$ if $1 \leq x < p$ and $f_{p,g}(x) = x$ otherwise. It is easy to verify that $f_{p,g}$ indeed defines a permutation on $\{0,1\}^n$. Moreover, it can be computed by an NC^1 circuit by incorporating $p, g, g^2, g^4, \ldots, g^{2^n}$ into the circuit. Finally, assuming the intractability of the discrete logarithm problem, the above collection is *weakly* one-way. It can be augmented into a collection, of (strongly) OWPs by using the standard reduction of strong OWFs to weak OWFs (i.e., using $f'_{p,g}(x_1, \ldots, x_n) = (f_{p,g}(x_1), \ldots, f_{p,g}(x_n))$).

When defining the cryptographic security of a collection of primitives, it is assumed that the adversary (e.g., inverter or distinguisher) is given the key z, in addition to its input in the single-function variant of the primitive. Here one should make a distinction between "private-coin collections," where this is all of the information available to the adversary, and "public-coin collections" in which the adversary is additionally given the internal coin tosses of the key-generator. (A similar distinction has recently been made in the specific context of CRHFs [34]; also, see the discussion of "enhanced TDP" in [24, Appendix C.1].) The above example for an OWP collection is of the public-coin type. Any public-coin collection is also a private-coin collection, but the converse may not be true.

Summarizing, we consider cryptographic primitives in three different settings:

1. *Single function setting.* The circuit family $\{C_n\}_{n\in\mathbb{N}}$ that computes the primitive is constructed by a deterministic polynomial time circuit generator that, given an input 1^n, outputs the circuit C_n. This is the default setting for most cryptographic primitives.
2. *Public-coin collection.* The circuit generator is a probabilistic polynomial time algorithm that, on input 1^n, samples a circuit from a collection of circuits. The adversary gets as an input the circuit produced by the generator, along with the randomness used to generate it. The experiments defining the success probability of the adversary incorporate the randomness used by the generator, in addition to the other random variables. As in the single function setting, this generation step can be thought of as being done "once and for all," e.g., in a preprocessing stage. Public-coin collections are typically useful for primitives based on discrete logarithm assumptions, where a large prime group should be set up along with its generator and precomputed exponents of the generator.

[23] Note that unlike the key-generation algorithm, which can be applied "once and for all," the domain sampler should be invoked for each application of the primitive.

3. *Private-coin collection.* This is the same as (2) except that the adversary does not know the randomness that was used by the circuit generator. This relaxation is typically useful for factoring-based constructions, where the adversary should not learn the trapdoor information associated with the public modulus (see [39, 44]).

We note that our general transformations apply to all of the above settings. In particular, given an NC^1 primitive in any of these settings, we obtain a corresponding NC^0 primitive in the same setting.

Appendix B. A generalization of the locality construction. In the locality construction (Construction 4.16), we showed how to encode a degree-d function by an NC^0_{d+1} encoding. We now describe a graph-based construction that generalizes the previous one. The basic idea is to view the encoding $\hat{f}$ as a graph. The nodes of the graph are labeled by terms of f and the edges by random inputs of $\hat{f}$. With each node we associate an output of $\hat{f}$ in which we add to its term the labels of the edges incident to the node. Formally, we have the following construction.

CONSTRUCTION B.1 (general locality construction). *Let* $f(x) = T_1(x) + \cdots + T_k(x)$, *where* $f, T_1, \ldots, T_k : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ *and summation is over* $\mathrm{GF}(2)$. *Let* $G = (V, E)$ *be a directed graph with* k *nodes* $V = \{1, \ldots, k\}$ *and* m *edges. The encoding* $\hat{f}_G : \mathrm{GF}(2)^{n+m} \to \mathrm{GF}(2)^k$ *is defined by*

$$\hat{f}_G(x, (r_{i,j})_{(i,j)\in E}) \stackrel{\text{def}}{=} \left(T_i(x) + \sum_{j|(j,i)\in E} r_{j,i} - \sum_{j|(i,j)\in E} r_{i,j} \right)_{i=1}^{k}.$$

From here on, we will identify with the directed graph G its underlying undirected graph. The above construction yields a perfect encoding when G is a tree (see Lemma B.2). The locality of an output bit of $\hat{f}_G$ is the locality of the corresponding term plus the degree of the node in the graph. The locality construction described in Construction 4.16 attempts to minimize the maximal locality of a node in the graph; hence it adds k "dummy" 0 terms to f and obtains a tree in which all of the k nondummy terms of f are leaves and the degree of each dummy term is at most 3. When the terms of f vary in their locality, a more compact encoding $\hat{f}$ can be obtained by increasing the degree of nodes which represent terms with lower locality.

LEMMA B.2 (generalized locality lemma). *Let* f *and* $\hat{f}_G$ *be as in Construction* B.1. *Then, the following hold:*

(1) $\hat{f}_G$ is a perfectly correct encoding of f.
(2) If G is connected, then $\hat{f}_G$ is also a balanced encoding of f (and in particular it is perfectly private).
(3) If G is a tree, then $\hat{f}_G$ is also stretch-preserving; that is, $\hat{f}_G$ perfectly encodes f.

Proof. (1) Given $\hat{y} = \hat{f}_G(x, r)$, we decode $f(x)$ by summing up the bits of $\hat{y}$. Since each random variable $r_{i,j}$ appears only in the ith and jth output bits, it contributes 0 to the overall sum and therefore the bits of $\hat{y}$ always add up to $f(x)$.

To prove (2) we use the same simulator as in the locality construction (see proof of Lemma 4.17). Namely, given $y \in \{0, 1\}$, the simulator S chooses $k-1$ random bits $r_1, \ldots, r_{k-1}$ and outputs $(r_1, \ldots, r_{k-1}, y-(r_1+\cdots+r_{k-1}))$. This simulator is balanced since the supports of $S(0)$ and $S(1)$ halve $\{0, 1\}^k$ and $S(y)$ is uniformly distributed over its support for $y \in \{0, 1\}$. We now prove that $\hat{f}_G(x, U_m) \equiv S(f(x))$. Since the support

of $S(f(x))$ contains exactly 2^{k-1} strings (namely, all k-bit strings whose bits sum up to $f(x)$), it suffices to show that for any input x and output $w \in \text{support}(S(f(x)))$ there are $2^m/2^{k-1}$ random inputs r such that $\hat{f}_G(x,r) = w$. (Note that $m \geq k-1$ since G is connected.) Let $T \subseteq E$ be a spanning tree of G. We argue that for any assignment to the $m-(k-1)$ random variables that correspond to edges in $E \setminus T$ there exists an assignment to the other random variables that is consistent with w and x. Fix some assignment to the edges in $E \setminus T$. We now recursively assign values to the remaining edges. In each step we make sure that some leaf is consistent with w by assigning the corresponding value to the edge connecting this leaf to the graph. Then, we prune this leaf and repeat the above procedure. Formally, let i be a leaf which is connected to T by an edge $e \in T$. Assume, without loss of generality, that e is an incoming edge for i. We set r_e to $w_i - (T_i(x) + \sum_{j|(j,i)\in E\setminus T} r_{j,i} - \sum_{j|(i,j)\in E\setminus T} r_{i,j})$ and remove i from T. By this we ensure that the ith bit of $\hat{f}_G(x,r)$ is equal to w_i. (This equality will not be violated by the following steps as i is removed from T.) We continue with the above step until the tree consists of one node. Since the outputs of $\hat{f}_G(x,r)$ always sum up to $f(x)$ it follows that this last bit of $\hat{f}_G(x,r)$ is equal to the corresponding bit of w. Thus, there are at least $2^{|E\setminus T|} = 2^{m-(k-1)}$ values of r that lead to w as required.

Finally, to prove (3) note that when G is a tree we have $m = k-1$, and therefore the encoding is stretch-preserving; combined with (1) and (2), $\hat{f}_G$ is also perfect. ∎

Appendix C. More on encryption schemes in $\mathbf{NC^0}$. We consider two issues regarding encryption, briefly mentioned in section 7.2.

C.1. On the impossibility of $\mathbf{NC^0}$ decryption. In this section we show that, in many settings, decryption in NC^0 is impossible regardless of the complexity of encryption. Here we consider standard *stateless* encryption schemes in contrast to the discussion at the end of section 7.2. We begin with the case of multiple-message security (in either the private-key or public-key setting). If a decryption algorithm $D(d,y)$ is in NC^0_k, then an adversary that gets n encrypted messages can correctly guess the first bits of *all* the plaintexts (jointly) with at least 2^{-k} probability. To do so, the adversary simply guesses at random the k (or fewer) bits of the key d on which the first output bit of D depends, and then computes this first output bit (which is supposed to be the first plaintext bit) on each of the n ciphertexts using the subkey it guessed. Whenever the adversary guesses the k bits correctly, it succeeds in finding the first bits of *all* n messages. When $n > k$, this violates the semantic security of the encryption scheme. Indeed, for the encryption scheme to be secure, the adversary's success probability (when the messages are chosen at random) can only be negligibly larger than 2^{-n}. (That is, an adversary cannot do much better than simply guessing these first bits.)

Even in the case of a single-message private-key encryption, it is impossible to implement decryption in NC^0_k with an arbitrary (polynomial) message length. Indeed, when the message length exceeds $(2|d|)^k$ (where $|d|$ is the length of the decryption key), there must be more than 2^k bits of the output of D which depend on the same k bits of the key, in which case we are in the same situation as before. That is, we can guess the value of more than 2^k bits of the message with constant success probability 2^{-k}. Again, if we consider a randomly chosen message, this violates semantic security.

C.2. Security against CPA, CCA1, and CCA2 attacks. In this section we address the possibility of applying our machinery to encryption schemes that enjoy

stronger notions of security. In particular, we consider schemes that are secure against CPA, CCA1, and CCA2. In all three attacks the adversary has to win the standard indistinguishability game (i.e., given a ciphertext $c = E(e, m_b)$, find out which of the two predefined plaintexts m_0, m_1 was encrypted), and so the actual difference lies in the power of the adversary. In a CPA attack the adversary can obtain encryptions of plaintexts of his choice (under the key being attacked), i.e., the adversary gets an oracle access to the encryption function. In CCA1 attack the adversary may also obtain decryptions of his choice (under the key being attacked), but he is allowed to do so only *before* the challenge is presented to him. In both cases, the security is preserved under randomized encoding. We briefly sketch the proof idea.

Let $\hat{B}$ be an adversary that breaks the encoding $\hat{\mathcal{E}}$ via a CPA attack (resp., CCA1 attack). We use $\hat{B}$ to obtain an adversary B that breaks the original scheme $\mathcal{E}$. As in the proof of Lemma 7.5, B uses the simulator to translate the challenge c, an encryption of the message m_b under $\mathcal{E}$, into a challenge $\hat{c}$, which is an encryption of the same message under $\hat{\mathcal{E}}$. Similarly, B answers the encryption queries of $\hat{B}$ (to the oracle $\hat{E}$) by directing these queries to the oracle E and applying the simulator to the result. Also, in the case of CCA1 attack, whenever $\hat{B}$ asks the decryption oracle $\hat{D}$ to decrypt some ciphertext $\hat{c}'$, the adversary B uses the decoder (of the encoding) to translate $\hat{c}'$ into a ciphertext c' of the same message under the scheme $\mathcal{E}$, and then uses the decryption oracle D to decrypt c'. This allows B to emulate the oracles $\hat{D}$ and $\hat{E}$ and thus to translate a successful CPA attack (resp., CCA1 attack) on the new scheme into a similar attack on the original scheme.

The situation is different in the case of a CCA2 attack. As in the case of a CCA1 attack, a CCA2 attacker has an oracle access to the decryption function corresponding to the decryption key in use; however, the adversary can query the oracle *even after* the challenge has been given to him, under the restriction that he cannot ask the oracle to decrypt the challenge c itself.

We start by observing that when applying a randomized encoding to a CCA2-secure encryption scheme, CCA2 security may be lost. Indeed, in the resulting encryption one can easily modify a given ciphertext challenge $\hat{c} = \hat{E}(e, x, r)$ into a ciphertext $\hat{c}' \neq \hat{c}$ which is also an encryption of the same message under the same encryption key. This can be done by applying the decoder (of the randomized encoding $\hat{E}$) and then the simulator on $\hat{c}$, that is $\hat{c}' = S(C(\hat{c}))$. Hence, one can break the encryption by simply asking the decryption oracle to decrypt $\hat{c}'$.

It is instructive to understand why the previous arguments fail to generalize to the case of CCA2 security. In the case of CCA1 attacks we transformed an adversary $\hat{B}$ that breaks the encoding $\hat{\mathcal{E}}$ into an adversary B for the original scheme in the following way: (1) we used the simulator to convert a challenge $c = E(e, m_b)$ into a challenge $\hat{c}$ which is an encryption of the same message under $\hat{\mathcal{E}}$; (2) when $\hat{B}$ asks $\hat{D}$ to decrypt a ciphertext $\hat{c}'$, the adversary B uses the decoder (of the encoding) to translate $\hat{c}'$ into a ciphertext c' of the same message under the scheme $\mathcal{E}$, and then asks the decryption oracle D to decrypt c'. However, recall that in a CCA2 attack the adversaries are not allowed to ask the oracle to decrypt the challenge itself (after the challenge is presented). So if $c' = c$ but $\hat{c}' \neq \hat{c}$, the adversary B cannot answer the (legitimate) query of $\hat{B}$.

To complement the above, we show that when applying a randomized encoding to a CCA2-secure encryption scheme not all is lost. Specifically, the resulting scheme still satisfies *replayable CCA (RCCA) security*, a relaxed variant of CCA2 security that was suggested in [12]. Loosely speaking, RCCA security captures encryption schemes that are CCA2 secure except that they allow anyone to generate new ciphers

that decrypt to the same value as a given ciphertext. More precisely, an RCCA attack is a CCA2 attack in which the adversary cannot ask the oracle to decrypt *any* cipher c' that decrypts to either m_0 or m_1 (cf. [12, Figure 3]). This limitation prevents the problem raised in the CCA2 proof, in which a legitimate query for $\hat{D}$ translates by the decoder into an illegitimate query for D. That is, if $\hat{c}'$ does not decrypt under $\hat{\mathcal{E}}$ to either m_0 or m_1, then (by correctness) the ciphertext c' obtained by applying the decoder to $\hat{c}'$ also does not decrypt to either of these messages. Hence, randomized encoding preserves RCCA security. As argued in [12], RCCA security suffices in most applications of CCA2 security.

Acknowledgments. We are grateful to Oded Goldreich for many useful suggestions and comments that helped improve this paper and, in particular, for simplifying the proof of Lemma 5.4. We also thank Iftach Haitner and Emanuele Viola for enlightening us about old and new constructions of PRGs from OWFs and for sharing with us the results of [30] and [50]. Finally, we thank Moni Naor and Amir Shpilka for helpful comments.

REFERENCES

[1] M. Agrawal, E. Allender, and S. Rudich, *Reductions in circuit complexity: An isomorphism theorem and a gap theorem*, J. Comput. System Sci., 57 (1998), pp. 127–143.

[2] M. Ajtai, *Generating hard instances of lattice problems*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC), 1996, pp. 99–108; full version in Electronic Colloquium on Computational Complexity (ECCC).

[3] M. Ajtai and C. Dwork, *A public-key cryptosystem with worst-case/average-case equivalence*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC), 1997, pp. 284–293.

[4] B. Applebaum, Y. Ishai, and E. Kushilevitz, *Computationally private randomizing polynomials and their applications*, in Proceedings of the 20th Annual IEEE Conference on Computational Complexity (CCC), 2005, pp. 260–274.

[5] B. Applebaum, Y. Ishai, and E. Kushilevitz, *On One-way Functions with Optimal Locality*, unpublished manuscript, 2005; available online at http://www.cs.technion.ac.il/∼abenny.

[6] L. Babai, N. Nisan, and M. Szegedy, *Multiparty protocols and logspace-hard pseudorandom sequences*, in Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC), 1989, pp. 1–11.

[7] D. A. Barrington, *Bounded-width polynomial-size branching programs recognize exactly those languages in* NC^1, in Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC), 1986, pp. 1–5.

[8] A. Blum, M. Furst, M. Kearns, and R. J. Lipton, *Cryptographic primitives based on hard learning problems*, in Advances in Cryptology—CRYPTO '93, Lecture Notes in Comput. Sci. 773, Springer-Verlag, Berlin, 1994, pp. 278–291.

[9] M. Blum, *Coin flipping by telephone: A protocol for solving impossible problems*, SIGACT News, 15 (1983), pp. 23–27.

[10] M. Blum and S. Goldwasser, *An efficient probabilistic public-key encryption scheme which hides all partial information*, in Advances in Cryptology—CRYPTO '84, Lecture Notes in Comput. Sci. 196, Springer-Verlag, Berlin, 1985, pp. 289–299.

[11] M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudo-random bits*, SIAM J. Comput., 13 (1984), pp. 850–864.

[12] R. Canetti, H. Krawczyk, and J. B. Nielsen, *Relaxing chosen ciphertext security*, in Advances in Cryptology—CRYPTO '03, Lecture Notes in Comput. Sci. 2729, Springer-Verlag, Berlin, 2003, pp. 565–582.

[13] M. Capalbo, O. Reingold, S. Vadhan, and A. Wigderson, *Randomness conductors and constant-degree lossless expanders*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC), 2002, pp. 659–668.

[14] B. Chor and O. Goldreich, *Unbiased bits from sources of weak randomness and probabilistic communication complexity*, SIAM J. Comput., 17 (1988), pp. 230–261.

[15] R. Cramer, S. Fehr, Y. Ishai, and E. Kushilevitz, *Efficient multi-party computation over rings*, in Advances in Cryptology—EUROCRYPT '03, Springer-Verlag, Berlin, 2003, pp. 596–613.

[16] M. Cryan and P. B. Miltersen, *On pseudorandom generators in* NC^0, in Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 2136, Springer-Verlag, Berlin, 2001, pp. 272–284.

[17] I. B. Damgård, *Collision free hash functions and public key signature schemes*, in Advances in Cryptology—EUROCRYPT '87, Lecture Notes in Comput. Sci. 304, Springer-Verlag, Berlin, 1988, pp. 203–216.

[18] I. B. Damgård, T. P. Pedersen, and B. Pfitzmann, *On the existence of statistically hiding bit commitment schemes and fail-stop signatures*, J. Cryptology, 10 (1997), pp. 163–194.

[19] T. ElGamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Trans. Inform. Theory, 31 (1985), pp. 469–472.

[20] A. V. Goldberg, M. Kharitonov, and M. Yung, *Lower bounds for pseudorandom number generators*, in Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1989, pp. 242–247.

[21] O. Goldreich, *Modern Cryptography, Probabilistic Proofs, and Pseudorandomness*, Algorithms Combin. 17, Springer-Verlag, Berlin, 1999.

[22] O. Goldreich, *Candidate one-way functions based on expander graphs*, Electronic Colloquium on Computational Complexity (ECCC), 7 (2000).

[23] O. Goldreich, *Foundations of Cryptography: Basic Tools*, Cambridge University Press, Cambridge, UK, 2001.

[24] O. Goldreich, *Foundations of Cryptography: Basic Applications*, Cambridge University Press, Cambridge, UK, 2004.

[25] O. Goldreich, S. Goldwasser, and S. Halevi, *Collision-free hashing from lattice problems*, Electronic Colloquium on Computational Complexity, 96 (1996).

[26] O. Goldreich and A. Kahan, *How to construct constant-round zero-knowledge proof systems for* NP, J. Cryptology, 9 (1996), pp. 167–189.

[27] O. Goldreich, H. Krawczyk, and M. Luby, *On the existence of pseudorandom generators*, SIAM J. Comput., 22 (1993), pp. 1163–1175.

[28] O. Goldreich and L. Levin, *A hard-core predicate for all one-way functions*, in Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC), 1989, pp. 25–32.

[29] S. Goldwasser and S. Micali, *Probabilistic encryption*, J. Comput. System Sci., 28 (1984), pp. 270–299.

[30] I. Haitner, D. Harnik, and O. Reingold, *On the Power of the Randomized Iterate*, Tech. report, ECCC TR05-135, 2005.

[31] S. Halevi and S. Micali, *Practical and provably-secure commitment schemes from collision-free hashing*, in Advances in Cryptology—CRYPTO '96, Lecture Notes in Comput. Sci. 1109, Springer-Verlag, Berlin, 1996, pp. 201–215.

[32] J. Håstad, *One-way permutations in* NC^0, Inform. Process. Lett., 26 (1987), pp. 153–155.

[33] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby, *A pseudorandom generator from any one-way function*, SIAM J. Comput., 28 (1999), pp. 1364–1396.

[34] C. Y. Hsiao and L. Reyzin, *Finding collisions on a public road, or do secure hash functions need secret coins?*, in Advances in Cryptology—CRYPTO '04, Lecture Notes in Comput. Sci. 3152, Springer-Verlag, Berlin, 2004, pp. 92–105.

[35] R. Impagliazzo and M. Luby, *One-way functions are essential for complexity based cryptography*, in Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1989, pp. 230–235.

[36] R. Impagliazzo and M. Naor, *Efficient cryptographic schemes provably as secure as subset sum*, J. Cryptology, 9 (1996), pp. 199–216.

[37] Y. Ishai and E. Kushilevitz, *Randomizing polynomials: A new representation with applications to round-efficient secure computation*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2000, pp. 294–304.

[38] Y. Ishai and E. Kushilevitz, *Perfect constant-round secure computation via perfect randomizing polynomials*, in Automata, Languages and Programming, Lecture Notes in Comput. Sci. 2380, Springer-Verlag, Berlin, 2002, pp. 244–256.

[39] M. Kharitonov, *Cryptographic hardness of distribution-specific learning*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC), 1993, pp. 372–381.

[40] J. Kilian, *Founding cryptography on oblivious transfer*, in Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC), 1988, pp. 20–31.

[41] M. Krause and S. Lucks, *On the minimal hardware complexity of pseudorandom function generators* (extended abstract), in STACS 2001, Lecture Notes in Comput. Sci. 2010, Springer-Verlag, Berlin, 2001, pp. 419–430.

[42] N. Linial, Y. Mansour, and N. Nisan, *Constant depth circuits, Fourier transform, and learnability*, J. ACM, 40 (1993), pp. 607–620.

[43] E. Mossel, A. Shpilka, and L. Trevisan, *On ϵ-biased generators in* NC^0, in Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2003, pp. 136–145.

[44] M. Naor and O. Reingold, *Number-theoretic constructions of efficient pseudo-random functions*, J. ACM, 51 (2004), pp. 231–262.

[45] N. Nisan, *Pseudorandom generators for space-bounded computation*, Combinatorica, 12 (1992), pp. 449–461.

[46] T. Pedersen, *Noninteractive and information-theoretic secure verifiable secret sharing*, in Advances in Cryptology—CRYPTO '91, Lecture Notes in Comput. Sci. 576, Springer-Verlag, Berlin, 1991, pp. 129–149.

[47] M. O. Rabin, *Digitalized Signatures and Public Key Functions as Intractable as Factoring*, Tech. report 212, Laboratory for Computer Science, MIT, 1979.

[48] O. Regev, *New lattice based cryptographic constructions*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC), 2003, pp. 407–416.

[49] R. L. Rivest, A. Shamir, and L. M. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Comm. ACM, 21 (1978), pp. 120–126.

[50] E. Viola, *On constructing parallel pseudorandom generators from one-way functions*, in Proceedings of the 20th Annual IEEE Conference on Computational Complexity (CCC), 2005, pp. 183–197.

[51] A. Wigderson, $\mathrm{NL}/poly \subseteq \oplus\mathrm{L}/poly$, in Proceedings of the 9th Annual Structure in Complexity Theory Conference, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 59–62.

[52] A. C. Yao, *Theory and application of trapdoor functions*, in Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1982, pp. 80–91.

[53] A. C. Yao, *How to generate and exchange secrets*, in Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1986, pp. 162–167.

[54] X. Yu and M. Yung, *Space lower-bounds for pseudorandom-generators*, in Proceedings of the 9th Annual Structure in Complexity Theory Conference, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 186–197.

SIAM J. COMPUT.
Vol. 36, No. 4, pp. 889–974

ROBUST PCPS OF PROXIMITY, SHORTER PCPS, AND APPLICATIONS TO CODING*

ELI BEN-SASSON†, ODED GOLDREICH‡, PRAHLADH HARSHA§, MADHU SUDAN¶, AND SALIL VADHAN‖

Abstract. We continue the study of the trade-off between the length of probabilistically checkable proofs (PCPs) and their query complexity, establishing the following main results (which refer to proofs of satisfiability of circuits of size n):

1. We present PCPs of length $\exp(o(\log\log n)^2)\cdot n$ that can be verified by making $o(\log\log n)$ Boolean queries.
2. For every $\varepsilon > 0$, we present PCPs of length $\exp(\log^{\varepsilon} n)\cdot n$ that can be verified by making a constant number of Boolean queries.

In both cases, false assertions are rejected with constant probability (which may be set to be arbitrarily close to 1). The multiplicative overhead on the length of the proof, introduced by transforming a proof into a probabilistically checkable one, is just quasi polylogarithmic in the first case (of query complexity $o(\log\log n)$), and is $2^{(\log n)^{\varepsilon}}$, for any $\varepsilon > 0$, in the second case (of constant query complexity). Our techniques include the introduction of a new variant of PCPs that we call "robust PCPs of proximity." These new PCPs facilitate proof composition, which is a central ingredient in the construction of PCP systems. (A related notion and its composition properties were discovered independently by Dinur and Reingold.) Our main technical contribution is a construction of a "length-efficient" robust PCP of proximity. While the new construction uses many of the standard techniques used in PCP constructions, it does differ from previous constructions in fundamental ways, and in particular does not use the "parallelization" step of Arora et al. [*J. ACM,* 45 (1998), pp. 501–555]. The alternative approach may be of independent interest. We also obtain analogous quantitative results for locally testable codes. In addition, we introduce a relaxed notion of locally decodable codes and present such codes mapping k information bits to codewords of length $k^{1+\varepsilon}$ for any $\varepsilon > 0$.

Key words. probabilistically checkable proofs, PCP, locally testable codes, locally decodable codes, property testing

AMS subject classifications. 68Q17, 68P30, 94B60

DOI. 10.1137/S0097539705446810

*Received by the editors December 27, 2004; accepted for publication (in revised form) January 24, 2006; published electronically December 15, 2006. Preliminary versions of this paper have appeared in *Proceedings of the 36th ACM Symposium on Theory of Computing* [BGH+04a] and *Electronic Colloquium on Computational Complexity* [BGH+04b]. Part of the work was done while the first, second, fourth, and fifth authors were fellows at the Radcliffe Institute for Advanced Study of Harvard University.

http://www.siam.org/journals/sicomp/36-4/44681.html

†Department of Computer Science, Technion – Israel Institute of Technology, Haifa, Israel (eli@cs.technion.ac.il).

‡Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel (oded.goldreich@weizmann.ac.il). The research of this author was partially supported by the Israel Science Foundation (grant 460/05).

§Toyota Technological Institute, Chicago, IL 60637 (prahladh@tti-c.org). This work was done while the author was at the Massachusetts Institute of Technology. The work of this author was supported in part by NSF Award CCR-0312575.

¶Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 (madhu@mit.edu). The work of this author was supported in part by NSF Award CCR-0312575.

‖Division of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138 (salil@eecs.harvard.edu). This author's work was supported in part by NSF grant CCR-0133096, ONR grant N00014-04-1-0478, and by a Sloan Research Fellowship.

1. Introduction. Probabilistically checkable proofs (PCPs) [FGL+96, AS98, ALM+98] (aka holographic proofs [BFLS91]) are NP witnesses that allow efficient probabilistic verification based on probing few bits of the NP witness. The celebrated PCP theorem [AS98, ALM+98] asserts that probing a constant number of bits suffices, and it turned out that three bits suffice for rejecting false assertions with probability almost 1/2 (cf. [Hås01, GLST98]).

Optimizing the query complexity of PCPs has attracted a lot of attention, motivated in part by the significance of query complexity for nonapproximability results (see, for example, [BGLR93, BGS95, Hås01, GLST98, ST00]). However, these works guarantee only that the new NP witness (i.e., the PCP) is of a length that is upper-bounded by a polynomial in the length of the original NP witness.[1] Optimizing the length of the new NP witness was the focus of [BFLS91, PS94, HS00, GS02, BSVW03], and in this work we continue the latter research direction.

In our view, the significance of PCPs extends far beyond their applicability to deriving nonapproximability results. The mere fact that NP witnesses can be transformed into a format that supports superfast probabilistic verification is remarkable. From this perspective, the question of how much redundancy is introduced by such a transformation is fundamental. Furthermore, PCPs have been used not only to derive nonapproximability results but also for obtaining positive results (e.g., computationally sound (CS) proofs [Kil92, Mic00] and their applications [Bar01, CGH04]), and the length of the PCP affects the complexity of those applications.

In any case, the length of PCPs is also relevant to nonapproximability results; specifically, it affects their *tightness with respect to the running time* (as noted in [Sze99]). For example, suppose (exact) satisfiability (SAT) has complexity $2^{\Omega(n)}$. The original PCP theorem [AS98, ALM+98] implies only that approximating maximum satisfiability (MaxSAT) requires time $2^{n^{\alpha}}$ for some (small) $\alpha > 0$. The work of [PS94] makes α arbitrarily close to 1, whereas the results of [GS02, BSVW03] further improve the lower-bound to $2^{n^{1-o(1)}}$. Our results reduce the $o(1)$ term.[2]

1.1. PCPs with better length versus query trade-off. How short can a PCP be? The answer may depend on the number of bits we are willing to read in order to reject false assertions (say) with probability at least 1/2. It is implicit in the work of [PS94] that, for proofs of satisfiability of circuits of size n, if we are willing to read $n^{0.01}$ bits, then the length of the new NP witness may be $\widetilde{O}(n)$. That is, stretching the NP witness by only a polylogarithmic amount, allows us to dramatically reduce the number of bits read (from n to $n^{0.01}$). More precisely, see the following theorem.[3]

THEOREM 1.1 (implicit in [PS94]). *Satisfiability of circuits of size n can be probabilistically verified by probing an NP witness of length* $\mathrm{poly}(\log n) \cdot n$ *in* $n^{o(1)}$ *bit locations. In fact, for any integer value of a parameter* $m \leq \log n$*, there is a PCP having randomness complexity* $(1-m^{-1}) \cdot \log_2 n + O(\log \log n) + O(m \log m)$ *and query complexity* $\mathrm{poly}(\log n) \cdot n^{1/m}$.

Recall that the proof length of a PCP is at most $2^r \cdot q$, where r is the randomness complexity and q is the query complexity of the PCP. Thus, the first part of the above theorem follows by setting $m = \log \log n / \log \log \log n$ in the second part.

Our results show that the query complexity can be reduced dramatically if we are

[1] We stress that in all the above works, as well as in the current work, the new NP witness can be computed in polynomial time from the original NP witness.

[2] A caveat: It is currently not known whether these improved lower-bounds can be achieved simultaneously with optimal approximation ratios, but the hope is that this can eventually be done.

[3] All logarithms in this work are of base 2, but in some places we choose to emphasize this fact by using the notation $\log_2$ rather than log.

willing to increase the length of the proof slightly. First, with a quasi-polylogarithmic stretch, the query complexity can be made double-logarithmic as follows.

THEOREM 1.2. *Satisfiability of circuits of size n can be probabilistically verified by probing an NP witness of length $\exp(o(\log\log n)^2)\cdot n$ in $o(\log\log n)$ bit locations. In fact, it has a PCP having randomness complexity $\log_2 n + O\big((\log\log n)^2/\log\log\log n\big)$ and query complexity $O(\log\log n/\log\log\log n)$.*

We mention that the only prior work claiming query complexity below $\exp(\sqrt{\log n})$ (cf. [GS02, BSVW03]) required stretching the NP witness by at least an $\exp(\sqrt{\log n})$ factor. With approximately such a stretch factor, these works actually achieved constant query complexity (cf. [GS02, BSVW03]). Thus, Theorem 1.2 represents a vast improvement in the query complexity of PCPs that use very short proofs (i.e., in the range between $\exp(o(\log\log n)^2)\cdot n$ and $\exp(\sqrt{\log n})\cdot n$). On the other hand, considering NP witnesses that allow probabilistic verification by a *constant* number of queries, we reduce the best known stretch factor from $\exp(\log^{0.5+\varepsilon} n)$ (established in [GS02, BSVW03]) to $\exp(\log^{\varepsilon} n)$, for any $\varepsilon > 0$ as follows.

THEOREM 1.3. *For every constant $\varepsilon > 0$, satisfiability of circuits of size n can be probabilistically verified by probing an NP witness of length $\exp(\log^{\varepsilon} n)\cdot n$ in a constant number of bit locations. In fact, it has a PCP having randomness complexity $\log_2 n + \log^{\varepsilon} n$ and query complexity $O(1/\varepsilon)$.*

It may indeed be the case that the trade-off (between length blow-up factors and query complexity) offered by Theorems 1.1–1.3 merely reflects our (incomplete) state of knowledge. In particular, we wonder whether circuit satisfiability can be probabilistically verified by a PCP having proof length $n\cdot\mathrm{poly}(\log n)$ and constant query complexity.

1.2. New notions and main techniques. A natural approach to reducing the query complexity of the PCP provided by Theorem 1.1 is via the "proof composition" paradigm of [AS98]. However, that PCP (as constructed in [PS94]) does not seem amenable to composition when the parameter m is nonconstant.[4] The reason is that standard proof composition requires the "outer" proof system to make a constant number of multivalue oracle queries (or be converted to such), whereas this specific PCP does not have this property and we cannot afford the standard parallelization involved in a suitable conversion. Thus, we begin by giving a new PCP construction whose parameters match those in Theorem 1.1, but is suitable for composition. As we will see, we cannot afford the standard proof composition techniques, and thus also introduce a new, more efficient composition paradigm.

The initial PCP. Our new proof of Theorem 1.1 modifies the constructions of [PS94] and [HS00]. The latter construction was already improved in [GS02, BSVW03] to reduce the length of PCPs to $n\cdot 2^{\widetilde{O}(\sqrt{\log n})}$. Our results go further by re-examining the "low-degree test" (query-efficient tests that verify if a given function is close to being a low-degree polynomial) and observing that the small-biased sample sets of [BSVW03] give an even more significant savings on the randomness complexity of low-degree tests than noticed in that work. However, exploiting this advantage takes a significant effort in modifying known PCP modules and redefining the ingredients in "proof composition."

For starters, PCP constructions tend to use many (i.e., a superconstant number of) functions and need to test if each is a low-degree polynomial. In prior results,

[4] Also for constant m, we get stronger quantitative results by using our new PCP construction as a starting point.

this was performed efficiently by combining the many different functions on, say, m variables, into a single new one on $m+1$ variables, where the extra variable provides an index into the many different old functions. Testing if the new function is of low-degree implicitly tests all the old functions. Such tricks, which involve introducing a few extra variables, turn out to be too expensive in our context. Furthermore, for similar reasons, we cannot use other "parallelization" techniques [FRS94, LS97, ALM+98, GS00, Raz98], which were instrumental to the proof composition technique of [AS98]. In turn, this forces us to introduce a new variant of the proof composition method, which is much more flexible than the one of [AS98]. Going back to the PCP derived in Theorem 1.1, we adapt it for our new composition method by introducing a "bundling" technique that offers a randomness-efficient alternative to parallelization.

Our new proof composition method refers to two new notions: the notion of a *PCP of proximity* and the notion of a *robust PCP*. Our method is related to the method discovered independently by Dinur and Reingold [DR04]. (There are significant differences between the two methods, as explained in section 1.3, where we also discuss our method in relation to Szegedy's work [Sze99].)

PCPs of proximity. Recall that a standard PCP is given an explicit input (which is supposedly in some NP language) as well as access to an oracle that is supposed to encode a "probabilistically verifiable" NP witness. The PCP verifier uses oracle queries (which are counted) in order to probabilistically verify whether the input, which is explicitly given to it, is in the language. In contrast, a *PCP of proximity* is given access to two oracles, one representing an input (supposedly in the NP language) and the other being a redundant encoding of an NP witness (as in a PCP). Indeed, the verifier may query both the input oracle and the proof oracle, but *its queries to the input oracle are also counted in its query complexity.* As usual we focus on verifiers having very low query complexity, certainly smaller than the length of the input. Needless to say, such a constrained verifier cannot distinguish inputs in the language from inputs out of the language, but it is not required to do so. A verifier for a PCP of proximity is required only to accept inputs that are in the language and reject inputs that are *far* from the language (i.e., far in Hamming distance from any input in the language). Indeed, PCPs of proximity are related to holographic proofs [BFLS91] and to "PCP spot-checkers" [EKR04]; see a further discussion in section 1.3.

Robust PCPs. To discuss robust PCPs, let us review the soundness guarantee of standard (nonadaptive) PCPs. The corresponding verifier can be thought of as determining, based on its coin tosses, a sequence of oracle positions and a predicate such that evaluating this predicate on the indicated oracle bits always accepts if the input is in the language and rejects with high probability otherwise. That is, in the latter case, we require that the assignment of oracle bits to the predicate satisfies the predicate. In a *robust PCP* we strengthen the latter requirement. We require that the said assignment (of oracle bits) not only fail to satisfy the predicate but also be *far* from any assignment that satisfies the predicate.

Proof composition. The key observation is that our proof composition works very smoothly when we compose an outer robust PCP with an inner PCP of proximity. We need neither worry about how many queries the outer robust PCP makes nor care about what coding the inner PCP of proximity uses in its proof oracle (much less apply the same encoding to the outer answers). All that we should make sure of is that the lengths of the objects match and that the distance parameter in the robustness condition (of the outer verifier) is at least as big as the distance parameter

in the proximity condition (of the inner verifier).

Indeed, Theorems 1.2 and 1.3 are proved by first extending Theorem 1.1 to provide a robust PCP of proximity of similar complexities, and then applying the new "proof composition" method. We stress that our contribution is in providing a proof of Theorem 1.1 that lends itself to a modification that satisfies the robustness property, and in establishing the latter property. In particular, the aforementioned "bundling" is applied in order to establish the robustness property. Some care is required when deriving Theorem 1.2 using a nonconstant number of proof compositions. In particular, Theorem 1.2 (resp., Theorem 1.3) is derived in a way that guarantees that the query complexity is linear rather than exponential in the number of proof compositions, where the latter is $o(\log \log n)$ (resp., $1/\varepsilon$). This, in turn, requires obtaining strong bounds on the robustness property of the ("robust") extension of Theorem 1.1.

We stress that the flexibility in composing robust PCPs of proximity plays an important role in our ability to derive quantitatively stronger results regarding PCPs. We believe that robust PCPs of proximity may play a similar role in other quantitative studies of PCPs. We note that the standard PCP theorem of [AS98, ALM+98] can be easily derived using a much weaker and simpler variant of our basic robust PCP of proximity, and the said construction seems easier than the basic PCPs used in the proof composition of [AS98, ALM+98].

In addition to their role in our proof composition method, PCPs of proximity also provide a good starting point for deriving improved locally testable codes (see the discussion in section 1.4). The relation of PCPs of proximity to "property testing" is further discussed in section 1.3.

1.3. Related work. As mentioned above, the notion of a PCP of proximity is related to notions that have appeared in the literature.

Relation to holographic proofs. First, the notion of a PCP of proximity generalizes the notion of holographic proofs set forth by Babai et al. [BFLS91]. In both cases, the verifier is given oracle access to the input, and we count its probes to the input in its query complexity. The key issue is that holographic proofs refer to inputs that are presented in an error-correcting format (e.g., one aims to verify that a graph that is *represented by an error-correcting encoding of its adjacency matrix* (or incidence list) is 3-colorable). In contrast, a PCP of proximity refers to inputs that are presented in any format but makes assertions only about their proximity to acceptable inputs (e.g., one is interested in whether a graph, represented by its adjacency matrix (or incidence list), *is* 3*-colorable or is far from being* 3*-colorable*).

Relation to property testing. PCPs of proximity are implicit in the low-degree testers that utilize auxiliary oracles (e.g., an oracle that provides the polynomial representing the value of the function restricted to a queried line); cf. [AS98, ALM+98]. PCPs of proximity are a natural special case of the PCP spot-checkers defined by Ergün, Kumar, and Rubinfeld [EKR04]. On the other hand, PCPs of proximity extend property testing [RS96, GGR98]. Loosely speaking, a *property tester* is given oracle access to an input and is required to distinguish the case in which the input has the property from the case in which it is far (say, in Hamming distance) from any input having the property. Typically, the interest is in testers that query their input on few bit locations (or at the very least on a sublinear number of such locations). In a PCP of proximity such a tester (now called a verifier) is also given oracle access to an alleged proof. Thus, the relation of PCPs of proximity to property testing is analogous to the relation of NP to BPP (or RP). Put differently, while property testing provides a notion of approximation for decision procedures, a PCP of proximity

provides a notion of approximation for (probabilistic) proof-verification procedures. In both cases, approximation means that inputs in the language should be accepted (when accompanied by suitable proofs), while inputs that are far from the language should be rejected (no matter what false proof is provided).

We comment that PCPs of proximity are provably stronger than property testers; that is, there are (natural) separations between property testers and PCPs of proximity (which may be viewed as the "approximation" versions of BPP and NP). For further discussions, refer to section 2.2.

Relation to assignment testers and another proof composition method. As stated above, our proof composition method is related to the method discovered independently by Dinur and Reingold [DR04]. Both methods use the same notion of PCPs of proximity (which are called assignment testers in [DR04]). A key difference between the two methods is that, while our method refers to the new notion of robustness (i.e., to the robustness of the outer verifier), the method of Dinur and Reingold refers to the number of (non-Boolean) queries (made by the outer verifier). Indeed, the method of Dinur and Reingold uses a (new) parallelization procedure (which reduces the number of queries by a constant factor), whereas we avoid parallelization altogether (but rather use a related "bundling" of queries into a nonconstant number of "bundles" such that robustness is satisfied at the bundle level).[5] We stress that we cannot afford the cost of any known parallelization procedure because, at the very least, these procedures increase the length of the proof by a factor related to the answer length, which is far too large in the context of Theorem 1.1 (which in turn serves as the starting point for all the other results in this work). We comment that the parallelization procedure of [DR04] is combinatorial (albeit inapplicable in our context), whereas our bundling relies on the algebraic structure of our proof system.

Relation to Szegedy's work [Sze99]. Some of the ideas presented in the current work are implicit in Szegedy's work [Sze99]. In particular, notions of robustness and proximity are implicit in [Sze99], in which a robust PCP of proximity (attributed to [PS94]) is composed with itself in a way that is similar to our composition theorem. We note that Szegedy does not seek to obtain PCPs with improved parameters, but rather to suggest a framework for deriving nicer proofs of existing results such as those in [PS94]. Actually, he focuses on proving the main result of [PS94] (i.e., a PCP of nearly linear length and constant number of queries) using as a building block a robust PCP of proximity that has length $\widetilde{O}(n)$ and makes $\widetilde{O}(\sqrt{n})$ queries (plus the constant query PCP of [ALM+98]).

We note that the aforementioned robust PCP of proximity is not presented in [Sze99], but is rather attributed to [PS94]. Indeed, observe that Theorem 1.1 above (implicit in [PS94]) achieves $\widetilde{O}(n)$ length and $\widetilde{O}(\sqrt{n})$ queries when the parameter $m = 2$. Thus, Szegedy's assertion is that this PCP can be strengthened to be a robust PCP of proximity, similarly to our main construct (specifically, Theorem 3.1, specialized to $m = 2$). However, our main construct achieves stronger parameters than those claimed in [Sze99], especially with respect to robust soundness. Indeed,

[5] The main part of the bundling technique takes place at the level of analysis, without modifying the proof system at all. Specifically, we show that the answers read by the verifier can be partitioned into a nonconstant number of (a priori fixed) bundles so that on any no instance, with high probability a constant fraction of the bundles read should be modified to make the verifier accept. We stress that the fact that certain sets of queries (namely, those in each bundle) are always made together is a feature that our particular proof system happens to have (or rather it was somewhat massaged to have). Once robust soundness is established at the bundle level, we need only modify the proof system so that the bundles become queries and the answers are placed in (any) good error-correcting format, which implies robustness at the bit level.

the parameters claimed in [Sze99] allow only for the robust PCP of proximity to be composed with itself a constant number of times.[6] As mentioned above, a significant amount of our effort is aimed at ensuring that our robust PCP of proximity has sufficiently strong parameters to be composed a nonconstant number of times and, moreover, to ensure that the query complexity grows only linearly rather than exponentially with the number of compositions. (See section 3.2 for further explanation.)

1.4. Applications to coding problems. The flexibility of PCPs of proximity makes them relatively easy to use towards obtaining results regarding locally testable and decodable error-correcting codes. In particular, using a suitable PCP of proximity, we obtain an improvement in the rate of locally testable codes (improving over the results of [GS02, BSVW03]). Loosely speaking, a codeword test (for a code C) is a randomized oracle machine that is given oracle access to a string. The tester may query the oracle at a constant number of bit locations and is required to (always) accept every codeword and reject with (relatively) high probability every string that is "far" from the code. The locally testable codes of [GS02, BSVW03] used codewords of length $\exp(\log^{0.5+\varepsilon} k) \cdot k$ in order to encode k bits of information for any constant $\varepsilon > 0$. Here we reduce the length of the codewords to $\exp(\log^{\varepsilon} k) \cdot k$ as follows.

THEOREM 1.4 (loosely stated; see section 4.1 for details). *For every constant $\varepsilon > 0$, there exists locally testable codes that use codewords of length $\exp(\log^{\varepsilon} k) \cdot k$ in order to encode k bits of information.*

We also introduce a relaxed notion of locally decodable codes and show how to construct such codes using any PCP of proximity (and ours in particular). Loosely speaking, a code is said to be *locally decodable* if, whenever relatively few location are corrupted, the decoder is able to recover each information bit, with high probability, based on a constant number of queries to the (corrupted) codeword. This notion was formally defined by Katz and Trevisan [KT00] and the best known locally decodable code has codewords of a length that is subexponential in the number of information bits. We relax the definition of locally decodable codes by requiring that, whenever few locations are corrupted, the decoder be able to recover most of the individual information bits (based on few queries), and for the rest of the locations, the decoder may output a fail symbol (but not the wrong value). That is, the decoder must still avoid errors (with high probability), but is allowed to say "`don't know`" on a few bit locations. We show that this relaxed notion of local decodability can be supported by codes that have codewords of a length that is almost linear in the number of information bits as follows.

THEOREM 1.5 (loosely stated; see section 4.2 for details). *For every constant $\varepsilon > 0$, there exists relaxed locally decodable codes that use codewords of length $k^{1+\varepsilon}$ in order to encode k bits of information.*

1.5. Subsequent work. Since the presentation of our results, there has been considerable progress in the construction of short PCPs.

Ben-Sasson and Sudan [BS05] constructed "shorter" PCPs for NP at the cost of a slightly larger query complexity. More precisely, they construct PCPs of length $n \cdot \mathrm{poly}(\log n)$ (to prove satisfiability of circuits of size n) that can be verified by querying at most $\mathrm{poly}(\log n)$ bits of the proof. They achieve this improvement in

[6] In the language of section 2, his soundness and robustness parameters are unspecified functions of the proximity parameter. In retrospect, it seems that the ideas of [PS94] may lead to a robust PCP of proximity with robustness that is at best linearly related to the proximity parameter; this would make the query complexity increase exponentially with the number of compositions (as discussed in section 3.2).

length by constructing PCPs of proximity for a specific problem, verifying membership in a *Reed–Solomon* code (i.e., verifying if a given function is close to the evaluation of some univariate polynomial of specified degree). Their construction also yields locally testable codes with similar parameters.

More recently, Dinur introduced a novel gap amplification technique to yield a fully combinatorial proof of the PCP theorem [Din06]. In addition, by applying the gap-amplification technique to the PCP constructed by Ben-Sasson and Sudan [BS05], she obtained PCPs for NP of length $n \cdot \text{poly}(\log n)$ and verifiable with a constant number of probes into the proof. Thus while Ben-Sasson and Sudan [BS05] reduce the proof length while increasing the number of queries, Dinur shows how to reduce the query size back to a constant, thereby improving both our results and those of [BS05].

The PCP verifiers in the constructions of both our paper and [BS05] require time at least polynomial in the length of the proof, though the verifiers probe at most $\text{poly}(\log n)$ locations in the proof. In a subsequent paper [BGH$^+$05], we demonstrate that both these constructions can in fact be accompanied with *superefficient* verifiers, i.e., verifiers that run in time at most polylogarithmic in the length of the proof.

Finally, the question of whether there exist nearly linear-sized PCPs for NP that achieve strong query-soundness trade-offs (e.g., achieving the parameters of Håstad [Hås01]) remains open. The recent work of Moshkovitz and Raz [MR06] may be useful towards answering this.

1.6. Organization. Theorems 1.2 and 1.3, which are the work's main results, are proved by constructing and using a robust PCP of proximity that achieves a very good trade-off between randomness and query complexity. Thus, this robust PCP of proximity is the main building block that underlies our work. Unfortunately, the construction of a very efficient robust PCP of proximity is quite involved and is thus deferred to the second part of this work (which starts with an overview). In the first part of this work we show how the aforementioned robust PCP of proximity can be used to derive all the results mentioned in the introduction (and, in particular, Theorems 1.2 and 1.3). Thus, the overall structure of this work is as follows.

Part I: Using the main building block (sections 2–4). We start by providing a definitional treatment of PCPs of proximity and robust PCPs. The basic definitions as well as some observations and useful transformations are presented in section 2. Most important, we analyze the natural composition of an outer robust PCP with an inner PCP of proximity.

In section 3, we state the properties of our main building block (i.e., a highly efficient robust PCP of proximity), and show how to derive Theorems 1.2 and 1.3, by composing this robust PCP of proximity with itself multiple times. Specifically, $o(\log \log n)$ compositions are used to derive Theorem 1.2, and $1/\varepsilon$ compositions are used to derive Theorem 1.3. The coding applications stated in Theorems 1.4 and 1.5 are presented in section 4.

Part II: Constructing the main building block (sections 5–8). We start this part by providing an overview of the construction. This overview (i.e., section 5) can be read before reading Part I, provided that the reader is comfortable with the notion of a robust PCP of proximity.

The construction itself is presented in section 6–8. We start by presenting a (highly efficient) ordinary PCP (establishing Theorem 1.1), which lends itself to the subsequent modifications. In section 7, we augment this PCP with a test of proximity, deriving an analogous PCP of proximity. In section 8 we present a robust version of

the PCP of proximity derived in the previous sections.

Part III: Appendices. The construction presented in section 3 also uses a PCP of proximity of polynomial randomness complexity and constant query complexity. Such a PCP of proximity can be derived by a simple augment of the Hadamard-based PCP of [ALM+98], which we present in Appendix A.

In Appendix B, we recall results regarding randomness-efficient low-degree tests and a related sampling lemma, which are used in Part II.

1.7. Relation to previous versions of this work. The current version includes a discussion of Szegedy's work [Sze99], of which we were unaware when writing the first version [BGH+04b]. The relation of his work to ours is now discussed in section 1.3.

Section 4 has been extensively revised, adding formal definitions and providing more precise descriptions of the main constructions and proofs. In addition, we identified a weaker form of the definition of a relaxed locally decodable code, proved that it essentially implies the original form, and restructured our presentation accordingly (see section 4.2).

The parameters of Theorem 1.2 in this version are stronger (to a limited extent) than that of earlier versions of this paper [BGH+04b, BGH+04a]. More specifically, we show that satisfiability of circuits of size n can be verified by probing $o(\log\log n)$ bit locations in an NP witness of length $\exp(o(\log\log n)^2)\cdot n$ as opposed to an NP witness of length $\exp(\widetilde{O}(\log\log n)^2)\cdot n$, as was claimed in earlier versions. We achieve this strengthening by improving the robustness parameter of the ALMSS-type robust PCP of proximity (Theorem 7.2) constructed in Part II of this paper, taking advantage of the greater slackness allowed in the randomness complexity of this PCP. (ALMSS-type robust PCP of proximity is one of the PCPs of proximity constructed in Part II. It is so called as it has parameters similar to the PCP constructed in [ALM+98].)

Part I. All but the main construct.

2. PCPs and variants: Definitions, observations, and transformations.

Notation. Except when otherwise noted, all *circuits* in this paper have fan-in 2 and fan-out 2, and we allow arbitrary unary and binary Boolean operations as internal gates. The *size* of a circuit is the number of gates. We will refer to the following languages associated with circuits: the P-complete language Circuit Value, defined as CktVal $= \{(C,w) : C(w) = 1\}$; the NP-complete Circuit Satisfiability, defined as CktSAT $= \{C : \exists w C(w) = 1\}$; and the NP-complete Nondeterministic CircuitValue, defined as NCktVal $= \{(C,w) : \exists z C(w,z) = 1\}$. (In the latter, we assume that the partition of the variables of C into w-variables and z-variables is explicit in the encoding of C.)

We will extensively refer to the *relative* distance between strings/sequences over some alphabet Σ: For $u, v \in \Sigma^\ell$, we denote by $\Delta(u,v)$ the fraction of locations on which u and v differ (i.e., $\Delta(u,v) \triangleq |\{i : u_i \neq v_i\}|/\ell$, where $u = u_1 \cdots u_\ell \in \Sigma^\ell$ and $v = v_1 \cdots v_\ell \in \Sigma^\ell$). We say that u is *δ-close* to v (resp., *δ-far* from v) if $\Delta(u,v) \leq \delta$ (resp., $\Delta(u,v) > \delta$). The relative distance of a string from a set of strings is defined in the natural manner; that is, $\Delta(u,S) \triangleq \min_{v\in S}\{\Delta(u,v)\}$. Occasionally, we will refer to the absolute Hamming distance, which we will denote by $\overline{\Delta}(u,v) \triangleq |\{i : u_i \neq v_i\}|$. We will also use the t-repetition x^t of a string x to denote the string formed by concatenating t copies of x (i.e., $x^t = x \ldots t \text{ times} \ldots x$).

Organization of this section. After recalling the standard definition of PCP (in section 2.1), we present the definitions of PCPs of proximity and robust PCPs (in

section 2.2 and 2.3, respectively). We then discuss (in section 2.4) the composition of a robust PCP with a PCP of proximity. Various observations and transformations regarding the new notions are presented in section 2.5.

2.1. Standard PCPs. We begin by recalling the formalism of a PCP verifier. Throughout this work, we restrict our attention to *nonadaptive* verifiers, not only for simplicity but also because one of our variants (namely robust PCPs) only makes sense for nonadaptive verifiers.

DEFINITION 2.1 (PCP verifiers).

- *A* verifier *is a* probabilistic polynomial-time *algorithm V that, on an input x of length n, tosses $r = r(n)$ random coins R and generates a* sequence *of $q = q(n)$ queries $I = (i_1, \ldots, i_q)$ and a circuit $D : \{0,1\}^q \to \{0,1\}$ of size at most $d(n)$.*
  We think of V as representing a probabilistic oracle machine that queries its oracle π for the positions in I, receives the q answer bits $\pi|_I \triangleq (\pi_{i_1}, \ldots, \pi_{i_q})$, and accepts iff $D(\pi|_I) = 1$.
- *We write $(I, D) \overset{R}{\leftarrow} V(x)$ to denote the queries and circuit generated by V on input x and random coin tosses and write $(I, D) = V(x; R)$ if we wish to specify the coin tosses R.*
- *We call r the* randomness complexity*, q the* query complexity*, and d the* decision complexity *of V.*

For simplicity in these definitions, we treat the parameters r, q, and d above (and other parameters below) as functions of only the input length n. However, at times we may also allow them to depend on other parameters, which should be understood as being given to the verifier together with the input. We now present the standard notion of PCPs, restricted to perfect completeness for simplicity.

DEFINITION 2.2 (standard PCPs). *For a function $s : \mathbb{Z}^+ \to [0,1]$, a verifier V is a* probabilistically checkable proof system *for a language L with* soundness error *s if the following two conditions hold for every string x:*

Completeness: If $x \in L$, then there exists π such that $V(x)$ accepts oracle π with probability 1*. Formally,*

$$\exists \pi \qquad \Pr_{(I,D) \overset{R}{\leftarrow} V(x)} [D(\pi|_I) = 1] = 1.$$

Soundness: If $x \notin L$, then for every oracle π, the verifier $V(x)$ accepts π with probability strictly less than s. Formally,

$$\forall \pi \qquad \Pr_{(I,D) \overset{R}{\leftarrow} V(x)} [D(\pi|_I) = 1] < s(|x|).$$

If s is not specified, then it is assumed to be a constant in $(0, 1)$*.*

Our main goal in this work is to construct *short* PCPs that use *very few queries*. Recalling that the length of a (nonadaptive) PCP is upper-bounded by $2^{r(n)} \cdot q(n)$, we focus on optimizing the (trade-off between) randomness and query complexities.

We will focus on constructing PCPs for the NP-complete problem CIRCUIT SATISFIABILITY, defined as CKTSAT $= \{C : \exists w\ C(w) = 1\}$. Recall that every language in **NTIME**$(t(n))$ reduces to CKTSAT in time $O(t(n) \log t(n))$ (cf. [HS66, PF79, Coo88]), and so a nearly linear-sized PCP for CKTSAT implies PCPs for **NTIME**$(t(n))$ of length nearly linear in $t(n)$ for every polynomial $t(n)$.

2.2. PCPs of proximity. We now present a relaxation of PCPs that verify only that the input is *close* to an element of the language. The advantage of this relaxation is that it allows for the possibility that the verifier may read only a small number of bits from the input. Actually, for greater generality, we will divide the input into two parts (x, y), giving the verifier x explicitly and y as an oracle, and we count only the verifier's queries to the latter. Thus we consider languages consisting of pairs of strings, which we refer to as *pair languages*. One pair language to keep in mind is the CircuitValue problem CktVal $= \{(C, w) : C(w) = 1\}$. For a pair language L, we define $L(x) = \{y : (x, y) \in L\}$. For example, CktVal$(C)$ is the set of satisfying assignments to C. It will be useful below to treat the two oracles to which the verifier has access as a single oracle; thus for oracles π^0 and π^1, we define the concatenated oracle $\pi = \pi^0 \circ \pi^1$ as $\pi_{b,i} = \pi_i^b$.

Definition 2.3 (PCPs of proximity (PCPPs)). *For functions $s, \delta : \mathbb{Z}^+ \to [0, 1]$, a verifier V is a* probabilistically checkable proof of proximity *(PCPP) system for a pair language L with* proximity parameter δ *and* soundness error s *if the following two conditions hold for every pair of strings (x, y):*

Completeness: If $(x, y) \in L$, then there exists π such that $V(x)$ accepts oracle $y \circ \pi$ with probability 1*. Formally,*

$$\exists \pi \qquad \Pr_{(I,D) \overset{R}{\leftarrow} V(x)} [D((y \circ \pi)|_I) = 1] = 1.$$

Soundness: If y is $\delta(|x|)$-far from $L(x)$, then for every π, the verifier $V(x)$ accepts oracle $y \circ \pi$ with probability strictly less than $s(|x|)$. Formally,

$$\forall \pi \qquad \Pr_{(I,D) \overset{R}{\leftarrow} V(x)} [D((y \circ \pi)|_I) = 1] < s(|x|).$$

If s and δ are not specified, then both are assumed to be constants in $(0, 1)$.

Note that the parameters (soundness, randomness, etc.) of a PCPP are measured as a function of the length of x, the explicit portion of the input.

In comparing PCPPs and PCPs, one should note two differences that have conflicting effects. On one hand, the soundness criterion of PCPPs is a relaxation of the soundness of PCPs. Whereas a PCP is required to reject (with high probability) every input that is not in the language, a PCPP is only required to reject input pairs (x, y) in which the second element (i.e., y) is far from being suitable for the first element (i.e., y is far from $L(x)$). That is, in a PCPP, nothing is required in the case that y is close to $L(x)$ and yet $y \notin L(x)$. On the other hand, the query complexity of a PCPP is measured more stringently, as it accounts also for the queries to the input-part y (on top of the standard queries to the proof π). This should be contrasted with a standard PCP that has free access to all its input and is charged only for access to an auxiliary proof. To summarize, PCPPs are required to do less (i.e., their performance requirements are more relaxed), but they are charged for more things (i.e., their complexity is evaluated more stringently). Although it may not be a priori clear, the stringent complexity requirement prevails. That is, PCPPs tend to be more difficult to construct than PCPs of the same parameters. For example, while CktVal has a trivial PCP (since it is in P), a PCPP for it implies a PCP for CktSAT as follows.

Proposition 2.4. *If* CktVal *has a PCPP, then* CktSAT *has a PCP with identical parameters (randomness, query complexity, decision complexity, and soundness).*

An analogous statement holds for any pair language L and the corresponding projection on first element $L_1 \triangleq \{x : \exists y \text{ s.t. } (x,y) \in L\}$; that is, if L has a PCPP, then L_1 has a PCP with identical parameters.

Proof. A PCP π for "$C \in$ CktSAT" can be taken to be $w \circ \pi'$, where w is a satisfying assignment to C and π' is a PCPP for $(C,w) \in$ CktVal. This proof π can be verified using the PCPP verifier. The key observation is that if $C \notin$ CktSAT, then there exists no w that is 1-close to CktVal(C), because the latter set is empty. ∎

Note that we obtain only a standard PCP for CktSAT rather than a PCP of proximity. Indeed, CktSAT is not a pair language, so it does not even fit syntactically into the definition of a PCPP. However, we can give a PCPP for the closely related (and also NP-complete) pair language Nondeterministic Circuit Value. Recall that it is the language NCktVal $= \{(C,w) : \exists z C(w,z) = 1\}$ (where the variables of C are explicitly partitioned into w-variables and z-variables).

Proposition 2.5. *If* CktVal *has a PCPP with proximity parameter $\delta(n)$, soundness $s(n)$, randomness $r(n)$, query complexity $q(n)$, and decision complexity $d(n)$, then* Nondeterministic Circuit Value *has a PCPP with proximity parameter $2\delta(4n)$, soundness $s(4n)$, randomness $r(4n)$, query complexity $q(4n)$, and decision complexity $d(4n)$.*

Proof. Given a circuit $C(\cdot,\cdot)$ of size n whose variables are partitioned into one group of size k and another of size ℓ, we transform it into a new circuit $C'(\cdot,\cdot)$ of size $n' = 4n$ in which the first group has size $k' \geq \ell$ and the second group has size ℓ. Specifically, we set $t = \lceil \ell/k \rceil$ and $k' = t \cdot k$ and define $C'(x',y)$ to be a circuit that checks whether $x' = x^t$ for some x such that $C(x,y) = 1$. It can be verified that this can be done in size $n + 3tk \leq 4n$ (over the full binary basis). In addition, if w is δ-far from being extendable to a satisfying assignment of C, then w^t is δ-far from being extendable to a satisfying assignment of C'.

Now, the NCktVal-verifier, on explicit input C and input oracle $w \in \{0,1\}^k$, will construct C' as above and expect a proof oracle of the form $z \circ \pi$, where $z \in \{0,1\}^m$ and π is a PCPP for $(C', w^t \circ z) \in$ CktVal satisfies as constructed above. That is, the NCktVal-verifier will simulate the CktVal-verifier on explicit input C', input oracle $w^t \circ z$ (which can easily be simulated given oracle access to w and z), and proof oracle π. Completeness can be verified by inspection. For soundness, suppose that w is 2δ-far from being extendable to a satisfying assignment of C. Then w^t is 2δ-far from being extendable to a satisfying assignment of C', which implies that, for any z, $w^t \circ z$ is δ-far from satisfying C'. Thus, by the soundness of the CktVal-verifier, the acceptance probability is at most $s(n') = s(4n)$ for any proof oracle π. ∎

Relation to property testing. Actually, the requirements of a PCPP for a pair language L refer only to its performance on the ("gap") promise problem $\Pi = (\Pi_Y, \Pi_N)$, where $\Pi_Y = L$ and $\Pi_N = \{(x,y) : y \text{ is } \delta\text{-far from } L(x)\}$. That is, this PCPP is required only to (always) accept inputs in Π_Y and reject (with high probability) inputs in Π_N (whereas nothing is required with respect to inputs not in $\Pi_Y \cup \Pi_N$). Such a gap problem corresponds to the notion of approximation in *property testing* [RS96, GGR98].[7] Indeed, property testers are equivalent to PCPP verifiers that have no access to an auxiliary proof π. Thus, the relation between property testing and PCPPs is analogous to the relation between BPP and NP (or MA). For example, the problem of testing bipartiteness can be cast by the pair language

[7] This notion of approximation (of decision problems) should not be confused with the approximation of (search) optimization problems, which is also closely related to PCPs [FGL+96, ALM+98].

$L = \{(n, G) : \text{the } n\text{-vertex graph } G \text{ is bipartite}\}$, where the first (i.e., explicit) input is used only to specify the length of the second (i.e., nonexplicit) input G, to which the tester has oracle access (measured in its query complexity). We comment that the formulation of pair languages allows us to capture more general property testing problems, where more information about the property (to be tested) itself is specified as part of the input (e.g., by a circuit, as in CktVal).

In both property testers and PCPPs, the interest is in testers/verifiers that query their input (and proof oracle) in only a small (preferably constant, and certainly sublinear) number of bit locations. It turns out that PCPPs are provably stronger than property testers; that is, there are (natural) separations between property testers and PCPPs. (Some of the following examples were pointed out in [EKR04].) In the adjacency matrix model (cf. [GGR98]), bipartiteness has a PCPP in which the verifier makes only $O(1/\delta)$ queries and rejects any graph that is δ-far from being bipartite with probability at least 2/3. (The proof oracle consists of an assignment of vertices to the two parts, and the verifier queries the assignment of the end-points of $O(1/\delta)$ random edges. This construction also generalizes to k-colorability, and in fact to any generalized graph partition property (cf. [GGR98]) with an efficient one-sided tester.) In contrast, Bogdanov and Trevisan [BT04] showed that any tester for bipartiteness that rejects graphs that are δ-far from being bipartite must make $\Omega(\delta^{-3/2})$ queries. More drastic separations are known in the incidence-lists (bounded-degree) model (of [GR02]): testing bipartiteness (resp., 3-colorability) of n-vertex graphs has query complexity $\Omega(\sqrt{n})$ [GR02] (resp., $\Omega(n)$ [BOT02]), but again a PCPP will use only $O(1/\delta)$ queries.

Another example comes from the domain of codes. For any good code (or "even" any code of linear distance), there exists a PCPP with constant queries for checking whether a given word is a codeword.[8] This stands in contrast to the linear lower-bound on the query complexity of codeword testing for some (good) linear codes, proved by Ben-Sasson, Harsha, and Raskhodnikova [BHR05].

Needless to say, there may be interesting cases in which PCPPs do not outperform property testers.

Queries versus proximity. Intuitively, the query complexity of a PCPP should depend on the proximity parameter δ. Proposition 2.8 (in section 2.5) confirms this intuition.

The relation of PCPP to other works. As discussed in the introduction (see section 1.3), notions related to (and equivalent to) PCPPs have appeared previously in the literature [BFLS91, EKR04]. In particular, holographic proofs are a special case of PCPPs (which refer to pair languages $L = \{(n, \mathcal{C}(x)) : x \in L' \cap \{0,1\}^n\}$, where $\mathcal{C}$ is an error-correcting code and $L' \in \text{NP}$), whereas PCPPs are a special case of PCP spot-checkers (when viewing decision problems as a special case of search problems). In addition, PCPPs play an important role also in the work of Dinur and Reingold [DR04]; again, see section 1.3. Recall that both our use and their use of PCPPs is for facilitating proof composition (of PCP-type constructs). Finally, existing PCP constructions (such as [ALM+98]) can be modified to yield PCPPs.

2.3. Robust soundness. In this section, we present a *strengthening* of the standard PCP soundness condition. Instead of asking that the bits which the verifier reads from the oracle be merely rejected with high probability, we ask that the bits which

[8] Indeed, this is a special case of our extension of the result of Babai et al. [BFLS91], discussed in section 1.3. On the other hand, this result is simpler than the locally testable code mentioned in section 1.4 because here the PCPP is not part of the codeword.

the verifier reads be *far* from being accepted with high probability. The main motivation for this notion is that, in conjunction with PCPPs, it allows for a very simple composition without the usual costs of "parallelization."

DEFINITION 2.6 (robust soundness). *For functions $s, \rho : \mathbb{Z}^+ \to [0,1]$, a PCP verifier V for a language L has* robust-soundness error *s with* robustness parameter *ρ if the following holds for every $x \notin L$: For every oracle π, the bits read by the verifier V are ρ-close to being accepted with probability strictly less than s. Formally,*

$$\forall \pi \quad \Pr_{(I,D) \overset{R}{\leftarrow} V(x)} [\exists a \text{ s.t. } D(a) = 1 \text{ and } \Delta(a, \pi|_I) \leq \rho] < s(|x|).$$

If s and ρ are not specified, then they are assumed to be constants in $(0,1)$. PCPPs with robust soundness are defined analogously, with the $\pi|_I$ being replaced by $(y \circ \pi)|_I$.

Note that for PCPs with query complexity q, robust soundness with any robustness parameter $\rho < 1/q$ is equivalent to standard PCP soundness. However, there can be robust PCPs with large query complexity (e.g., $q = n^{\Omega(1)}$) yet constant robustness, and indeed such robust PCPs will be the main building block of our construction.

Various observations regarding robust PCPs are presented in section 2.5. We briefly mention here the relation of robustness to parallelization; specifically, when applied to a robust PCP, the simple query-reduction technique of Fortnow, Rompel, and Sipser [FRS94] performs less poorly than usual (i.e., the resulting soundness is determined by the robustness parameter rather than by the number of queries).

2.4. Composition. As promised, a robust "outer" PCP composes very easily with an "inner" PCPP. Loosely speaking, we can compose such schemes provided that the decision complexity of the outer verifier matches the input length of the inner verifier, and soundness holds provided that the robustness parameter of the outer verifier upper-bounds the proximity parameter of the inner verifier. Note that composition does not refer to the query complexity of the outer verifier, which is always upper-bounded by its decision complexity.

THEOREM 2.7 (composition theorem). *Suppose that for functions $r_{\text{out}}, r_{\text{in}}, d_{\text{out}}, d_{\text{in}}, q_{\text{in}} : \mathbb{N} \to \mathbb{N}$, and $\varepsilon_{\text{out}}, \varepsilon_{\text{in}}, \rho_{\text{out}}, \delta_{\text{in}} : \mathbb{N} \to [0,1]$, the following hold:*

- *Language L has a* robust PCP *verifier V_{out} with randomness complexity r_{out}, decision complexity d_{out}, robust-soundness error $1 - \varepsilon_{\text{out}}$, and robustness parameter ρ_{out}.*
- CKTVAL *has a* PCPP *verifier V_{in} with randomness complexity r_{in}, query complexity q_{in}, decision complexity d_{in}, proximity parameter δ_{in}, and soundness error $1 - \varepsilon_{\text{in}}$.*
- *$\delta_{\text{in}}(d_{\text{out}}(n)) \leq \rho_{\text{out}}(n)$ for every n.*

Then, L has a (standard) *PCP, denoted V_{comp}, with*

- *randomness complexity $r_{\text{out}}(n) + r_{\text{in}}(d_{\text{out}}(n))$,*
- *query complexity $q_{\text{in}}(d_{\text{out}}(n))$,*
- *decision complexity $d_{\text{in}}(d_{\text{out}}(n))$, and*
- *soundness error $1 - \varepsilon_{\text{out}}(n) \cdot \varepsilon_{\text{in}}(d_{\text{out}}(n))$.*

Furthermore, there exists a universal algorithm with black-box access to V_{out} and V_{in} that can perform the actions of V_{comp} (i.e., evaluating $(I, D) \leftarrow V_{\text{comp}}(x; R)$). On inputs of length n, this algorithm runs in time n^c for a universal constant c, with one call to V_{out} on an input of length n and one call to V_{in} on an input of length $d_{\text{out}}(n)$. In addition,

- *if* (instead of being a PCP) *the verifier V_{out} is a PCPP with proximity parameter $\delta_{\text{out}}(n)$, then V_{comp} is a PCPP with proximity parameter $\delta_{\text{out}}(n)$;*

- *if V_{in} has robust soundness with robustness parameter $\rho_{\text{in}}(n)$, then V_{comp} has robust soundness with robustness parameter $\rho_{\text{in}}(d_{\text{out}}(n))$.*

Proof. We will use the inner PCPP to verify that the oracle positions selected by the (robust) outer verifier are close to being accepted by the outer verifier's decision circuit. Thus, the new proof will consist of a proof for the outer verifier as well as proofs for the inner verifier, where each of the latter corresponds to a possible setting of the outer verifier's coin tosses (and is intended to prove that the bits that should have been read by the outer verifier satisfy its decision circuit). We will index the positions of the new (combined) oracle by pairs such that (out, i) denotes the ith position in the part of the oracle that represents the outer verifier's proof oracle, and (R, j) denotes the jth position in the Rth auxiliary block (which represents the Rth possible proof oracle (for the inner verifiers), which in turn is associated with the outer verifier's coins $R \in \{0,1\}^{r_{\text{out}}}$). For notational convenience, we drop the input length n from the notation below; all parameters of V_{out} are with respect to length n and all parameters of V_{in} are with respect to length $d_{\text{out}}(n)$. With these conventions, the following is the description of the composed verifier $V_{\text{comp}}(x)$:

1. Choose $R \overset{\text{R}}{\leftarrow} \{0,1\}^{r_{\text{out}}}$.
2. Run $V_{\text{out}}(x; R)$ to obtain $I_{\text{out}} = (i_1, \ldots, i_{q_{\text{out}}})$ and D_{out}.
3. Run $V_{\text{in}}(D_{\text{out}})$ (on random coin tosses) to obtain $I_{\text{in}} = ((b_1, j_1), \ldots, (b_{q_{\text{in}}}, j_{q_{\text{in}}}))$ and D_{in}.
   (Recall that V_{in}, as a PCPP verifier, expects two oracles, an input oracle and a proof oracle, and thus makes queries of the form (b, j), where $b \in \{0,1\}$ indicates which oracle it wishes to query.)
4. For each $\ell = 1, \ldots, q_{\text{in}}$, determine the queries of the composed verifier as follows:
   (a) If $b_\ell = 0$, set $k_\ell = (\text{out}, i_{j_\ell})$; that is, V_{in}'s queries to its input oracle are directed to the corresponding locations in V_{out}'s proof oracle. Recall that the jth bit in V_{in}'s input oracle is the jth bit in the input to D_{out}, which in turn is the i_jth bit in the proof oracle of V_{out}.
   (b) If $b_\ell = 1$, set $k_\ell = (R, j_\ell)$; that is, V_{in}'s queries to its Rth possible proof oracle are directed to the corresponding locations in the auxiliary proof. Recall that the jth bit in the proof oracle that V_{in} is using to verify the claim referring to the outer verifier's coins R is the jth bit in the Rth block of the auxiliary proof.
5. Output $I_{\text{comp}} = (k_1, \ldots, k_{q_{\text{in}}})$ and D_{in}.

The claims about V_{comp}'s randomness, query, decision, and computational complexities can be verified by inspection. Thus, we proceed to check completeness and soundness.

Suppose that $x \in L$. Then, by completeness of the outer verifier, there exists a proof π_{out} making V_{out} accept with probability 1. In other words, for every $R \in \{0,1\}^{r_{\text{out}}}$, if we set $(I_{\text{out}}, D_{\text{out}}) = V_{\text{out}}(x; R)$, we have $D_{\text{out}}(\pi_{\text{out}}|_{I_{\text{out}}}) = 1$. By completeness of the inner verifier, there exists a proof π_R such that $V_{\text{in}}(D_{\text{out}})$ accepts the oracle $\pi_{\text{out}}|_{I_{\text{out}}} \circ \pi_R$ with probability 1. If we set $\pi(t, \cdot) = \pi_t(\cdot)$ for all $t \in \{\text{out}\} \cup \{0,1\}^{r_{\text{out}}}$, then V_{comp} accepts π with probability 1.

Suppose that $x \notin L$, and let π be any oracle. Define oracles $\pi_t(\cdot) = \pi(t, \cdot)$. By the robust soundness (of V_{out}), with probability greater than ε_{out} over the choices of $R \in \{0,1\}^{r_{\text{out}}}$, if we set $(I_{\text{out}}, D_{\text{out}}) = V_{\text{out}}(x; R)$, then $\pi_{\text{out}}|_{I_{\text{out}}}$ is ρ_{out}-far from satisfying D_{out}. Fixing such an R, by the PCPP soundness of V_{in} (and $\delta_{\text{in}} \leq \rho_{\text{out}}$), it holds that $V_{\text{in}}(D_{\text{out}})$ rejects the oracle $\pi_{\text{out}}|_{I_{\text{out}}} \circ \pi_R$ (or, actually, any proof oracle

augmenting the input oracle $\pi_{\text{out}}|_{I_{\text{out}}}$) with probability greater than ε_{in}. Therefore, $V_{\text{comp}}(x)$ rejects oracle π with probability at least $\varepsilon_{\text{out}} \cdot \varepsilon_{\text{in}}$.

The additional items follow by similar arguments. If V_{out} is a PCPP verifier, then the input is of the form (x, y), where y is given via oracle access. In this case, throughout the proof above we should replace oracle π_{out} with oracle $y \circ \pi_{\text{out}}$, and for soundness we should consider the case that y is δ_{out}-far from $L(x)$. If V_{in} has robust soundness, then at the end of the soundness analysis, we note that not only is $\pi_{\text{out}}|_{I_{\text{out}}} \circ \pi_R$ rejected with probability greater than ε_{in} but it is also ρ_{in}-far from being accepted by V_{in} (and hence also by V_{comp}). □

The above theorem can serve as a substitute for the original composition theorem in the derivation of the original PCP theorem [ALM+98]. Specifically, one simply needs to modify the (precomposition) verifiers of [ALM+98] to both test proximity and have robust soundness. As we shall see in the next section, robust soundness can be obtained automatically from "parallelized PCPs" (as already constructed in [ALM+98]). In addition, the PCPs [ALM+98] can easily be made PCPPs by augmenting them with appropriate "proximity tests." Thus, all the technical work in Part II is not forced by the new notion of robust PCPPs, but rather is aimed at constructing PCPPs (and thus PCPs) which have *nearly linear length*.

2.5. Various observations and transformations. Most of this subsection refers to robust PCPs, but we start with an observation regarding PCPPs.

Queries versus proximity. Intuitively, the query complexity of a PCPP should depend on the proximity parameter δ. The following proposition confirms this intuition.

PROPOSITION 2.8 (queries versus proximity). *Suppose pair language L has a PCPP with proximity parameter δ, soundness error $1 - \varepsilon$, and query complexity q. Suppose further that there exists $(x, y) \in L$ such that $|x| = n$ and $|y| = m$, such that if we let $z \in \{0,1\}^m$ be a random string of relative Hamming distance $\delta' \triangleq \delta'(x)$ from y, we have*

$$\Pr_z[z \text{ is } \delta\text{-far from } L(x)] \geq \gamma \triangleq \gamma(x).$$

Then

$$q > \frac{\varepsilon \cdot \gamma}{\delta'}$$

In particular, if $L =$ CKTVAL, then $q \geq \varepsilon/(\delta + O(1/n))$.

The first part of Proposition 2.8 does not specify the relation of δ' to δ (although, surely, $\delta' > \delta$ must hold for any $\gamma > 0$ because $\Delta(z, L(x)) \leq \Delta(z, y) = \delta'$). The second part relies on the fact that, for CIRCUIT VALUE, one may set δ' as low as $\delta + O(1/n)$.

Proof. By completeness, there exists an oracle π such that the PCPP verifier $V(x)$ accepts oracle $y \circ \pi$ with probability 1. Consider $z = y \oplus \eta$, where $\eta \in \{0,1\}^m$ is a uniformly distributed string with relative Hamming weight δ'. If we invoke $V(x)$ with oracle $z \circ \pi$, then the probability (over the choice of η) that any of the positions read by V has been changed is at most $q \cdot \delta'$. Thus, $V(x)$ rejects oracle $(y \oplus \eta) \circ \pi$ with probability at most $q \cdot \delta'$.

On the other hand, by assumption z is δ-far from $L(x)$ with probability at least γ, in which case $V(x)$ should reject oracle $z \circ \pi$ with probability greater than ε by the PCPP soundness. Thus, $V(x)$ should reject with probability greater than $\gamma \cdot \varepsilon$ (over the choice of z and the coin tosses of V), and we conclude that $q \cdot \delta' > \gamma \cdot \varepsilon$, as desired.

For the application to CIRCUIT VALUE, let $C : \{0,1\}^m \to \{0,1\}$ be a circuit of size n that accepts only the all-zeroes string 0^m, for $m = \Omega(n)$. Then we have $(C, 0^m) \in \text{CKTVAL}$, but for every $\delta' > \delta$ and every string z of relative Hamming weight δ', we see that (C, z) is δ-far from satisfying C. Setting $\gamma = 1$ and δ' such that $\delta' m$ is the least integer greater than δm completes the proof. □

Expected robustness. Occasionally, we will be interested in a variant of robust soundness, which refers to distance on average rather than with high probability.

DEFINITION 2.9 (expected robustness). *For a function $\rho : \mathbb{Z}^+ \to [0,1]$, a PCP has* expected robustness ρ *if for every $x \notin L$, we have*

$$\forall \pi, \mathrm{E}_{(I,D) \overset{R}{\leftarrow} V(x)}[\Delta(\pi|_I, D^{-1}(1))] > \rho(|x|).$$

Expected robustness for PCPPs is defined analogously.

We now present several generic transformations regarding robustness and soundness. Although we state them only for PCPs, all of these results also hold for PCPPs, with no change in the proximity parameter. The following proposition relates robust soundness to expected robustness.

PROPOSITION 2.10 (robust soundness versus expected robustness). *If a PCP has robust-soundness error $1 - \varepsilon$ with robustness ρ, then it has expected robustness $\varepsilon \cdot \rho$. On the other hand, if a PCP has expected robustness ρ, then for every $\varepsilon \leq \rho$, it has robust-soundness error $1 - \varepsilon$ with robustness parameter $\rho - \varepsilon$.*

Expected robustness can easily be amplified to standard robustness *with low robust-soundness error*, using any averaging (i.e., oblivious) sampler (cf. [Gol97]). Combined with Proposition 2.10, we get a (soundness) error reduction for robust PCPs. For example, using the expander-neighborhood sampler of [GW97], we have the following.

LEMMA 2.11 (error reduction via expander neighborhoods). *If a language L has a PCP with expected robustness ρ, randomness complexity r, query complexity q, and decision complexity d, then for every two functions $s, \gamma : \mathbb{Z}^+ \to [0,1]$ then L has a PCP with*

- *robust-soundness error s with robustness parameter $\rho - \gamma$,*
- *randomness complexity $r + O(\log(1/s) + \log(1/\gamma))$,*
- *query complexity $O(1/(s\gamma^2)) \cdot q$, and*
- *decision complexity $O(1/(s\gamma^2)) \cdot d$.*

An alternative error-reduction procedure that will also be used is given by pairwise independent samples as follows.

LEMMA 2.12 (error reduction via pairwise independence). *If a language L has a PCP with expected robustness ρ, randomness complexity r, query complexity q, and decision complexity d such that $\rho \cdot 2^r \geq 2$, then L has a PCP with*

- *robust-soundness error $1/2$ with robustness parameter $\rho/2$,*
- *randomness complexity $2r$,*
- *query complexity $2q/\rho$, and*
- *decision complexity $2d/\rho$.*

Non-Boolean PCPs. The next few transformations involve non-Boolean PCPs, that is, PCPs in which the oracle returns symbols over some larger alphabet $\Sigma = \{0,1\}^a$ rather than bits; we refer to $a = a(n)$ as the *answer length* of the PCP. (Often, non-Boolean PCPs are discussed in the language of multiprover interactive proofs, but it is simpler for us to work with the PCP formulation.)

Robust soundness of a non-Boolean PCP is defined in the natural way, using Hamming distance over the alphabet Σ. (In the case of a robust non-Boolean PCPP,

we still treat the input oracle as binary.)

The first transformation provides a way of converting non-Boolean PCPs to Boolean PCPs in a way that preserves robust soundness.

LEMMA 2.13 (alphabet reduction). *If a language L has a non-Boolean PCP with answer length a, query complexity q, randomness complexity r, decision complexity d, and robust-soundness error s with robustness parameter ρ, then L has a Boolean PCP with query complexity $O(a \cdot q)$, randomness complexity r, decision complexity $d + O(a \cdot q)$, and robust soundness error s with robustness parameter $\Omega(\rho)$. If, instead of robust-soundness, the non-Boolean PCP has expected robustness ρ, then the Boolean PCP has expected robustness $\Omega(\rho)$.*

The proof uses a good error-correcting code (i.e., constant relative distance and rate). Furthermore, to obtain decision complexity $d + O(a \cdot q)$ we should use a code having linear-sized circuits for encoding (cf. [Spi96]). Using more classical codes would only give decision complexity $d + \widetilde{O}(a \cdot q)$, which is actually sufficient for our purposes.

Proof. This transformation is analogous to converting non-Boolean error-correcting codes to Boolean ones via "code concatenation." Let V be the given non-Boolean PCP verifier, with answer length a. Let $\mathrm{ECC} : \{0,1\}^a \to \{0,1\}^b$ for $b = O(a)$ a binary error-correcting code of constant relative minimum distance, which can be computed by an explicit circuit of size $O(a)$. We will augment the original oracle π having a-bit entries with an additional oracle τ having b-bit entries, where τ_i is supposed to be $\mathrm{ECC}(\pi_i)$. (We note that including the original oracle simplifies the argument as well as frees us from assuming a noiseless decoding algorithm.)

Our new verifier V'(x), on oracle access to $\pi \circ \tau$, will simulate $V(x)$, and for each query i made by V, will query the a bits in π_i and the b bits in τ_i, for a total of $q \cdot (a + b)$ binary queries. That is, if V queries positions $I = (i_1, \ldots, i_q)$, V' will query positions $I' = ((0, i_1), \ldots, (0, i_q), (1, i_1), \ldots, (1, i_q))$. If V outputs a decision circuit $D : (\{0,1\}^a)^q \to \{0,1\}$, V' will output the circuit $D' : (\{0,1\}^a)^q \times (\{0,1\}^b)^q \to \{0,1\}$ defined by

$$D'(x_1, \ldots, x_q, y_1, \ldots, y_q) = D(x_1, \ldots, x_q) \wedge C(x_1, \ldots, x_q, y_1, \ldots, y_q),$$

where

$$C(x_1, \ldots, x_q, y_1, \ldots, y_q) = \bigwedge_{i=1}^{q} (y_i = \mathrm{ECC}(x_i)).$$

Since ECC can be evaluated by a circuit of size $O(a)$, we see that $|D'| = |D| + O(a \cdot q)$, as desired.

For completeness of V', we note that any accepting oracle π for V can be augmented to be an accepting oracle for V' by setting $\tau_i = \mathrm{ECC}(\pi_i)$ for all i. For soundness of V', suppose $x \notin L$ and let (π, τ) be any pair of oracles. Define a "decoded" oracle $\hat{\pi}$ by setting $\hat{\pi}_i$ to be the string $x \in \{0,1\}^a$ which minimizes the distance between $\mathrm{ECC}(x)$ and τ_i. We will relate the robustness of V on oracle $\hat{\pi}$ to the robustness of V' on oracles π and τ. Specifically, let $\beta > 0$ be a constant such that the (absolute) minimum distance of ECC is greater than $2\beta \cdot (a + b)$. Then we will show that for every sequence R of coin tosses and for every $\alpha > 0$, if the bits read by $V'(x; R)$ from $\pi \circ \tau$ are $\alpha\beta$-close to being accepted, then the bits read by V from $\hat{\pi}$ are α-close to being accepted. Thus, both robustness parameters (standard and expected) decrease by at most a factor of β.

Consider any sequence R of coin tosses, let $(I, D) = V(x; R)$, and write $I = (i_1, \ldots, i_q)$. Suppose that $(\pi_{i_1}, \ldots, \pi_{i_q}, \tau_{i_1}, \ldots, \tau_{i_q})$ is $\alpha\beta$-close to some $(\pi'_{i_1}, \ldots, \pi'_{i_q},$

$\tau'_{i_1}, \ldots, \tau'_{i_q}$) that satisfies $D' = D \wedge C$. Then, for at least a $1 - \alpha$ fraction of $j \in [q]$, the pair (π_{i_j}, τ_{i_j}) is β-close to $(\pi'_{i_j}, \tau'_{i_j}) = (\pi'_{i_j}, \mathrm{ECC}(\pi'_{i_j}))$. For such j, the choice of β implies that $\mathrm{ECC}(\pi'_{i_j})$ is the closest codeword to τ_{i_j} and hence $\hat{\pi}_{i_j} = \pi'_{i_j}$. Since the π''s satisfy D, we conclude that the $\hat{\pi}$'s are α-close to satisfying D, as desired. ∎

The usual "parallelization" paradigm of PCPs [LS97, ALM+98] converts a Boolean PCP with many queries into a non-Boolean PCP with a constant number of queries, where this is typically the first step in PCP composition. As mentioned in the introduction, we cannot afford parallelization, and robust soundness will be our substitute. Nevertheless, there is a close (but not close enough for us) connection between parallelized PCPs and PCPs with robust soundness as follows.

PROPOSITION 2.14 (parallelization versus robustness).

1. *If a language L has a non-Boolean PCP with answer length a, query complexity q, randomness complexity r, decision complexity d, and soundness error s, then L has a* (Boolean) *PCP with query complexity $O(a \cdot q)$, randomness complexity r, decision complexity $d + O(a \cdot q)$, and robust-soundness error s with robustness parameter $\rho = \Omega(1/q)$.*

2. *If a language L has a* (Boolean) *PCP with query complexity q, randomness complexity r, decision complexity d, and expected robustness ρ, then L has a 2-query non-Boolean PCP with answer length q, randomness complexity $r + \log q$, decision complexity $d + O(1)$, and soundness error $1 - \rho$.*

Thus, for constant soundness and constant robustness parameters, q-query robust (Boolean) PCPs are essentially equivalent to constant query non-Boolean PCPs with answer length $\Theta(q)$. However, note that in passing from robust soundness to a 2-query non-Boolean PCP, the randomness complexity increases by $\log q$. It is precisely this cost that we cannot afford, and hence we work with robust soundness in the rest of the paper.

Proof. For item 1, note that any non-Boolean PCP with query complexity q and soundness error s has robust-soundness error s for any robustness parameter $\rho < 1/q$. Thus, the claim follows from Lemma 2.13.

Turning to item 2, let V be a robust PCP verifier for L with the stated parameters. We use the usual query-reduction technique for PCPs [FRS94] and observe that when applied to a robust PCP, the detection probability (i.e., one minus the soundness error) does not deteriorate by a factor of q as usual. Instead, the detection probability of the resulting 2-query (non-Boolean) PCP equals the expected robustness of V.[9] Specifically, the 2-query non-Boolean PCP verifier V' is defined as follows:

- V' expects two oracles: one Boolean oracle π corresponding to the oracle for V, and a second oracle τ with answer length q, indexed by random strings of V.
- On input x, the verifier V' selects a random string R for V and $j \stackrel{R}{\leftarrow} [q]$ and computes $(I, D) = V(x; R)$, where $I = (i_1, \ldots, i_q)$. It sets $I' = (R, i_j)$ (which means the queries for the values τ_R and π_{i_j}) and $D'(a, b) = [(D(a) = 1) \wedge (a_j = b)]$; that is, it accepts if and only if $[D(\tau_R) = 1] \wedge [(\tau_R)_j = \pi_{i_j}]$.

[9] It may be more instructive (but more cumbersome) to discuss what is happening in terms of ordinary robustness. Suppose that V has robust-soundness error $s = 1 - d$ with respect to robustness ρ. The standard analysis ignores the robustness and asserts that the 2-query (non-Boolean) PCP has soundness error $s' = 1 - d'$, where $d' = d/q$. This crude analysis implicitly assumes the trivial bound (i.e., $1/q$) of the robustness parameter. A more refined analysis takes advantage of the actual bound of the robustness parameter and asserts that the 2-query (non-Boolean) PCP has soundness error $s' = 1 - \rho \cdot d$.

It can be verified that the probability that V' rejects a false assertion is precisely the expected robustness of V. In particular, suppose that $V'(x)$ accepts the oracle pair (π, τ) with probability p. We may assume, without loss of generality, that $D(\tau_R) = 1$ for any R, where $(\cdot, D) = V(x; R)$. Then, it follows that the expected (relative) distance of $\pi|_I$ from τ_R, where $(I, D) = V(x; R)$ for a random R, equals $1-p$ (because $1-p = \Pr_{R,j}[(\tau_R)_j \neq \pi_{i_j}]$, which in turn equals $\mathrm{E}_R[\Delta(\tau_R, \pi|_I)]$). This means that on the average, π is $(1-p)$-close to assignments that satisfy the corresponding decision circuits. Thus, if $x \notin L$, then $1 - p > \rho$, and $p < 1 - \rho$ follows. ∎

Robustness versus proximity. Finally, for PCPPs, we prove that the robustness parameter is upper-bounded by the proximity parameter.

PROPOSITION 2.15 (robustness versus proximity). *Suppose a pair language L has a PCPP with proximity parameter δ and expected robustness ρ. Suppose further that there exists $(x, y) \in L$ such that $|x| = n$ and $|y| = m$, such that if we let $z \in \{0,1\}^m$ be a random string at relative Hamming distance $\delta' \triangleq \delta'(x)$ from y, we have*

$$\Pr_z[z \text{ is } \delta\text{-far to } L(x)] \geq \gamma \triangleq \gamma(x).$$

Then

$$\rho \leq \delta'/\gamma.$$

In particular, if $L = \textsc{CktVal}$, then $\rho \leq \delta + O(1/n)$.

Proof. The proof is similar to that of Proposition 2.8. By completeness, there exists an oracle π such that the PCPP verifier $V(x)$ accepts oracle $y \circ \pi$ with probability 1. If we run $V(x)$ with oracle $z \circ \pi$ instead, then bits read by V have expected distance at most δ' from being accepted, where the expectation is taken over the choices of z (even when fixing the coins of V).

On the other hand, z is δ-far from $L(x)$ with probability at least γ, and for any such fixed z the bits read by V from $z \circ \pi$ should have expected distance greater than ρ from being accepted (over the coin tosses of V). Thus, the expected distance of $z \circ \pi$ from being accepted is greater than $\gamma \cdot \rho$, where here the expectation is taken over the choice of z and the coin tosses of V. We conclude that $\delta' > \gamma \cdot \rho$, as desired.

Recall that in the proof of Proposition 2.8, we have demonstrated the existence of a pair (C, w) such that for any string z at distance $\delta' = \delta + O(1/n)$ from w it holds that w is δ-far from satisfying C. Setting $\gamma = 1$, the second part follows. ∎

3. Very short PCPs with very few queries. In this section we prove the main results of this work; that is, we establish Theorems 1.2 and 1.3. Our starting point is the following robust PCPP, which is constructed in Part II, sections 5–8.

THEOREM 3.1 (main construct). *There exists a universal constant c such for all $n, m \in \mathbb{Z}^+$, $0 < \delta, \gamma < 1/2$ satisfying $n^{1/m} \geq m^{cm}/(\gamma\delta)^3$ and $\delta \leq \gamma/c$, $\textsc{CktVal}$ has a robust PCPP (for circuits of size n) with the following parameters:*

- *randomness $\left(1 - \frac{1}{m}\right)\log n + O(m \log m) + O(\log\log n) + O(\log(1/\delta))$;*
- *decision complexity $n^{1/m} \cdot \mathrm{poly}(\log n, 1/\delta)$; which also upper-bounds the query complexity;*[10]
- *perfect completeness; and*
- *for proximity parameter δ, a verifier having robust-soundness error γ with robustness parameter $(1-\gamma)\delta$.*

[10] In fact, we will upper-bound the query complexity by $q = n^{1/m} \cdot \mathrm{poly}(\log n, 1/\delta)$ and show that the verifier's decision can be implemented by a circuit of size $\widetilde{O}(q)$, which can also be bounded by $n^{1/m} \cdot \mathrm{poly}(\log n, 1/\delta)$ with a slightly larger unspecified polynomial.

We comment that the condition $\delta < \gamma/c$ merely means that we present robust PCPPs only for the more difficult cases (when δ is small), and our robustness parameter does not improve for larger values of δ. We call the reader's attention to the typically small values of the query and randomness complexities, which yield a proof length that is upper-bounded by $\text{poly}(m^m \log n) \cdot n$ (for δ and γ as small as $1/\text{poly}(m^m, \log n)$), as well as to the small values of the soundness error and the small deterioration of robustness with respect to proximity.

Note that the main construct (of Theorem 3.1) works only when n, the size of the input circuit, is not too small (more precisely, when $n^{1/m} \geq m^{cm}/\delta^3$). While constructing our short PCPs (via proof composition), we need robust PCPPs that work for even smaller values of n. For this purpose, we also construct the following robust PCPP (of Theorem 3.2) that has parameters similar to a PCP constructed in [ALM+98]. In comparison to the main construct (of Theorem 3.1), this PCPP is not as efficient in randomness (i.e., it has randomness complexity $O(\log n)$ rather than $(1 - o(1)) \log_2 n$). However, since we plan to use the latter (robust) PCPP only towards the final stages of composition, we can afford to pay this cost in randomness. Theorem 3.2 will be proved in the second part of this work by modifying the proof of Theorem 3.1. An alternate construction of this robust PCPP can be obtained by adding a suitable proximity test to the "parallelized PCPs" of [ALM+98].

THEOREM 3.2 (ALMSS-type robust PCPP). *For all $n \in \mathbb{Z}^+$ and $\delta \in (0, 1)$,* CKTVAL *has a robust PCPP (for circuits of size n) with the following parameters:*

- *randomness $O(\log n)$;*
- *decision complexity* $\text{poly} \log n$*, which also upper-bounds the query complexity;*
- *perfect completeness; and*
- *for proximity parameter δ, a verifier having robust-soundness error $1 - \Omega(\delta)$ with robustness parameter $\Omega(1)$.*

Theorems 3.1 and 3.2 differ also in their robustness parameters. Theorem 3.2 provides a better bound on the robustness parameter (i.e., $\Omega(1)$ rather than $(1 - \gamma)\delta$ provided by Theorem 3.1), while guaranteeing only a much weaker robust-soundness error (i.e., $1 - \Omega(\delta)$ rather than γ), where $\gamma > \delta > 0$ is typically small. It is instructive to compare the expected robustness provided by the two results: The expected robustness in Theorem 3.1 is at least $(1-\gamma)^2\delta$, while that in Theorem 3.2 is $\Omega(\delta) \cdot \Omega(1) = \Omega(\delta)$. Thus, for $\gamma \ll 1$, the expected robustness in Theorem 3.1 can be very close to the proximity parameter δ (which is close to optimal; see Proposition 2.15), whereas in Theorem 3.2 the expected robustness is always a constant factor smaller than the proximity parameter. Hence, the robust PCPP of Theorem 3.1 is suitable for a large number of proof composition operations, whereas the one in Theorem 3.2 is useful when the query complexity of the outer verifier is already very small (and Theorem 3.1 can no longer be applied). Indeed, this is exactly how these two theorems are used in the construction of our short PCPs. Using Theorems 3.1 and 3.2, we derive a general trade-off between the length of PCPs and their query complexity as follows.

THEOREM 3.3 (randomness versus query complexity trade-off for PCPPs). *For every parameter $n, t \in \mathbb{N}$ such that $3 \leq t \leq \frac{2 \log \log n}{\log \log \log n}$ there exists a PCPP for* CKTVAL *(for circuits of size n) with the following parameters:*

- *randomness complexity* $\log_2 n + A_t(n)$*, where*

$$A_t(n) \triangleq O(t + (\log n)^{1/t}) \log \log n + O((\log n)^{2/t}); \tag{3.1}$$

- *query complexity $O(1)$;*
- *perfect completeness; and*

- *soundness error* $1-\Omega(1/t)$ *with respect to proximity parameter* $\Theta(1/t)$.

Alternatively, we can have query complexity $O(t)$ *and soundness error* $1/2$ *maintaining all other parameters.*

For $t \in [3, \ldots, \frac{0.99 \log\log n}{\log\log\log n}]$, we have $(\log n)^{1/t} > (\log\log n)^{1/0.99}$, and thus $A_t(n) = O((\log n)^{2/t})$. On the other hand, for $t \geq \frac{1.01 \log\log n}{\log\log\log n}$, we have $(\log n)^{1/t} \leq (\log\log n)^{1/1.01}$, and thus $A_t(n) = O\big(\frac{(\log\log n)^2}{\log\log\log n}\big) = o(\log\log n)^2$.

Theorem 3.3 actually asserts a PCPP (for CktVal), but a PCP for CktSAT and a PCPP for Nondeterministic Circuit Value (of the same complexity) follow; see Propositions 2.4 and 2.5. Theorems 1.2 and 1.3 follow by suitable settings of the parameter t. Further details as well as a corollary appear in section 3.2.

3.1. Proof of Theorem 3.3. Theorem 3.3 is proved by using the robust PCPP described in Theorem 3.1. Specifically, this robust PCPP is composed with itself several times (using the composition theorem from section 2). Each such composition drastically reduces the query complexity of the resulting PCP, while only increasing very moderately its randomness complexity. The deterioration of the soundness error and the robustness is also very moderate. After composing the robust PCPP with itself $O(t(n))$ times, we compose the resulting robust PCP with the ALMSS-type robust PCPP thrice to reduce the query complexity to $\operatorname{poly}\log\log\log n$. Finally, we compose this resultant robust PCPP with a PCPP parameter roughly $\Omega(1/t)$ that has query complexity $O(1)$ and exponential length. The latter PCPP can be obtained by a suitable modification of the Hadamard-based PCP of [ALM+98], as shown in Appendix A. We now turn to the actual proof.

Proof. We construct the PCPP of Theorem 3.3 by composing the robust PCPP described in Theorem 3.1 with itself several times. Each such composition reduces the query complexity from n to approximately $n^{1/m}$. Ideally, we would like to do the following: Set $m = (\log n)^{1/t}$ and compose the robust PCPP of Theorem 3.1 with parameter m with itself $t-1$ times. This would result in a robust PCPP of query complexity roughly $n^{1/m^t} = n^{1/\log n} = O(1)$, giving us the desired result. However, we cannot continue this repeated composition for all the $t-1$ steps, as the requirements of Theorem 3.1 (namely, $n^{1/m} \geq m^{cm}/(\delta\gamma)^3$) are violated in the last two steps of the repeated composition. So we instead do the following: In the first stage, we compose the (new and) highly efficient verifier from Theorem 3.1 with itself $t-3$ times. This yields a verifier with query complexity roughly $n^{1/m^{t-2}} = (n^{1/m^t})^{m^2} = 2^{m^2} = \exp(\log^{2/t} n) \ll n$, while the soundness error is bounded away from 1 and robustness is $\Omega(1/t)$. In the second stage, we compose the resultant robust PCPP a constant number of times with the ALMSS-type robust PCPP described in Theorem 3.2 to reduce the query complexity to $\operatorname{poly}\log\log\log n$ (and keeping the other parameters essentially the same). The ALMSS-type PCPP is (relatively) poor in terms of randomness; however, the input size to the ALMSS-type PCPP is too small to affect the randomness of the resultant PCPP. Finally, we compose with the Hadamard-based verifier of Theorem A.1 to bring the query complexity down to $O(1)$. In all stages, we invoke the composition theorem (Theorem 2.7).

Throughout the proof, n denotes the size of the circuit that is given as the explicit input to the PCPP verifier that we construct. We shall actually construct a sequence of such verifiers. Each verifier in the sequence will be obtained by composing the prior verifier (used as the outer verifier in the composition) with an adequate inner verifier. In the first stage, the inner verifier will be the verifier obtained from Theorem 3.1, whereas in the second and third stages it will be the one obtained from Theorem 3.2

and Theorem A.1, respectively. Either way, the inner verifier will operate on circuits of much smaller size (than n) and will use a proximity parameter that is upper-bounded by the robustness parameter of the corresponding outer verifier.

Stage I. Let $m = (\log n)^{\frac{1}{t}} \geq 2$ and $\gamma = \frac{1}{t}$. For this choice of m and γ, let V_0 be the verifier obtained from Theorem 3.1. We recall the parameters of this verifier: For circuits of size ℓ and any proximity parameter $\delta_0 \in (\gamma/3c, \gamma/c)$, its randomness complexity is $r_0(\ell) \triangleq (1-\frac{1}{m}) \cdot \log_2 \ell + O(\log\log\ell) + O(m \log m) + O(\log t)$, its decision (and query) complexity is $d_0(\ell) \triangleq \ell^{\frac{1}{m}} \cdot \mathrm{poly}(\log \ell, t)$, its soundness error is $s_0 \triangleq \gamma$, and its robustness is $\rho_0 \geq (1-\gamma)\delta_0$.

We compose V_0 with itself $t-3$ times for the same fixed choice of m and γ to obtain a sequence of verifiers of increasingly smaller query complexity.[11] While doing so, we will use the largest possible proximity parameter for the inner verifier (V_0) in each step; that is, in the ith composition, we set the proximity parameter of the inner verifier to equal the robustness of the outer verifier, where the latter is the result of $i-1$ compositions of V_0 with itself. We get a sequence of verifiers $V_1, \ldots, V_{t-2}$ such that $V_1 = V_0$ and the verifier V_i is obtained by composing (the outer verifier) V_{i-1} with (the inner verifier) V_0, where the proximity parameter of the latter is set to equal the robustness of the former. Unlike V_0, which is invoked on different circuit sizes and (slightly) different values of the proximity parameter, all the V_i's ($i \in [t-2]$) refer to circuit size n and proximity parameter $\delta \triangleq \gamma/c < 1/t$.

Let r_i, d_i, δ_i, s_i, and ρ_i denote the randomness complexity, decision (and query) complexity, proximity parameter, soundness error, and the robustness parameter of the verifier V_i. (Recall that V_i will be composed with the inner verifier V_0, where in this composition the input size and proximity parameter of the latter will be set to d_i and ρ_i, respectively, and so we will need to verify that $d_i^{1/m} \geq m^{cm}/(\gamma\rho_i)^3$ and $\rho_i < \gamma/c$ for $i < t-2$).[12] We first claim that the decision complexity, proximity, soundness-error, robustness, and proof length parameters satisfy the following conditions:

(1) Decision complexity: $d_i(n) \leq a(n,m)^2 \cdot n^{1/m^i}$, where $a(\ell, m) \triangleq d_0(\ell)/\ell^{1/m} = \mathrm{poly}(\log \ell, t)$. On the other hand, $d_i(n) \geq n^{1/m^i}$.
(2) Proximity: $\delta_i = \delta$.
(3) Soundness error: $s_i \leq 1 - (1-\gamma)^i$. (In particular, $s_i < i\gamma$.)
(4) Robustness: $\rho_i \geq (1-\gamma)^i \cdot \delta$. On the other hand, $\rho_i \leq \rho_0 < \gamma/c$.
(5) Proof length: $2^{r_i(n)} d_i(n) \leq b(n,m)^i \cdot n$, where $b(\ell, m) \triangleq 2^{r_0(\ell)} \cdot d_0(\ell)/\ell = \mathrm{poly}(m^m, \log \ell, t)$.

We prove this claim by induction on i. For starters, note that the base case (i.e., $i = 1$) follows from the properties of V_0; in particular, $d_1(n) \leq \mathrm{poly}(\log n, t) \cdot n^{1/m}$ and $2^{r_1(n)} d_1(n) \leq \mathrm{poly}(m^m, \log n, t) \cdot n$. Turning to the induction step, assuming that

[11] We assume, for simplicity, that $t \geq 3$. Note that it suffices to establish the claimed result for t greater than any universal constant.

[12] We also need to verify that $n^{1/m} \geq m^{cm}/(\gamma\delta_0)^3$ and $\delta_0 < \gamma/c$ for the initial verifier $V_1 = V_0$, but this is true for our choice of parameters. Furthermore, as ρ_i can only deteriorate with each composition, we have that $\delta_0 = \rho_i \leq \rho_0 \leq \gamma/c$. Thus, the only condition that needs to be verified is $d_i^{1/m} \geq m^{cm}/(\gamma\rho_i)^3$ for $i < t-2$.

these claims holds for V_i, we prove that they hold also for V_{i+1}. For (1), note that

$$\begin{aligned}
d_{i+1}(n) &= d_0(d_i(n)) && \text{[by the composition theorem]}\\
&= a(d_i(n), m) \cdot d_i(n)^{1/m} && \text{[by definition of } a(\cdot,\cdot)\text{]}\\
&\le a(n, m) \cdot d_i(n)^{1/m} && \text{[by monotonicity of } a(\cdot,\cdot) \text{ and } d_i(n) \le n\text{]}\\
&\le a(n, m) \cdot \left(a(n, m)^2 \cdot n^{1/m^i}\right)^{1/m} && \text{[by induction]}\\
&\le a(n, m)^2 \cdot n^{1/m^{i+1}} && \text{[using } m \ge 2\text{]}
\end{aligned}$$

and $d_{i+1}(n) \ge d_i(n)^{1/m} \ge n^{1/m^{i+1}}$ also holds. Clearly $\delta_i = \delta$ and the bound on s_i is straightforward from the composition theorem. Recalling that the proximity parameter for V_0 in this composition is set to ρ_i, we see that the robustness of the composed verifier V_{i+1} is $\rho_{i+1} = (1-\gamma)\rho_i = (1-\gamma)^{i+1}\delta$ as desired. Furthermore, $\rho_i = (1-\gamma)^i\delta \ge (1-\frac{1}{t})^t\delta \ge e^{-1}\delta = \gamma/O(1)$. We now move to the last condition (essentially bounding the randomness). Notice first that $r_{i+1}(n) = r_i(n) + r_0(d_i(n))$, and thus

$$\begin{aligned}
2^{r_{i+1}(n)} \cdot d_{i+1}(n) &= 2^{r_i(n)} \cdot 2^{r_0(d_i(n))} \cdot d_0(d_i(n)) && \text{[by the composition theorem]}\\
&= 2^{r_i(n)} \cdot d_i(n) \cdot b(d_i(n), m) && \text{[by definition of } b(\cdot,\cdot)\text{]}\\
&\le b(n, m)^i \cdot n \cdot b(n, m) && \text{[by induction and monotonicity of } b(\cdot,\cdot)\text{]}\\
&= n \cdot b(n, m)^{i+1}.
\end{aligned}$$

Thus, Part (5) is verified. Recall that we have to verify that $d_i^{1/m} \ge m^{cm}/(\gamma\rho_i)^3$ for $i < t-2$ as promised. We have $d_i^{1/m} \ge (n^{1/m^i})^{1/m} = n^{1/m^{i+1}} \ge n^{1/m^{t-2}}$ (since $i < t-2$). Since $m = (\log n)^{1/t}$, we have $n^{1/m^t} = 2$. Hence, $d_i^{1/m} \ge (n^{1/m^t})^{m^2} = 2^{m^2}$. On the other hand, $m^{cm}/(\gamma\rho_i)^3 \le m^{cm}/(e^{-1}\gamma\delta)^3 = m^{cm} \cdot \mathrm{poly}(t)$ because $\delta = \gamma/c$ and $\gamma = 1/t$. Thus it suffices to verify that $2^{m^2}/m^{cm} \ge \mathrm{poly}(t)$ for $2 \le t \le 2\log\log n/\log\log\log n$, which is straightforward.[13]

Lastly, we consider the running time of V_i, denoted T_i, which ought to be polynomial. A careful use of the composition theorem (Theorem 2.7) indicates that $T_i(n) = \mathrm{poly}(n) + T_{i-1}(n)$ for every $i = 2, \ldots, t-2$, where $T_1(n) = \mathrm{poly}(n)$ (since $V_1 = V_0$). Alternatively, unraveling the inductive composition, we note that V_i consists of invoking V_0 i times, where in the first invocation V_0 is invoked on V_i's input and in later invocations V_0 is invoked on an input obtained from the previous invocation. Furthermore, the output of V_i is obtained by combining the inputs obtained in these $i \le t-2 < n$ invocations.

We now conclude the first stage by showing that the final verifier $V_c = V_{t-2}$ has the desired properties. By Part (5) above (and the fact that $d_{t-2} \ge 1$), we have $r_c(n) = r_{t-2}(n) \le \log n + (t-2)\cdot\log b(n,m) \le \log n + t\log b(n,m)$. By the definition of $b(n,m)$, we have $\log b(n,m) = O(\log\log n) + O(m\log m) + O(\log t) = O(\log\log n + m\log m)$, whereas $m\log m = (\log n)^{\frac{1}{t}} \cdot \frac{1}{t}\log\log n$. Thus $r_c(n) \le \log_2 n + O(t\cdot\log\log n) + t\cdot O(m\log m) = \log_2 n + O(t + (\log n)^{\frac{1}{t}})\cdot\log\log n$. The decision complexity of V_c is $d_c(n) = d_{t-2}(n) \le a(n,m)^2 \cdot n^{1/m^{t-2}} = a(n,m)^2 \cdot 2^{m^2}$, because $n^{1/m^t} = 2$. Using

[13] Note that as t varies from 2 to $2\log\log n/\log\log\log n$, the value of m varies from $\sqrt{\log n}$ to $\sqrt{\log\log n}$. For $t \in [2, 2\log\log n/\log\log\log n]$, the maximum value of $\mathrm{poly}(t)$ is $\mathrm{poly}(\log\log n/\log\log\log n) = \mathrm{poly}(\log\log n)$. On the other hand, for $m \in [\sqrt{\log\log n}, \sqrt{\log n}]$, the minimum value of $2^{m^2}/m^{cm} > 2^{m^2/2}$ is $2^{\sqrt{\log\log n}^2/2} = \sqrt{\log n} \gg \mathrm{poly}(\log\log n)$.

$a(n,m) = \text{poly}(\log n, t)$, it follows that $d_c(n) \leq 2^{m^2} \cdot \text{poly}(\log n)$. The proximity of V_c equals δ, its soundness error is $s_c = s_{t-2} = 1-(1-\gamma)^{t-2} = 1-(1-1/t)^{t-2} < 1-\Omega(1)$, and its robustness is $\rho_c = \rho_{t-2} \geq (1-\gamma)^{t-2}\delta = \delta/e = \Omega(1/t)$.

Stage II. We now compose the verifier V_c with the ALMSS-type verifier V_a described in Theorem 3.2 thrice to obtain the verifiers V', V'', and V''', respectively; that is, V' equals V_c composed with V_a, whereas V'' equals V' composed with V_a, and V''' equals V'' composed with V_a. We apply composition as before, setting the proximity parameter of the inner verifier to equal the robustness parameter of the outer verifier. Recall from Theorem 3.2 that the ALMSS-type verifier V_a has the following parameters: randomness $r_a(\ell,\delta_a) = O(\log \ell)$, decision complexity $d_a(\ell,\delta_a) = \text{poly}\log \ell$, soundness error $s_a(\ell,\delta_a) = 1-\Omega(\delta_a)$, and robustness $\rho_a(\ell,\delta_a) = \Omega(1)$ for input size ℓ and proximity parameter δ_a. Recall that when composing V_c with V_a we set $\delta_a = \rho_c = \Omega(1/t)$, whereas when composing V' (resp., V'') with V_a we set $\delta_a = \rho' = \Omega(1)$ (resp., $\delta_a = \rho'' = \Omega(1)$). Each composition with the inner verifier V_a adds $O(\log d)$ to the randomness, while reducing the query complexity to $\text{poly}\log d$, where d is the decision complexity of the outer verifier. Furthermore, when composing any of these outer verifiers (i.e., either V_c, V', or V'') with V_a, the resulting verifier has robustness parameter $\Omega(1)$ while its robust-soundness error is $1-\Omega((1-s)\rho)$, where ρ and s are the robustness parameter and soundness error of the outer verifier. Hence, the parameters of the verifiers V', V'', and V''' are as follows:

Parameters of V' (recall that $d_c = 2^{m^2} \cdot \text{poly}(\log n)$ and $\rho_c = \Omega(\delta)$):
$r' = r_c + O(\log d_c(n)) = r_c + O(m^2 + \log\log n)$,
$d' = \text{poly}(\log d_c(n)) = \text{poly}(m, \log\log n)$,
$s' = 1-\Omega((1-s_c)\rho_c) = 1-\Omega(\delta)$,
and $\rho' = \Omega(1)$.

Parameters of V'':
$r'' = r' + O(\log d') = r' + O(\log m + \log\log\log n)$,
$d'' = \text{poly}(\log d') = \text{poly}(\log m, \log\log\log n)$,
$s'' = 1-\Omega((1-s')\rho') = 1-\Omega(\delta)$,
and $\rho'' = \Omega(1)$.

Parameters of V''':
$r''' = r'' + O(\log d'') = r'' + O(\log\log m + \log\log\log\log n)$,
$d''' = \text{poly}(\log d'') = \text{poly}(\log\log m, \log\log\log\log n)$,
$s''' = 1-\Omega((1-s'')\rho'') = 1-\Omega(\delta)$,
and $\rho''' = \Omega(1)$.

Recall that the proximity parameter for all three verifiers equals that of V_c (i.e., δ). We have that

$$\begin{aligned} r''' &= r_c + O(m^2 + \log\log n) \\ &= \log_2 n + O(t + (\log n)^{1/t}) \cdot \log\log n + O(m^2), \\ q''' < d''' &= \text{poly}(\log\log\log\log n, \log\log m), \end{aligned}$$

whereas $s''' = 1-\Omega(\delta)$ and $\rho''' = \Omega(1)$. Substituting $m = (\log n)^{1/t}$, we get $r''' = \log_2 n + O(t + (\log n)^{1/t}) \cdot \log\log n + O((\log n)^{2/t})$ and $q''' = \text{poly}(\log\log\log n)$.

Stage III. Finally, we compose V''' with the Hadamard-based inner verifier V_h of Theorem A.1 to obtain our final verifier V_f. The query complexity of V_h, and hence that of V_f, is constant. The randomness complexity of V_f is $r_f(n) \triangleq r'''(n) + r_h(q'''(n)) = r'''(n) + \text{poly}(\log\log\log n)$, because $r_h(\ell) = O(\ell^2)$. Thus, $r_f(n) = \log_2 n + O(t + (\log n)^{1/t}) \cdot \log\log n + O((\log n)^{2/t})$. On proximity parameter δ_h, the soundness error of V_h is $s_h = 1-\Omega(\delta_h)$. Setting $\delta_h = \rho''' = \Omega(1)$, we conclude that

the soundness error of V_f on proximity parameter δ is $1 - \Omega(\delta) = 1 - \Omega(1/t)$, since the soundness error of V''' is $1 - \Omega(\delta)$.

To obtain soundness error $1/2$, we perform $O(t)$ repetitions of V_f, yielding a query complexity of $O(t)$. This can be done without increasing the randomness complexity by using "recycled randomness" (specifically, the neighbors of a uniformly selected vertex in a Ramanujan expander graph; see [Gol97, Apdx. C.4]). □

Comment. We note that the tight bound on the robustness (as a function of the proximity parameter) in our main construct (Theorem 3.1) plays an important role in the proof of Theorem 3.3. The reason is that when we compose two robust PCPPs, the proximity parameter of the second must be upper-bounded by the robustness parameter of the first. Thus, when we compose many robust PCPPs, the robustness parameter deteriorates exponentially in the number of composed systems, where the base of the exponent is determined by the tightness of the robustness (of the second verifier). That is, let $\tau \triangleq \rho/\delta$, where δ and ρ are the proximity and robustness parameters of the system. Then composing this system t times with itself means that at the lowest PCP-instance we need to set the proximity parameter to be τ^{t-1} times the initial proximity. This requires the lowest PCP-instance to make at least $1/\tau^{t-1}$ queries (or be composed with a PCPP that can handle proximity parameter τ^t, which again lower-bounds the number of queries). For a constant $\tau < 1$, we get $\exp(t)$ query complexity, whereas for $\tau = 1 - \gamma = (1 - (1/t))$ we get query complexity that is linear in $1/((1-\gamma)^t \cdot \gamma) = O(t)$. Finally, we argue that in the context of such an application, setting $\gamma = 1/t$ is actually the "natural" choice. Such a choice assigns to each proof oracle encountered in the composition an almost equal weight (of $1/t$); that is, such a proof oracle is assigned weight $1/t$ when it appears as the current proof oracle and maintains its weight when it appears as part of the input oracle in subsequent compositions.

3.2. Corollary to Theorem 3.3. Recall that Theorem 3.3 asserts a PCPP (for CktVal) with randomness complexity $\log_2 n + A_t(n)$, where $A_t(n) \triangleq O(t + (\log n)^{1/t}) \log\log n + O((\log n)^{2/t})$ and query complexity $O(t)$ (for soundness error $1/2$). For constant $t \geq 3$, we have $A_t(n) = O((\log n)^{2/t})$. On the other hand, for $t \geq \frac{1.01 \log\log n}{\log\log\log n}$, we have $A_t(n) = o(\log\log n)^2$. Using Proposition 2.4, these PCPPs yield corresponding PCPs for CktSAT.

Deriving Theorems 1.2 *and* 1.3. Two extreme choices of $t(n)$ are when $t(n) = \frac{2}{\varepsilon}$ for some $\varepsilon > 0$ (which maintains a constant query complexity) and $t(n) = \frac{2 \log\log n}{\log\log\log n}$ (which minimizes the randomness complexity of the verifier). Setting $t(n) = \frac{2}{\varepsilon}$ yields Theorem 1.3 (i.e., constant query complexity $O(\frac{1}{\varepsilon})$ and randomness $\log_2 n + O(\log^{\varepsilon} n)$), whereas setting $t(n) = \frac{2 \log\log n}{\log\log\log n}$ yields Theorem 1.2 (i.e., query complexity $O\left(\frac{\log\log n}{\log\log\log n}\right)$ and randomness $\log_2 n + O\left(\frac{(\log\log n)^2}{\log\log\log n}\right)$. Thus, both Theorems 1.2 and 1.3 follow from Theorem 3.3.

Deriving a PCPP for Nondeterministic Circuit Value. By Proposition 2.5, we conclude that for every $3 \leq t(n) \leq \frac{2 \log\log n}{\log\log\log n}$, there exists a PCPP for Nondeterministic Circuit Value of the same complexities (i.e., randomness complexity $\log_2 n + A_t(n)$, query complexity $O(t(n))$, perfect completeness, and soundness error $1/2$ with respect to proximity $\delta = \Omega(1/t(n))$).

A more flexible notion of a PCPP. Our definition of a PCPP (see Definition 2.3) specifies for each system a unique proximity parameter. In many settings (see, e.g., section 4.1), it is better that the proximity parameter is given as an input to the verifier and that the latter behaves accordingly (i.e., makes an adequate number of

queries). We refrain from presenting either a formal definition of such relaxed PCPPs or a general transformation of PCPPs into their more relaxed form. Instead, we state the following corollary to Theorem 3.3.

COROLLARY 3.4. *For all parameters* $n, t_1, t_2 \in \mathbb{N}$ *such that* $3 \leq t_1 \leq t_2 \leq \frac{2 \log \log n}{\log \log \log n}$ *there exists a PCPP for* CKTVAL (*for circuits of size* n) *with proof length* $2^{A_{t_1}(n)} \cdot n$, *where* $A_t(n)$ *is as in* (3.1), *query complexity* $O(t_2)$, *perfect completeness, and soundness error* $1/2$ *with respect to proximity parameter* $1/t_2$. *Furthermore, when given* (as auxiliary input) *a proximity parameter* $\delta \geq 1/t_2$, *the verifier makes only* $O(\max\{1/\delta, t_1\})$ *queries and rejects any input oracle that is* δ*-far from satisfying the circuit with probability at least* $1/2$.

Underlying the following proof is a general transformation of PCPPs to the more relaxed form as stated in Corollary 3.4.

Proof. The proof oracle consists of a sequence of proofs for the system of Theorem 3.3, when instantiated with proximity parameter 2^{-i} for $i = \lfloor \log_2 t_1 \rfloor, \ldots, \lceil \log_2 t_2 \rceil$. When the new verifier is invoked with proximity parameter δ, it invokes the original verifier with proximity parameter 2^{-i}, where $i = \lceil \log_2 1/\delta \rceil$, and emulates the answers using the ith portion of its proof oracle. □

4. Applications to coding problems. In this section we show that, combined with any good code, any PCPP yields a locally testable code (LTC). Using our PCPPs, we obtain an improvement in the rate of LTCs (improving over the results of [GS02, BSVW03]). We also introduce a relaxed notion of locally decodable codes (LDCs) and show how to construct such codes using any PCPP (and ours in particular).

Preliminaries For a string $w \in \{0,1\}^n$ and $i \in [n] \triangleq \{1, 2, \ldots, n\}$, unless stated differently, w_i denotes the ith bit of w.

We consider codes mapping sequences of k (input) bits into sequences of $n \geq k$ (output) bits. Such a generic code is denoted by $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$, and the elements of $\{\mathrm{C}(x) : x \in \{0,1\}^k\} \subseteq \{0,1\}^n$ are called *codewords* (of C). Throughout this section, *the integers* k *and* n *are to be thought of as parameters*, and we are typically interested in the relation of n to k (i.e., how n grows as a function of k). Thus, we actually discuss infinite families of codes (which are associated with infinite sets of possible k's), and whenever we say that some quantity of the code is a constant we mean that this quantity is constant for the entire family (of codes).

The distance of a code $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$ is the minimum (Hamming) distance between its codewords; that is, $\min_{x \neq y}\{\overline{\Delta}(\mathrm{C}(x), \mathrm{C}(y))\}$, where $\overline{\Delta}(u, v)$ denotes the number of bit locations on which u and v differ. Throughout this work, we focus on codes of "linear distance," that is, codes $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$ of distance $\Omega(n)$. The *distance of* $w \in \{0,1\}^n$ *from a code* $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$, denoted $\overline{\Delta}_{\mathrm{C}}(w)$, is the minimum distance between w and the codewords, that is, $\overline{\Delta}_{\mathrm{C}}(w) \triangleq \min_x\{\overline{\Delta}(w, \mathrm{C}(x))\}$. For $\delta \in [0,1]$, the n-bit long strings u and v are said to be δ*-far* (resp., δ*-close*) if $\overline{\Delta}(u, v) > \delta \cdot n$ (resp., $\overline{\Delta}(u, v) \leq \delta \cdot n$). Similarly, w is δ*-far from* C (resp., δ*-close to* C) if $\overline{\Delta}_{\mathrm{C}}(w) > \delta \cdot n$ (resp., $\overline{\Delta}_{\mathrm{C}}(w) \leq \delta \cdot n$).

As in the case of PCPs, all oracle machines considered below are *nonadaptive*. Here these oracle machines will model highly efficient testing and decoding procedures, which probe their input $w \in \{0,1\}^n$ in relatively few places. Thus, these procedures are modeled as oracle machines having oracle access to w (which is viewed as a function $w : \{1, \ldots, n\} \to \{0,1\}$).

4.1. LTCs. Loosely speaking, by a *codeword test* (for the code $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$) we mean a randomized (nonadaptive) oracle machine, also called a *tester*,

that is given oracle access to $w \in \{0,1\}^n$. The tester may query the oracle at a constant number of bit locations and is required to (always) accept every codeword and reject with (relatively) high probability every oracle that is "far" from the code. Indeed, since our focus is on positive results, we use a strict formulation in which the tester is required to accept each codeword with probability 1. (This corresponds to "perfect completeness" in the PCP setting.) The first definition below provides a general template (in terms of several parameters) for the rejection condition. Later we will discuss the kinds of asymptotic parameters we would like to achieve.

DEFINITION 4.1 (codeword tests). *A randomized* (nonadaptive) *oracle machine M is called a (δ, s)-codeword test for $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$ if it satisfies the following two conditions:*

1. Accepting codewords (i.e., completeness)*: For every $x \in \{0,1\}^k$, given oracle access to $w = \mathrm{C}(x)$, machine M accepts with probability* 1. *That is,* $\Pr[M^{\mathrm{C}(x)}=1] = 1$ *for every $x \in \{0,1\}^k$.*

2. Rejection of noncodeword (i.e., soundness)*: Given oracle access to any $w \in \{0,1\}^n$ that is δ-far from* C*, machine M accepts with probability at most s. That is,* $\Pr[M^w=1] \leq s$ *for every $w \in \{0,1\}^n$ that is δ-far from* C.

The parameter δ is called the proximity parameter *and s is called the* soundness error. *The* query complexity *q of M is the maximum number of queries it makes (taken over all sequences of coin tosses).*

Note that *this definition requires nothing with respect to noncodewords that are relatively close to the code* (i.e., are δ-close to C). In addition to the usual goals in constructing error-correcting codes (e.g., maximizing minimum distance and minimizing the blocklength $n = n(k)$), here we are also interested in simultaneously minimizing the query complexity q, the proximity parameter δ, and the soundness error s. More generally, we are interested in the trade-off between q, δ, and s. (As usual, the soundness error can be reduced to s^k by increasing the query complexity to $k \cdot q$.) A minimalistic goal is to have a family of codes with q, δ, and s all fixed constants. However, note that this would be interesting only if δ were sufficiently small with respect to the distance parameters of the code, e.g., smaller than half the relative minimum distance. (For example, if δ is larger than the "covering radius" of the code, then there does not exist any string that is δ-far from the code, and the soundness condition becomes vacuous.) A stronger definition requires the tester to work for any *given* proximity parameter $\delta > o(1)$, but allows its query complexity to depend on δ as follows.

DEFINITION 4.2 (LTCs). *A family of codes $\{\mathrm{C}_k : \{0,1\}^k \to \{0,1\}^n\}_{k \in \mathbb{N}}$ is* locally testable *if it satisfies the following.*

1. Linear distance*: There is a constant $\rho > 0$, such that for every k, C_k has minimum distance at least $\rho \cdot n$.*

2. Local testability*: There is a randomized, nonadaptive oracle machine M such that for every constant $\delta > 0$, there is a constant $q = q(\delta)$ such that for all sufficiently large k, $M^w(1^k, \delta)$ is a $(\delta, 1/2)$-codeword test for C_k with query complexity q.*

The family is called explicit *if both C_k and $M^w(1^k, \delta)$ can be evaluated with computation time polynomial in k.*

We comment that Definition 4.2 is somewhat weaker than the definitions used in [GS02].[14]

[14] In the stronger of the definitions in [GS02], the tester is not given δ as input (and thus has query complexity that is a fixed constant independent of δ) but is required to be a $(\delta, 1 - \Omega(\delta))$-codeword test for every constant $\delta > 0$ and sufficiently large k. That is, strings that are δ-far from the code

Using an adequate PCPP, we can transform any code to a related code that has a codeword tester. This is done by appending each codeword with a PCPP proving the codeword is indeed the encoding of a message. One technical problem that arises is that the PCPP constitutes most of the length of the new encoding. Furthermore, we cannot assume much about the Hamming distance between different proofs of the same statement, and thus the distance of the new code may deteriorate. But this is easily fixed by repeating the codeword many times so that the PCPP constitutes only a small fraction of the total length.[15] Specifically, given a code $\mathrm{C}_0 : \{0,1\}^k \to \{0,1\}^m$, we consider the code $\mathrm{C}(x) \triangleq (\mathrm{C}_0(x)^t, \pi(x))$, where $t = (d(k)-1) \cdot |\pi(x)|/|\mathrm{C}_0(x)|$ such that (say) $d(k) = \log k$, and $\pi(x)$ is a PCPP that asserts that an m-bit string (given as an input oracle) is a codeword (of C_0).

CONSTRUCTION 4.3. *Let d be a free parameter to be determined later, let $\mathrm{C}_0 : \{0,1\}^k \to \{0,1\}^m$ be a code, and let V be a PCPP verifier for membership in $S_0 = \{\mathrm{C}_0(x) : x \in \{0,1\}^k\}$. Let $\pi(x)$ be the proof oracle corresponding to the claim that the input oracle equals $\mathrm{C}_0(x)$; that is, $\pi(x)$ is the canonical proof obtained by using x as an NP witness for membership of $\mathrm{C}_0(x)$ in S_0. Consider the code $\mathrm{C}(x) \triangleq (\mathrm{C}_0(x)^t, \pi(x))$, where $t = (d-1) \cdot |\pi(x)|/|\mathrm{C}_0(x)|$.*

The codeword test emulates the PCP verifier in the natural way. Specifically, given oracle access to $w = (w_1, \ldots, w_t, \pi) \in \{0,1\}^{t \cdot m + \ell}$, the codeword tester selects uniformly $i \in [t]$ and emulates the PCP verifier, providing it with oracle access to the input oracle w_i and to the proof oracle π. In addition, the tester checks that the repetitions are valid (by inspecting randomly selected positions in some q_{rep} randomly selected pairs of m-bit long blocks, where q_{rep} is a free parameter to be optimized later). Let us denote this tester by T. That is, T^w

1. uniformly selects $i \in [t]$ and invokes $V^{w_i, \pi}$.
2. repeats the following q_{rep} times: Uniformly selects $i_1, i_2 \in [t]$ and $j \in [m]$ and checks whether $(w_{i_1})_j = (w_{i_2})_j$.

PROPOSITION 4.4. *Let d and q_{rep} be the free parameters in the above construction of the code C and tester T. Suppose that the code $\mathrm{C}_0 : \{0,1\}^k \to \{0,1\}^m$ has a relative minimum distance of ρ_0, and that the PCPP has a proof length of $\ell > m$, soundness error $1/4$ for proximity parameter δ_{pcpp}, and query complexity q_{pcpp}. Then, the code C and tester T have the following properties:*

1. *The blocklength of C is $n \triangleq d \cdot \ell$ and its relative minimum distance is at least $\rho_0 - 1/d$.*
2. *The oracle machine T is a $(\delta, \frac{1}{2})$-codeword tester for the C, where $\delta = \delta_{\mathrm{pcpp}} + \frac{4}{q_{\mathrm{rep}}} + \frac{1}{d}$.*
3. *The query complexity of T is $q = q_{\mathrm{pcpp}} + 2q_{\mathrm{rep}}$.*

Proof. The parameters of the code C are obvious from the construction. In particular, C has blocklength $t \cdot m + \ell = d \cdot \ell = n$, and the PCPP $\pi(x)$ constitutes only an $\ell/n = 1/d$ fraction of the length of the codeword $\mathrm{C}(x)$. Since the remainder consists of replicated versions of $\mathrm{C}_0(x)$, it follows that the relative minimum distance of C is at least $(n-\ell)\rho_0/n > \rho_0 - 1/d$.

The query complexity of T is obvious from its construction, and so we need only show that it is a good codeword tester. Completeness follows immediately from

---

are rejected with probability $\Omega(\delta)$. Such a tester implies a tester as in Definition 4.2, with query complexity $q(\delta) = O(1/\delta)$.

[15] Throughout this section we will use repetitions to adjust the "weights" of various parts of our codes. An alternative method would be to work with weighted Hamming distance (i.e., where different coordinates of a codeword receive different weights), and indeed these two methods (weighting and repeating) are essentially equivalent. For the sake of explicitness we work only with repetitions.

the completeness of the PCPP, and so we focus on the soundness condition. We consider an arbitrary $w = (w_1, \ldots, w_t, \pi) \in \{0,1\}^{t \cdot m + \ell}$ that is δ-far from C, and observe that $w' = (w_1, \ldots, w_t)$ must be δ'-far from $\mathrm{C}' = \{\mathrm{C}_0(x)^t : x \in \{0,1\}^k\}$, where $\delta' \geq (\delta n - \ell)/n = \delta - (1/d)$. Let $u \in \{0,1\}^m$ be a string that minimizes $\Delta(w', u^t) = \sum_{i=1}^t \Delta(w_i, u)$; that is, u^t is the "repetition sequence" closest to w'. We consider two cases:

Case 1. $\Delta(w', u^t) \geq 1/q_{\mathrm{rep}}$. In this case, a single execution of the basic repetition test (comparing two locations) rejects with probability:

$$\begin{aligned} \mathrm{E}_{r,s \in [t]} \left[\Delta(w_r, w_s)\right] &\geq \mathrm{E}_{r \in [t]} \left[\Delta(w_r, u)\right] \\ &= \Delta(w', u^t) \\ &\geq 1/q_{\mathrm{rep}}, \end{aligned}$$

where the last inequality is due to the case hypothesis. It follows that q_{rep} executions of the repetition test would accept with probability at most $(1 - 1/q_{\mathrm{rep}})^{q_{\mathrm{rep}}} < 1/e < 1/2$.

Case 2. $\Delta(w', u^t) \leq 1/q_{\mathrm{rep}}$. In this case

$$\Delta_{\mathrm{C}_0}(u) = \Delta_{\mathrm{C}'}(u^t) \geq \Delta_{\mathrm{C}'}(w') - \Delta(w', u^t) \geq \delta' - \frac{1}{q_{\mathrm{rep}}},$$

where the last inequality is due to the case hypothesis. Also, recalling that on the average (i.e., average i) w_i is $1/q_{\mathrm{rep}}$-close to u, it holds that at least two-thirds of the w_i's are $3/q_{\mathrm{rep}}$-close to u. Recalling that u is $(\delta' - (1/q_{\mathrm{rep}}))$-far from C_0 and using $\delta_{\mathrm{pcpp}} = \delta' - (4/q_{\mathrm{rep}})$, it follows that at least two-thirds of the w_i's are δ_{pcpp}-far from C_0. Thus, by the soundness condition of the PCP of proximity, these w_i will be accepted with probability at most $1/4$. Thus, in the current case, the tester accepts with probability at most $\frac{1}{3} + \frac{2}{3} \cdot \frac{1}{4} = \frac{1}{2}$. The soundness condition follows. ∎

To prove Theorem 1.4, we instantiate the above construction as follows. We let $\mathrm{C}_0 : \{0,1\}^k \to \{0,1\}^m$ come from a family of codes with a constant relative minimum distance of $\rho_0 > 0$ and nearly linear blocklength $m = \widetilde{O}(k)$, where encoding can be done by circuits of nearly linear size $s_0 = s_0(k) = \widetilde{O}(k)$. We take the PCPP from Corollary 3.4, setting $t_1 = O(1/\varepsilon)$ (for an arbitrarily small constant $\varepsilon > 0$) and $t_2 = 2 \log \log s_0 / \log \log \log s_0 = \omega(1)$. Thus, we obtain proof length $\ell = s_0 \cdot \exp(\log^{\varepsilon/2} s_0)$ and query complexity $q_{\mathrm{pcpp}} = O(\max\{1/\delta_{\mathrm{pcpp}}, t_1\}) = O(1/\delta_{\mathrm{pcpp}})$ for any proximity parameter $\delta_{\mathrm{pcpp}} \geq 1/t_2 = o(1)$. We actually invoke the verifier twice to reduce its soundness error to $1/4$. Setting $d = \log k = \omega(1)$, we obtain a final blocklength of $n = d \cdot \ell < k \cdot \exp(\log^{\varepsilon} k)$ and relative distance $\rho_0 - o(1)$. We further specify the test T as follows. Given a proximity parameter $\delta \geq 6/t_2 = o(1)$, the tester T invokes the aforementioned PCPP with $\delta_{\mathrm{pcpp}} = \delta/6$ and performs the repetition test $q_{\mathrm{rep}} = 6/\delta$ times. Observing that $\delta_{\mathrm{pcpp}} + (4/q_{\mathrm{rep}}) + (1/d) < \delta$, we conclude that the resulting test (i.e., $T = T(1^k, \delta_{\mathrm{pcpp}})$) is a $(\delta, 1/2)$-codeword tester of query complexity $O(1/\delta_{\mathrm{pcpp}}) + 2q_{\mathrm{rep}} = O(1/\delta)$. Thus we conclude as follows.

Conclusion (restating Theorem 1.4). *For every constant $\varepsilon > 0$, there exists a family of LTCs $\mathrm{C}_k : \{0,1\}^k \to \{0,1\}^n$, where $n = \exp(\log^{\varepsilon} k) \cdot k$, with query complexity $q(\delta) = O(1/\delta)$ (i.e., for every $\delta > 0$, the tester rejects words that are δ-far from C with probability $1/2$ by querying at most $q(\delta) = O(1/\delta)$ queries).*

4.2. Relaxed LDCs. We first recall the definition of LDCs, as formally stated by Katz and Trevisan [KT00]. A code $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$ is *locally decodable* if for some constant $\delta > 0$ (which is independent of k) there exists an efficient oracle

machine M that, on input, any index $i \in [k]$ and access to any oracle $w \in \{0,1\}^n$ such that $\Delta(w, \mathrm{C}(x)) \leq \delta$, recovers the ith bit of x with probability at least $2/3$ while making a constant number of queries to w. That is, whenever relatively few locations are corrupted, the decoder should be able to recover each information bit, with high probability, based on a constant number of queries to the (corrupted) codeword.

Katz and Trevisan showed that if M makes q queries, then $n = \Omega(k^{1+1/(q-1)})$ must hold [KT00].[16] This lower-bound is quite far from the best known upper-bound, due to Beimal et al. [BIKR02], that asserts $n = O(\exp(k^{\varepsilon(q)}))$, where $\varepsilon(q) = O((\log\log q)/(q \log q)) = o(1/q)$, which improves (already for $q = 4$) a previous upper-bound where $\varepsilon(q) = 1/(2q+1)$. It has been conjectured that, for a constant number of queries, n should be exponential in k; that is, for every constant q there exists a constant $\varepsilon > 0$ such that $n > \exp(k^{\varepsilon})$ must hold. In view of this state of affairs, it is natural to relax the definition of LDCs, with the hope of obtaining more efficient constructions (e.g., $n = \mathrm{poly}(k)$).

We relax the definition of LDCs by requiring that, whenever few location are corrupted, the decoder should be able to recover most (or almost all) of the individual information bits (based on few queries), and for the remaining locations the decoder outputs either the right message bit or a fail symbol (but not the wrong value). That is, the decoder must still avoid errors (with high probability), but is allowed to say "`don't know`" on a few bit locations. The following definition is actually weaker, yet, the (aforementioned) stronger formulation is obtained when considering $\rho \approx 1$ (and using amplification to reduce the error from $1/3$ to any desired constant).[17] Furthermore, it is desirable to recover all bits of the information whenever the codeword is not corrupted.

DEFINITION 4.5 (relaxed LDC). *A code* $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$ *is* relaxed locally decodable *if for some constants* $\delta, \rho > 0$ *there exists an efficient probabilistic oracle machine* M *that makes a constant number of queries and satisfies the following three conditions with respect to any* $w \in \{0,1\}^n$ *and* $x \in \{0,1\}^k$ *such that* $\Delta(w, \mathrm{C}(x)) \leq \delta$*:*

1. *If* $w = \mathrm{C}(x)$ *is a codeword, then the decoder correctly recovers every bit of* x *with probability at least* $\frac{2}{3}$*. That is, for every* $x \in \{0,1\}^k$ *and* $i \in [k]$*, it holds that* $\Pr[M^{\mathrm{C}(x)}(i) = x_i] \geq \frac{2}{3}$.

2. *On input, any index* $i \in [k]$ *and given access to the oracle* w*, with probability at least* $\frac{2}{3}$ *machine* M *outputs either the* i*th bit of* x *or a special failure symbol, denoted* $\perp$*. That is, for every* i*, it holds that* $\Pr[M^w(i) \in \{x_i, \perp\}] \geq \frac{2}{3}$.

3. *For at least a* ρ *fraction of the indices* $i \in [k]$*, on input* i *and oracle access to* $w \in \{0,1\}^n$*, with probability at least* $\frac{2}{3}$*, machine* M *outputs the* i*th bit of* x*. That is, there exists a set* $I_w \subseteq [k]$ *of size at least* ρk *such that for every* $i \in I_w$ *it holds that* $\Pr[M^w(i) = x_i] \geq \frac{2}{3}$.

We call δ *the* proximity parameter.

One may strengthen the definition by requiring that ρ be greater than $1/2$ or any other favorite constant smaller than 1 (but one should probably refrain from setting $\rho > 1 - \delta$, for example). A different strengthening is for condition 1 to hold with

[16] Their lower-bound refers to nonadaptive decoders and yields a lower-bound of $n = \Omega(k^{1+1/(2^q-1)})$ for adaptive decoders. A lower-bound of $n = \Omega(k^{1+1/O(q)})$ for adaptive decoders was presented in [DJK+02], and a lower-bound of $n = \Omega(k^{1+1/(q/2-1)})$ for nonadaptive decoders was presented in [KdW04]. (We note that we use a nonadaptive (relaxed) decoder.)

[17] Here error reduction may be performed by estimating the probability that the machine outputs each of the possible bits, and outputting the more frequent bit only if it has sufficient statistical support (e.g., 50% support, which the wrong bit cannot have). Otherwise, one outputs the `don't know` symbol.

probability 1 (i.e., $\Pr[M^{\mathrm{C}(x)}(i) = x_i] = 1$). In fact, we achieve both the stronger forms.

Remark 4.6. The above definition refers only to strings w that are δ-close to the code. However, using Construction 4.3, any relaxed LDC can be augmented so that strings that are δ-far from the code are rejected with high probability (i.e., for every index i, the decoder outputs $\perp$ with high probability). This can be achieved with only a nearly linear increase in the length of the code (from length n to length $n \cdot \exp(\log^{\varepsilon} n)$).

Remark 4.7. We stress that condition 2 does not mean that, for every i, and for every w that is δ-close to $\mathrm{C}(x)$, either $\Pr[M^w(i) = x_i] \geq \frac{2}{3}$ or $\Pr[M^w(i) = \perp] \geq \frac{2}{3}$ holds. We refer to the latter condition as condition X and conjecture that the seemingly minor difference between conditions 2 and X is actually substantial. This conjecture is enforced by a recent work of Buhrman and de Wolf [BdW04] who showed that codes that satisfy condition X are actually locally decodable in the standard, nonrelaxed sense (i.e., according to the definition of [KT00]).

4.2.1. Definitional issues and transformations. Note that it is very easy to come up with constructions that satisfy each one of the three conditions of Definition 4.5. For example, condition 2 can be satisfied by (any code and) a trivial decoder that always returns $\perp$. On the other hand, the identity encoding (combined with a trivial decoder) satisfies conditions 1 and 3.[18] Our aim, however, is to obtain a construction that satisfies all conditions and beats the performance of the known LDCs.

It turns out that codes that satisfy conditions 1 and 2 can be converted into "equally good" codes that satisfy all three conditions. Let us start with a key definition, which refers to the distribution of the decoder's queries *when asked to recover a random bit position.*

DEFINITION 4.8 (average smoothness). *Let M be a randomized nonadaptive oracle machine having access to an oracle $w \in \{0,1\}^n$ and getting input $i \in [k]$. Further suppose that M always makes q queries. Let $M(i,j,r)$ denote the jth query of M on input i and coin tosses r. We say that M satisfies the* average smoothness *condition if, for every $v \in [n]$,*

$$\frac{1}{2n} < \Pr_{i,j,r}[M(i,j,r) = v] < \frac{2}{n},$$

where the probability is taken uniformly over all possible choices of $i \in [k]$, $j \in [q]$, and coin tosses r.

By having M randomly permute its queries, average smoothness implies that for every $j \in [q]$ and $v \in [n]$, it holds that $\frac{1}{2n} < \Pr_{i,r}[M(i,j,r) = v] < \frac{2}{n}$, where now the probability is taken uniformly over all possible choices of $i \in [k]$ and the coin tosses r. We stress that *average smoothness is different from the notion of smoothness as defined by Katz and Trevisan* [KT00]: They require that *for every* $i \in [k]$ (and for every $j \in [q]$ and $v \in [n]$), it holds that $\frac{1}{2n} < \Pr_r[M(i,j,r) = v] < \frac{2}{n}$. Indeed, average smoothness is a weaker requirement, and (as we will shortly see) any code and decoder pair can be easily modified to satisfy it, while preserving the decoding properties (of Definition 4.5). (In contrast, Katz and Trevisan [KT00] present a

[18] In case one wishes the code to have a linear distance this can be achieved too, consider $\mathrm{C}(x) = (x, \mathrm{C}'(x))$, where C' is any code of linear length and linear distance, and a decoder that merely retrieves the desired bit from the first part.

modification that achieves smoothness while preserving strict local decodability, but their transformation does not preserve Definition 4.5.)

LEMMA 4.9. *Let* $\mathrm{C}:\{0,1\}^k \to \{0,1\}^n$ *be a code and* M *a machine that satisfies conditions* 1 *and* 2 *of Definition* 4.5 *with respect to proximity parameter* δ*. Then, for some* $n' \in [3n, 4n]$*, there exists a code* $\mathrm{C}':\{0,1\}^k \to \{0,1\}^{n'}$ *and a machine* M' *that satisfies average smoothness as well as conditions* 1 *and* 2 *of Definition* 4.5 *with respect to proximity parameter* $\delta' = \delta/20$*. Furthermore, the query complexity of* M' *is twice the one of* M*, and if* M *satisfies also condition* 3*, with respect to a constant* ρ*, then so does* M'*.*

Jumping ahead, we mention that, for a decoder that satisfies average smoothness, conditions 1 and 2 essentially imply condition 3. Hence our interest in average smoothness and in Lemma 4.9.

Proof. As noted above, we may assume without loss of generality that each of M's queries is distributed identically. Throughout the analysis, we refer to the distribution of queries for a uniformly distributed index $i \in [k]$. Let q denote the query complexity of M.

We first modify M such that for a random $i \in [k]$, each query probes each possible location with probability $\Omega(1/n)$. This is done by adding q dummy queries, each being uniformly distributed. Thus, each location gets probed by each query with probability at least $1/2n$.

Next we modify the code and the decoder such that each location is probed with almost uniform distribution. The idea is to repeat heavily probed locations for an adequate number of times and have the decoder probe a random copy. Specifically, let p_v be the probability that location v is probed (i.e., $p_v \triangleq \Pr_{i\in[k],r}[M(i,1,r)=v]$ or equivalently $p_v = \sum_{i\in[k],j\in[2q]} \Pr_r[M(i,j,r)=v]/2kq$). By the above modification, we have $p_v \geq 1/2n$. Now, we repeat location v $r_v = \lfloor 4np_v \rfloor$ times. Note that $r_v \leq 4np_v$ and $r_v > 4np_v - 1 \geq 2 - 1$ (and so $r_v \geq 2$). We obtain a new code C' of length $n' = \sum_v r_v \leq 4n$. (Note that $n' > 3n$.) The relative distance of C' is at least one-fourth that of C, and the rate changes in the same way. The new decoder, M', when seeking to probe location v, will select and probe at random one of the r_v copies of that location. (Interestingly, there is no need to augment this decoder by a testing of the consistency of the copies of an original location.)

Each new location is probed with probability $p'_v \triangleq p_v \cdot \frac{1}{r_v}$ (by each of these queries). Recalling that $\frac{p_v}{r_v} = \frac{p_v}{\lfloor 4np_v \rfloor}$, it follows that $p'_v \geq 1/4n$ and $p'_v \leq \frac{p_v}{4np_v - 1} \leq 1/2n$ (using $p_v \geq 1/2n$). Recalling that $n' \in [3n, 4n]$, each p'_v is in $[(3/4)\cdot(1/n'), 2\cdot(1/n')]$, i.e., within a factor of 2 from uniform.

Clearly, M' satisfies condition 1 (of Definition 4.5), and we show that it (essentially) satisfies condition 2 as well. Let $w = (w_1, \ldots, w_n) \in \{0,1\}^{n'}$ be δ'-close to $\mathrm{C}'(x)$, where $|w_v| = r_v$. Let Y_v be a 0-1 random variable that represents the value of a random bit in w_v; that is, $\Pr[Y_v = 1]$ equals the fraction of 1's in w_v. Then, $\Pr[Y_v \neq \mathrm{C}(x)_v] > 0$ implies that $\overline{\Delta}(w_v, c_v) \geq 1$, where $\mathrm{C}'(x) = (c_1, \ldots, c_n)$ and $|c_v| = r_v$. For $Y = Y_1 \cdots Y_n$, it follows that $\mathrm{E}(\overline{\Delta}(Y, \mathrm{C}(x))) \leq \overline{\Delta}(w, \mathrm{C}'(x))$, and so $\mathrm{E}(\overline{\Delta}(Y, \mathrm{C}(x))) \leq \delta' n' \leq (\delta/5)\cdot n$ (since $\delta' = \delta/20$ and $n' \leq 4n$). Thus, with probability at least $4/5$, the random string Y is δ-close to $\mathrm{C}(x)$, in which case the M must succeed with probability at least $2/3$. Noting that $M'^w(i)$ merely invokes $M^Y(i)$, we conclude that

$$\begin{aligned}\Pr[M'^w(i) \in \{x_i, \perp\}] &= \Pr[M^Y(i) \in \{x_i, \perp\}] \\ &\geq \Pr[\overline{\Delta}(Y, \mathrm{C}(x)) \leq \delta n] \cdot \Pr[M^Y(i) \in \{x_i, \perp\} \mid \overline{\Delta}(Y, \mathrm{C}(x)) \leq \delta n] \\ &\geq \frac{4}{5} \cdot \frac{2}{3} = \frac{8}{15}.\end{aligned}$$

An analogous argument can be applied in the case M satisfies condition 3. In both cases, additional error reduction is needed in order to satisfy the actual conditions, which require success with probability at least 2/3. (For details see footnote 17.) □

LEMMA 4.10. *Let* $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$ *be a code and* M *be a machine that satisfies conditions* 1 *and* 2 *of Definition* 4.5 *with respect to a constant* δ. *Suppose that* M *satisfies the average smoothness condition and has query complexity* q. *Then, invoking* M *for a constant number of times (and ruling as in footnote* 17*) yields a decoder that satisfies all three conditions of Definition* 4.5. *Specifically, condition* 3 *holds with respect to a constant* $\rho = 1 - 18q\delta$. *Furthermore, for any* w *and* x, *for a* $1 - 18q\overline{\Delta}(w, \mathrm{C}(x))$ *fraction of the* i*'s, it holds that* $\Pr[M^w(i) = x_i] \geq 5/9$.

Our usage of the average smoothness condition actually amounts to using the hypothesis that, for a uniformly distributed $i \in [k]$, each query hits any fixed position with probability at most $2/n$.

Proof. By condition 1, for any $x \in \{0,1\}^k$ and every $i \in [k]$, it holds that $\Pr[M^{\mathrm{C}(x)}(i) = x_i] \geq 2/3$. Considering any w that is δ-close to $\mathrm{C}(x)$, the probability that on input a *uniformly distributed* $i \in [k]$ machine M queries a location on which w and $\mathrm{C}(x)$ disagree is at most $q \cdot (2/n) \cdot \delta n = 2q\delta$. This is due to the fact that, for a uniformly distributed i, the queries are almost uniformly distributed; specifically, no position is queried with probability greater than $2/n$ (by a single query).

Let p_i^w denote the probability that on input i machine M queries a location on which w and $\mathrm{C}(x)$ disagree. We have just established that $(1/k) \cdot \sum_{i=1}^k p_i^w \leq 2q\delta$. For $I_w \triangleq \{i \in [k] : p_i^w \leq 1/9\}$, it holds that $|I_w| \geq (1 - 18q\delta) \cdot k$. Observe that for any $i \in I_w$, it holds that $\Pr[M^w(i) = x_i] \geq (2/3) - (1/9) = 5/9$. Note that, by replacing δ with $\overline{\Delta}(w, \mathrm{C}(x))/n$, the above argument actually establishes that for a $1 - 18q \cdot \overline{\Delta}(w, \mathrm{C}(x))$ fraction of the i's, it holds that $\Pr[M^w(i) = x_i] \geq 5/9$.

Additional error reduction is needed in order to satisfy the actual definition (of condition 3), which requires success with probability at least 2/3. The error reduction should be done in a manner that preserves conditions 1 and 2 of Definition 4.5. For details see footnote 17. □

In view of the last sentence of Lemma 4.10, it makes sense to state a stronger definition of relaxed LDCs.

DEFINITION 4.11 (relaxed LDC, revisited). *A code* $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$ *is* relaxed locally decodable *if for some constants* $\delta > 0$ *there exists an efficient probabilistic oracle machine* M *that makes a constant number of queries and satisfies the following two conditions with respect to any* $w \in \{0,1\}^n$ *and* $x \in \{0,1\}^k$ *such that* $\overline{\Delta}(w, \mathrm{C}(x)) \leq \delta n$*:*

1. *For every* $i \in [k]$ *it holds that* $\Pr[M^w(i) \in \{x_i, \perp\}] \geq \frac{2}{3}$.
2. *There exists a set* $I_w \subseteq [k]$ *of density at least* $1 - O(\overline{\Delta}(w, \mathrm{C}(x))/n)$ *such that for every* $i \in I_w$ *it holds that* $\Pr[M^w(i) = x_i] \geq \frac{2}{3}$.

Note that the "everywhere good" decoding of codewords (i.e., condition 1 of Definition 4.5) is implied by condition 2 of Definition 4.11. By combining Lemmas 4.9 and 4.10, we get the following theorem.

THEOREM 4.12. *Let* $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$ *be a code and* M *be a machine that makes a constant* q *number of queries and satisfies conditions* 1 *and* 2 *of Definition* 4.5 *with respect to a constant* δ. *Then, for some* $n' \in [3n, 4n]$, *there exists a code* $\mathrm{C}' : \{0,1\}^k \to \{0,1\}^{n'}$ *that is relaxed locally decodable with respect to proximity parameter* $\delta' = \delta/20$. *Furthermore, this code satisfies Definition* 4.11.

4.2.2. Constructions. In view of Lemma 4.10, we focus on presenting codes with decoders that satisfy conditions 1 and 2 of Definition 4.5 as well as the average

smoothness property. (The latter property will save us the need to invoke Lemma 4.9.) We will start with a code that has nearly quadratic length (i.e., $n = k^{2+o(1)}$), which serves as a good warm-up towards our final construction in which $n = k^{1+\varepsilon}$ for any desired constant $\varepsilon > 0$.

Motivation to our construction. We seek a code of linear distance that has some weak "local decodability" properties. One idea is to separate the codeword into two parts, the first allowing for "local decodability" (e.g., using the identity map) and the second providing the distance property (e.g., using any code of linear distance). It is obvious that a third part that guarantees the consistency of the first two parts should be added, and it is natural to try to use a PCPP in the latter part. The natural decoder will check consistency (via the PCPP), and in case it detects no error will decode according to the first part. Indeed, the first part may not be "robust to corruption" but the second part is "robust to corruption" and consistency means that both parts encode the same information. Considering this vague idea, we encounter two problems. First, a PCPP is unlikely to detect a small change in the first part. Thus, if we use the identity map in the first part, then the decoder may output the wrong value of some (although few) bits. In other words, the "proximity relaxation" in PCPPs makes sense for the second part of the codewords but not for the first part. Our solution is to provide, *for each bit* (position) in the first part, a proof of the consistency of this bit (value) with the entire second part. The second problem is that the PCPPs (let alone all of them combined) are much longer than the first two parts, whereas the corruption rate is measured in terms of the entire codeword. This problem is easy to fix by repeating the first two parts sufficiently many times. However, it is important not to "overdo" this repetition because if the third part is too short, then corrupting it may prevent meaningful decoding (as per condition 3 of Definition 4.5) even at low corruption rates (measured in terms of the entire codeword). In other words, if the third part is too short, then we have no chance to satisfy the average smoothness condition.

The actual construction. Let $\mathrm{C}_0 : \{0,1\}^k \to \{0,1\}^m$ be a good code of relative distance δ_0; then we encode $x \in \{0,1\}^k$ by $\mathrm{C}(x) \triangleq (x^t, \mathrm{C}_0(x)^{t'}, \pi_1(x), \ldots, \pi_k(x))$, where $t = |\pi_1(x), \ldots, \pi_k(x)|/|x|$ (resp., $t' = |\pi_1(x), \ldots, \pi_k(x)|/|\mathrm{C}_0(x)|$), and $\pi_i(x)$ is a PCPP to be further discussed. We first note that the replicated versions of x (resp., $\mathrm{C}_0(x)$) take a third of the total length of $\mathrm{C}(x)$. As for $\pi_i(x)$, it is a PCPP that refers to an input of the form $(z_1, z_2) \in \{0,1\}^{m+m}$ and asserts that there exists an $x = x_1 \cdots x_k$ (indeed the one that is a parameter to π_i) such that $z_1 = x_i^m$ and $z_2 = \mathrm{C}_0(x)$.[19] We use our PCPP from Theorem 3.3, while setting its parameters such that the proximity parameter is small enough but the query complexity is a constant. Specifically, let $\delta_{\mathrm{pcpp}} > 0$ be the proximity parameter of the PCPP, which will be set to be sufficiently small, and let $q = O(1/\delta_{\mathrm{pcpp}})$ denote the number of queries the verifier makes in order to support a soundness error of 1/6 (rather than the standard 1/2). A key observation regarding this verifier is that its queries to its input-oracle are uniformly distributed. The queries to the proof oracle can be made almost uniform by a modification analogous to the one used in the proof of Lemma 4.9.

Observe that the code C maps k-bit long strings to codewords of length $n \triangleq 3 \cdot k \cdot \ell$, where $\ell = s_0(m)^{1+o(1)}$ denotes the length of the PCPP proof and $s_0(m)$ denotes the size of the circuit for encoding relative to C_0. Using a good code $\mathrm{C}_0 : \{0,1\}^k \to \{0,1\}^m$ (i.e., of constant relative distance δ_0, linear length $m = O(k)$, and $s_0(m) = \widetilde{O}(m)$), we obtain $n = k^{2+o(1)}$. The relative distance of C is at least $\delta_0/3$.

[19] Indeed, z_1 is merely the bit x_i repeated $|C_0(x)|$ times in order to give equal weight to each part in measuring proximity.

We now turn to the description of the decoder D. Recall that a valid codeword has the form $(x^t, \mathrm{C}_0(x)^{t'}, \pi_1(x), \ldots, \pi_k(x))$. The decoding of the ith information bit (i.e., x_i) will depend on a random (possibly wrong) copy of x_i located in the first part (which supposedly equals x^t), a random (possibly corrupted) copy of $\mathrm{C}_0(x)$ located in the second part, and the relevant (i.e., ith) proof located in the third part (which is also possibly corrupted). On input $i \in [k]$ and oracle access to $w = (w_1, w_2, w_3) \in \{0,1\}^n$, where $|w_1| = |w_2| = |w_3|$, the decoder invokes the PCPP verifier while providing it with access to an input oracle (z_1, z_2) and a proof oracle π that are defined and emulated as follows: The decoder selects uniformly $r \in [t]$ and $r' \in [t']$, and defines each bit of z_1 to equal the $((r-1)k+i)$th bit of w_1, the string z_2 is defined to equal the r'th (m-bit long) block of w_2, and π is defined to equal the ith block (ℓ-bit long) of w_3. That is, when the verifier asks to access the jth bit of z_1 (resp., z_2, resp., π), the decoder answers with the $((r-1)k+i)$th bit of w_1 (resp., $((r'-1)m+j)$th bit of w_2 (resp., the $((i-1)\ell+j)$th bit of w_3). If the verifier rejects, then the decoder outputs a special (failure) symbol. Otherwise, it outputs the $((r-1)k+i)$th bit of w_1.

The above construction can be performed for any sufficiently small constant proximity parameter $\delta \in (0, \delta_0/18)$. All that this entails is setting the proximity parameter of the PCPP to be sufficiently small but positive (e.g., $\delta_{\mathrm{pcpp}} = (\delta_0 - 18\delta)/2$). We actually need to augment the decoder such that it makes an equal number of queries to each of the three (equal length) parts of the codeword, which is easy to do by adding (a constant number of) dummy queries. Let us denote the resulting decoder by D.

PROPOSITION 4.13. *The above code and decoder satisfy conditions* 1 *and* 2 *of Definition* 4.5 *with respect to proximity parameter* $\delta \in (0, \delta_0/18)$. *Furthermore, this decoder satisfies the average smoothness property.*

Proof. Condition 1 (of Definition 4.5) is obvious from the construction (and the completeness property of the PCPP). In fact, the perfect completeness of the PCPP implies that bits of an uncorrupted codeword are recovered with probability 1 (rather than with probability at least 2/3). The average smoothness property of the decoder is obvious from the construction and the smoothness property of the PCPP. We thus turn to establish condition 2 (of Definition 4.5).

Fixing any $x \in \{0,1\}^k$, we consider an arbitrary oracle $w = (w_1, w_2, w_3)$ that is δ-close to $\mathrm{C}(x)$, where w_1 (resp., w_2) denotes the alleged replication of x (resp., $\mathrm{C}_0(x)$) and $w_3 = (u_1, \ldots, u_k)$ denotes the part of the PCPPs. Note that w_2 is 3δ-close to $\mathrm{C}_0(x)^{t'}$. To analyze the performance of $D^w(i)$, we define random variables Z_1 and Z_2 that correspond to the input oracles to which the PCP verifier is given access. Specifically, $Z_1 = \sigma^m$, where σ is set to equal the $((r-1)k+i)$th bit of w_1, when r is uniformly distributed in $[t]$. Likewise, Z_2 is determined to be the r'th block of w_2, where r' is uniformly distributed in $[t']$. Finally, we set the proof oracle, π, to equal the ith block of w_3 (i.e., $\pi = u_i$). We bound the probability that the decoder outputs $\neg x_i$ by considering three cases as follows.

Case 1. $\sigma = x_i$. Recall that σ is the bit read by D from w_1, and that by construction D always outputs either σ or $\perp$. Thus, in this case, Condition 2 is satisfied (because, regardless of whether D outputs σ or $\perp$, the output is always in $\{x_i, \perp\}$).

Case 2. Z_2 is 18δ-far from $\mathrm{C}_0(x)$. Recall that w_2 is 3δ-close to $\mathrm{C}_0(x)^{t'}$, which means that the expected relative distance of Z_2 and $\mathrm{C}_0(x)$ is at most 3δ. Thus, the current case occurs with probability at most 1/6.

Case 3. Z_2 is 18δ-close to $\mathrm{C}_0(x)$ and $\sigma \neq x_i$. Then, on one hand, (Z_1, Z_2) is 1/2-

far from $(x_i^m, \mathrm{C}_0(x))$, because $Z_2 = \sigma^t$. On the other hand, Z_2 is $(\delta_0 - 18\delta)$-far from any other codeword of C_0, because Z_2 is 18δ-close to $\mathrm{C}_0(x)$ and the codewords of C_0 are δ_0-far from one another. Thus, (Z_1, Z_2) is $(\delta_0 - 18\delta)/2$-far from any string of the form $(y_i^m, \mathrm{C}_0(y))$. Using $\delta_{\mathrm{pcpp}} \leq (\delta_0 - 18\delta)/2$, we conclude that the PCPP verifier accepts (Z_1, Z_2) with probability at most 1/6. It follows that, in the current case, the decoder outputs $\neg x_i$ with probability at most 1/6.

Thus, in total, the decoder outputs $\neg x_i$ with probability at most $1/6 + 1/6 = 1/3$. □

Improving the rate. The reason that our code has quadratic length codewords (i.e., $n = \Omega(k^2)$) is that we augmented a standard code with proofs regarding the relation of the standard codeword to the value of *each* information bit. Thus, we had k proofs, each relating to a statement of length $\Omega(k)$. Now, consider the following improvement: Partition the message into $\sqrt{k}$ blocks, each of length $\sqrt{k}$. Encode the original message, as well as each of the smaller blocks, via a good error-correcting code. Let w be the encoding of the entire message, and let w_i $(i = 1, \ldots, \sqrt{k})$ be the encodings of the blocks. For every $i = 1, \ldots, \sqrt{k}$, append a PCPP for the claim "w_i is the encoding of the ith block of a message encoded by w." In addition, for each message bit $x_{(i-1)\sqrt{k}+j}$ residing in block i, append a PCPP of the statement "$x_{(i-1)\sqrt{k}+j}$ is the jth bit of the $\sqrt{k}$bit long string encoded in w_i." The total encoding length has decreased, because we have $\sqrt{k}$ proofs of statements of length $O(k)$ and k proofs of statements of length $O(\sqrt{k})$, leading to a total length that is almost linear in $k^{3/2}$.

In general, for any constant ℓ, we consider ℓ successively finer partitions of the message into blocks, where the $(i+1)$st partition is obtained by breaking each block of the previous partition into $k^{1/\ell}$ equally sized pieces. Thus, the ith partition uses $k^{i/\ell}$ blocks, each of length $k^{1-(i/\ell)}$. Encoding is done by providing, for each $i = 0, 1, \ldots, \ell$, encodings of each of the blocks in the ith partition by a good error-correcting code. Thus, for $i = 0$ we provide the encoding of the entire messages, whereas for $i = \ell$ we provide an "encoding" of individual bits. Each of these $\ell + 1$ levels of encodings will be assigned equal weight (via repetitions) in the new codeword. In addition, the new codeword will contain PCPPs that assert the consistency of "directly related" blocks (i.e., blocks of consecutive levels that contain one another). That is, for every $i = 1, \ldots, \ell$ and $j \in [k^{i/\ell}]$, we place a proof that the encoding of the jth block in the ith level is consistent with the encoding of the $\lceil j/k^{1/\ell} \rceil$th block in the $(i-1)$st level. The ith such sequence of proofs contains $k^{i/\ell}$ proofs, where each such proof refers to statements of length $O(k^{1-(i/\ell)} + k^{1-((i-1)/\ell)}) = O(k^{1-((i-1)/\ell)})$, which yields a total proof length that is upper-bounded by $k^{i/\ell} \cdot (k^{1-((i-1)/\ell)})^{1+o(1)} = k^{1+(1/\ell)+o(1)}$. Each of these sequences will be assigned equal weight in the new codeword, and the total weight of all the encodings will equal the total weight of all proofs. The new decoder will just check the consistency of the ℓ relevant proofs and act accordingly. We stress that, as before, the proofs in use are PCPPs. In the current context these proofs refer to two input oracles of vastly different lengths, and so the bit positions of the shorter input oracle are given higher "weight" (by repetition) such that both input oracles are assigned the same weight.[20]

CONSTRUCTION 4.14. *Let* C_0 *be a code of minimal relative distance* δ_0*, constant rate, and nearly linear-sized encoding circuits. For simplicity, assume that a single bit is encoded by repetitions; that is,* $\mathrm{C}_0(\sigma) = \sigma^{O(1)}$ *for* $\sigma \in \{0,1\}$*. Let*

[20] Indeed, this was also done in the simpler code analyzed in Proposition 4.13.

V be a PCPP of membership in $S_0 = \{\mathrm{C}_0(x) : x \in \{0,1\}^*\}$ having almost linear proof length, query complexity $O(1/\delta_{\mathrm{pcpp}})$, and soundness error $1/9$, for proximity parameter δ_{pcpp}. Furthermore, V's queries to both its input oracle and proof oracle are distributed almost uniformly.[21] *For a fixed parameter $\ell \in \mathbb{N}$, let $b \triangleq k^{1/\ell}$. For $x \in \{0,1\}^k$, we consider ℓ different partitions of x, such that the jth partition denoted $(x_{j,1}, \ldots, x_{j,b^j})$, where $x_{j,j'} = x_{(j'-1)\cdot b^{\ell-j}+1} \cdots x_{j'\cdot b^{\ell-j}}$. We define $\mathrm{C}_j(x) \triangleq (\mathrm{C}_0(x_{j,1}), \mathrm{C}_0(x_{j,2}), \ldots, \mathrm{C}_0(x_{j,b^j}))$, and $\pi_j(x) = (\pi_{j,1}(x), \ldots, \pi_{j,b^j}(x))$, where $p_{j,j'}(x)$ is a PCPP proof oracle that asserts the consistency of j'th block of $\mathrm{C}_j(x)$ and the $\lceil j'/b \rceil$th block of $\mathrm{C}_{j-1}(x)$. That is, $p_{j,j'}(x)$ refers to an input oracle of the form (z_1, z_2), where $|z_1| = |z_2| = O(b^{\ell-j+1})$, and asserts the existence of x such that $z_1 = \mathrm{C}_0(x_{j,j'})^b$ and $z_2 = \mathrm{C}_0(x_{j-1,\lceil j'/b \rceil})$. We consider the code*

$$\mathrm{C}(x) \triangleq (\mathrm{C}_0(x)^{t_0}, \mathrm{C}_1(x)^{t_0}, \ldots, \mathrm{C}_\ell(x)^{t_0}, \pi_1^{t_1}, \ldots, \pi_\ell^{t_\ell}),$$

where the t_j's are selected such that each of the $2\ell+1$ parts of $\mathrm{C}(x)$ has the same length. The decoder, denoted D, operates as follows on input $i \in [k]$ and oracle access to $w = (w_0, w_1, \ldots, w_\ell, v_1, \ldots, v_\ell)$, where $|w_0| = |w_j| = |v_j|$ for all j:

- *D selects uniformly $r_0, r_1, \ldots, r_\ell \in [t_0]$, and $(r'_1, r'_2, \ldots, r'_\ell) \in [t_1] \times [t_2] \times \cdots \times [t_\ell]$.*
- *For $j = 1, \ldots, \ell$, D invokes the PCPP verifier, providing it with access to an input oracle (z_1, z_2) and a proof oracle π that are defined as follows:*
  - *$z_1 = u^b$, where u is the $((r_j - 1) \cdot b^j + \lceil i/b^{\ell-j} \rceil)$th block of w_j.*
  - *z_2 is the $((r_{j-1} - 1) \cdot b^{j-1} + \lceil i/b^{\ell-j+1} \rceil)$th block of w_{j-1}.*
  - *π is the $((r'_j - 1) \cdot b^j + \lceil i/b^{\ell-j} \rceil)$th block of v_j.*

  The PCPP verifier is invoked with proximity parameter $\delta_{\mathrm{pcpp}} = 13\ell\delta > 0$, where $\delta \le \delta_0/81\ell$ is the proximity parameter sought for the decoder.
- *If the PCPP verifier rejects, in any of the aforementioned ℓ invocations, then the decoder outputs a special* (failure) *symbol. Otherwise, the decoder outputs a random value in the $((r_\ell - 1) \cdot k + i)$th block of w_ℓ* (which is supposedly a repetition code of x_i).
- *D satisfies the average smoothness property when we issue some dummy queries that are uniformly distributed in adequate parts of w that are queried less by the above. (In other words, suppose that V makes q_1 (resp., q_2) queries to the first (resp., second) part of its input oracle and q' queries to its proof oracle. Then, w_0 is accessed q_2 times, w_ℓ is accessed q_1 times, each other w_j is accessed $q_1 + q_2$ times, and each v_j is accessed q' times. Thus, we may add dummy queries to make each part accessed $\max(q_1 + q_2, q')$ times, which means increasing the number of queries by a factor of at most $(2\ell+1)/(\ell-1)$ assuming $\ell \ge 2$.)*

Using an adequate PCPP, it holds that $|\mathrm{C}(x)| = \ell \cdot (|x|^{1+(1/\ell)})^{1+o(1)} < |x|^{1+\varepsilon}$ for $\varepsilon = 2/\ell$. The query complexity of D is $O(\ell) \cdot O(1/\delta_{\mathrm{pcpp}}) = O(\ell^2)$. The proof of Proposition 4.13 can be extended, obtaining the following.

PROPOSITION 4.15. *The code and decoder of Construction* 4.14 *satisfy conditions* 1 *and* 2 *of Definition* 4.5 *with respect to proximity parameter $\delta \le \delta_0/81\ell$. Furthermore, this decoder satisfies the average smoothness property.*

Using Lemma 4.10, Theorem 1.5 follows.

[21] Recall that all these conditions hold for the PCPP of Theorem 3.3, where almost uniformly distributed queries to the proof oracle are obtained by a modification analogous to the proof of Lemma 4.9.

Proof. Again, condition 1 as well as the average smoothness property are obvious from the construction, and thus we focus on establishing condition 2. Thus, we fix an arbitrary $i \in [k]$ and follow the outline of the proof of Proposition 4.13.

We consider an oracle $(w_0, w_1, \ldots, w_\ell, \pi_1, \ldots, \pi_\ell)$ that is δ-close to an encoding of $x \in \{0,1\}^k$, where each w_j is assumed to consist of encodings of the $k^{j/\ell}$ (nonoverlapping) $k^{1-(j/\ell)}$-bit long blocks of x, and π_i consists of the corresponding proofs of consistency. It follows that each w_j is $(2\ell+1)\cdot\delta$-close to $\mathrm{C}_j(x)^{t_0}$. Let Z_j denote the block of w_j that was selected and accessed by D. Thus, the expected relative distance of Z_0 from $\mathrm{C}_0(x)$ is at most $(2\ell+1)\cdot\delta$, but we do not know the same about the other Z_j's because their choice depends on i (or rather on $\lceil i/b^{\ell-j}\rceil$). Assuming, without loss of generality, that $\delta_0 < 1/3$ (and $\ell \geq 1$), we consider three cases as follows.

Case 1. Z_ℓ is 1/9-close to $\mathrm{C}_0(x_i)$. In this case, D outputs either $\perp$ or a uniformly selected bit in Z_ℓ, and so D outputs $\neg x_i$ with probability at most 1/9.

Using $\delta \leq \delta_0/81\ell$ and $\delta_0 < 1/3$, it follows that $27\ell\delta < 1/9$. Thus, if Case 1 does not hold, then Z_ℓ is $27\ell\delta$-far from $\mathrm{C}_0(x_i)$.

Case 2. Z_0 is $27\ell\delta$-far from $\mathrm{C}_0(x)$. This case may occur with probability at most 1/9, because $\mathrm{E}[\overline{\Delta}(Z_0, \mathrm{C}_0(x))] \leq 3\ell\delta \cdot |\mathrm{C}_0(x)|$.

Note that if both Cases 1 and 2 do not hold, then Z_0 is $27\ell\delta$-close to $\mathrm{C}_0(x)$ but Z_ℓ is $27\ell\delta$-far from $\mathrm{C}_0(x_i)$. Also note that $x = x_{0,1}$ and $x_i = x_{\ell,i}$.

Case 3. For some $j \in [\ell]$, it holds that Z_{j-1} is $27\ell\delta$-close to $\mathrm{C}_{j-1}(x_{j-1,\lceil i/b^{\ell-j+1}\rceil})$ but Z_j is $27\ell\delta$-far from $\mathrm{C}_j(x_{j,\lceil i/b^{\ell-j}\rceil})$. In this case, the pair (Z_j^b, Z_{j-1}) is $27\ell\delta/2$-far from the consistent pair $(\mathrm{C}_j(x_{j,\lceil i/b^{\ell-j}\rceil}), \mathrm{C}_{j-1}(x_{j-1,\lceil i/b^{\ell-j+1}\rceil}))$ and is $(\delta_0 - 27\ell\delta)/2$-far from any other consistent pair. Using $\delta_{\mathrm{pcpp}} = 13\ell\delta < \min(27\ell\delta/2, (\delta_0 - 27\ell\delta)/2)$, which holds because $\delta \leq \delta_0/81\ell$, it follows that in the current case the PCPP verifier accepts (and the decoder does not output $\perp$) with probability at most 1/9.

Thus, in total, the decoder outputs $\neg x_i$ with probability at most 1/3. □

Conclusion (restatement of Theorem 1.5). *For every constant $\varepsilon > 0$, there exists a code $\mathrm{C} : \{0,1\}^k \to \{0,1\}^n$, where $n = k^{1+\varepsilon}$, that is relaxed locally decodable under Definition* 4.11. *The query complexity of the corresponding decoder is $O(1/\varepsilon^2)$ and the proximity parameter is $\varepsilon/O(1)$.*

Open problem. We wonder whether one can obtain a relaxed LDC that can be decoded using q queries while having length $n = o(k^{1+/(q-1)})$. The existence of such a relaxed LDC will imply that our relaxation (i.e., relaxed LDC) is actually strict, because such codes will beat the lower-bound currently known for LDC (cf. [KT00]). Alternatively, it may be possible to improve the lower-bound for the (q-query) LDC to $n > k^{1+\sqrt{c/q}}$ for any constant c and every sufficiently large constant q (where, as usual, k is a parameter, whereas q is a fixed constant). In fact, some conjecture that n must be superpolynomial in k for any constant q.

4.3. Linearity of the codes. We note that the codes presented above (establishing both Theorems 1.4 and 1.5) are actually $\mathbb{F}_2$-linear codes whenever the base code C_0 is also $\mathbb{F}_2$-linear. Proving this assertion reduces to proving that the PCPPs used (in the aforementioned constructions) have proof oracles in which each bit is a linear function of the bits to which the proof refers. The main part of the latter task is undertaken in section 8.4, where we show the main construct (i.e., the PCPPs stated in Theorems 3.1 and 3.2) when applied to a linear circuit yields an $\mathbb{F}_2$-linear transformation of assignments (satisfying the circuit) to proof oracles (accepted by the verifier). In addition, we also need to show that the construction underlying the proof of Theorem 3.3 satisfies this property. This is done next, and consequently we get the following.

PROPOSITION 4.16. *If C is a linear circuit (see Definition* 8.13*), then there is a linear transformation T mapping satisfying assignments w of C to proof oracles $T(w)$ such that the PCPP verifier of Theorem* 3.3 *will, on input C, accept oracle $(w, T(w))$ with probability* 1*.*

Proof sketch. In section 8.4, we establish a corresponding result for the main construct (i.e., Proposition 8.14 refers to the linearity of the construction used in the proof of Theorem 3.1, which in turn underlies Theorems 3.1 and 3.2). Here we show that linearity is preserved in composition as well as by the most inner (i.e., bottom) verifier.

In each composition step, we append the proof oracle with new (inner) PCPPs per each test of the (outer) verifier. Since all these tests are linear, we can apply Proposition 8.14 and infer that the new appended information is a linear transformation of the input oracle and the outer proof oracle (where, by induction, the latter is a linear transformation of the input).

At the bottom level of composition we apply a Hadamard-based PCP (Appendix A). The encoding defined there is not $\mathbb{F}_2$-linear (rather it is quadratic), but this was necessary for dealing with nonlinear gates. It can be verified that for a linear circuit, one can perform all necessary tests of Appendix A with the Hadamard encoding of the input. Thus, we conclude that this final phase of the encoding is also linear, and this completes the proof of Proposition 4.16. ∎

Part II. The main construct: A short, robust PCPP.

5. Overview of our main construct. Throughout this section, n denotes the length of the explicit input given to the PCPP verifier, which in the case of CKTVAL is defined as the size of the circuit (given as explicit input). As stated in the introduction, our main results rely on the following highly efficient robust PCPP.

THEOREM 3.1 (main construct restated). *There exists a universal constant c such that for all $n, m \in \mathbb{Z}^+$, $0 < \delta, \gamma < 1/2$ satisfying $n^{1/m} \geq m^{cm}/(\gamma\delta)^3$ and $\delta \leq \gamma/c$,* CKTVAL *has a robust PCPP (for circuits of size n) with the following parameters:*

- *randomness $\left(1 - \frac{1}{m}\right) \log n + O(m \log m) + O(\log \log n) + O(\log(1/\delta))$;*
- *decision complexity $n^{1/m} \cdot \mathrm{poly}(\log n, 1/\delta)$, which also upper-bounds the query complexity;*[22]
- *perfect completeness; and*
- *for proximity parameter δ, the verifier has robust-soundness error $\Omega(\gamma)$ with robustness parameter $(1 - \gamma)\delta$.*

A (simplified) variant of Theorem 3.1 also yields the ALMSS-type robust PCPP (of Theorem 3.2). Following is an overview of the proof of Theorem 3.1; the actual proof is given in the subsequent three sections.

Theorem 3.1 is proved by modifying a construction that establishes Theorem 1.1. We follow [HS00] and modify their construction. (An alternative approach would be to start from [PS94], but that construction does not seem amenable to achieving robust soundness.) The construction of [HS00] may be abstracted as follows: To verify the satisfiability of a circuit of size n, a verifier expects oracles $\mathrm{A}_i : \mathbb{F}^m \to \mathbb{F}$, $i \in \{1 \ldots, t = \mathrm{poly} \log n\}$, where $\mathbb{F}$ is a field and m is a parameter such that $|\mathbb{F}^m| \approx m^m \cdot n$. The verifier then needs to test that (1) each of the A_i's is close to an m-variate polynomial of *low degree* and (2) the polynomials satisfy some *consistency*

[22] In fact, we will upper-bound the query complexity by $q = n^{1/m} \cdot \mathrm{poly}(\log n, 1/\delta)$ and show that the verifier's decision can be implemented by a circuit of size $\widetilde{O}(q)$, which can also be bounded by $n^{1/m} \cdot \mathrm{poly}(\log n, 1/\delta)$ with a slightly larger unspecified polynomial.

properties which verify that A_i is locally consistent with A_{i-1}.[23] (These consistency checks include tests which depend on the input circuit and verify that A_i's actually encode a satisfying assignment to the circuit.)

We work within this framework—namely, our verifier will also try to access oracles for the A_i's and test low degreeness and consistency. Our key modification to this construction is a randomness reduction in the low-degree test obtained by using the small collection of (small-biased) lines of [BSVW03], while using only the "canonical" representations of these lines (and avoiding any complication that was introduced towards proof composition). In particular, unlike in [HS00, GS02, BSVW03], we cannot afford to pack the polynomials $A_1, \ldots, A_t$ into a single polynomial (by using an auxiliary variable that blows up the proof length by a factor of the size of the field in use). Instead, we just maintain all these t polynomials separately and test them separately to obtain Theorem 1.1. (In the traditional framework of parallelized PCPs, this would give an unaffordable increase in the number of (non-Boolean) queries. However, we will later ameliorate this loss by a bundling technique that will yield robust soundness.)

The resulting PCP is converted into a PCPP by comparing the input oracle (i.e., the supposed satisfying assignment to the circuit) to the proof oracle (which is supposed to include an encoding of the said assignment). That is, we read a random location of the input and the corresponding location of the proof oracle, and test for equality. Actually, these locations of the proof oracle must be accessed via a self-correction mechanism (rather than by merely probing at the desired points of comparison), since they constitute only a small part of the proof oracle (and thus corruptions there may not be detected). This technique was already suggested in [BFLS91].

The most complex and subtle part of the proof of Theorem 3.1 is establishing the robust-soundness property. We sketch how we do this below, first dealing with the low-degree test and the consistency tests separately, and then showing how to reconcile the two "different" fixes.

Low-degree tests of $A_1, \ldots, A_t$. Selecting a random line $\ell : \mathbb{F} \to \mathbb{F}^m$ (from the aforementioned sample space), we can check that (for each i) the restriction of A_i to the line ℓ (i.e., the function $f_i(j) \triangleq A_i(\ell(j))$) is a low-degree (univariate) polynomial. Each of these tests is *individually robust*; that is, if A_i is far from being a low-degree polynomial, then with high probability the restriction of A_i to a random line ℓ (in the sample space) is far from being a low-degree polynomial. The problem is that the conjunction of the t tests is not sufficiently robust; that is, if one of the A_i's is δ-far from being a low-degree polynomial, then it is guaranteed only that the sequence of t restrictions (i.e., the sequence of the f_i's) is (δ/t)-far from being a sequence of t low-degree (univariate) polynomials. Thus, robustness decreases by a factor of t, which we cannot afford for nonconstant t.

Our solution is to observe that we can "bundle" the t functions together into a function $\overline{A} : \mathbb{F}^m \to \mathbb{F}^t$ such that if one of the A_i's is far from being a low-degree polynomial, then the restriction of $\overline{A}$ to a random line will be far from being a bundling of t low-degree univariate polynomials. Specifically, for every $x \in \mathbb{F}^m$, define $\overline{A}(x) \triangleq (A_1(x), \ldots, A_t(x))$. To test that $\overline{A}$ is a bundling of low-degree polynomials, select a

[23] Strictly speaking, the consistency checks are a little more complicated, with the functions really being indexed by two subscripts, and consistency tests being between $A_{i,j}$ and $A_{i,j-1}$, as well as between $A_{i,0}$ and $A_{i+1,0}$. However, these differences don't alter our task significantly—we ignore them in this section to simplify our notation.

random line ℓ (as above), and check that $\overline{f}_\ell(j) = \overline{\mathrm{A}}(\ell(j))$ is a bundling of low-degree univariate polynomials. Thus, we establish *robustness at the bundle level*; that is, if one of the A_i's is far from being low degree then, with high probability, one must modify $\overline{f}_\ell$ on a constant fraction of values in order to make the test accept. The point is that this robustness refers to the Hamming distance over the alphabet $\mathbb{F}^t$, rather than alphabet $\mathbb{F}$ as before. We can afford this increase in alphabet size, as we later encode the values of $\overline{\mathrm{A}}$ using an error-correcting code in order to derive *robustness at the bit level*.

We wish to highlight a key point that makes the above approach work: when we look at the values of $\overline{\mathrm{A}}$ restricted to a random line, we get the values of the individual A_i's restricted to a random line, which is exactly what a low-degree test of each A_i needs. This fact is not very surprising, given that we are subjecting all A_i's to the same test. *But what happens when we need to make two different types of tests?* This question is not academic and does come up in the consistency tests.

Consistency tests. To bundle the t consistency tests between A_i and A_{i+1} we need to look into the structure of these tests. We note that for every i, a random test essentially refers to the values of A_i and A_{i+1} on (random) ith axis-parallel lines. That is, for every i, and a random $x' = (x_1, \ldots, x_{i-1}) \in \mathbb{F}^{i-1}$ and $x'' = (x_{i+1}, \ldots, x_m) \in \mathbb{F}^{m-i}$, we need to check some relation between $\mathrm{A}_i(x', \cdot, x'')$ and $\mathrm{A}_{i+1}(x', \cdot, x'')$.[24] Clearly, querying $\overline{\mathrm{A}}$ as above on the ith axis-parallel line, we can obtain the relevant values from $\overline{\mathrm{A}}(x', \cdot, x'')$, but this works only for one specific value of i, and other values of i will require us to make other queries. The end result is that we gain nothing from the bundling (i.e., from $\overline{\mathrm{A}}$) over using the individual A_i's, which yields a factor of t loss in the robustness.[25] Fortunately, a different bundling works in this case.

Consider $\overline{\mathrm{A}}'$ such that $\overline{\mathrm{A}}'(x) \triangleq (\mathrm{A}_1(x), \mathrm{A}_2(S(x)), \ldots, \mathrm{A}_t(S^{t-1}(x)))$, for every $x \in \mathbb{F}^m$, where S denotes a (right) cyclic-shift (i.e., $S(x_1, \ldots, x_m) = (x_m, x_1, \ldots, x_{m-1})$ and $S^i(x_1, \ldots, x_m) = (x_{m-(i-1)}, \ldots, x_m, x_1, x_2, \ldots, x_{m-i})$). Now, if we ask for the value of $\overline{\mathrm{A}}'$ on the first and last axis-parallel lines (i.e., on $(\cdot, x_2, \ldots, x_m)$ and $(x_2, \ldots, x_m, \cdot) = S^{-1}(\cdot, x_2, \ldots, x_m)$), then we get all we need for all the m tests. Specifically, for every i, the ith component in the bundled function $\overline{\mathrm{A}}'(\cdot, x_2, \ldots, x_m)$ is $\mathrm{A}_i(S^{i-1}(\cdot, x_2, \ldots, x_m)) = \mathrm{A}_i(x_{m-i+2}, \ldots, x_m, \cdot, x_2, \ldots, x_{m-i+1})$, whereas the $(i+1)$st component in $\overline{\mathrm{A}}'(S^{-1}(\cdot, x_2, \ldots, x_m))$ is $\mathrm{A}_{i+1}(S^i(S^{-1}(\cdot, x_2 \ldots, x_m))) = \mathrm{A}_{i+1}(x_{m-i+2}, \ldots, x_m, \cdot, x_2, \ldots, x_{m-i+1})$. Thus, we need only query two bundles (rather than t), and robustness drops only by a constant factor.

Reconciling the two bundlings. But what happens with the low-degree tests that we need to do (which were "served" nicely by the original bundling $\overline{\mathrm{A}}$)? Note that we cannot use both $\overline{\mathrm{A}}$ and $\overline{\mathrm{A}}'$, because this will requires testing consistency between them, which will introduce new problems as well as a cost in randomness that we cannot afford. Fortunately, the new bundling (i.e., $\overline{\mathrm{A}}'$), designed to serve the axis-parallel line comparisons, can also serve the low-degree tests. Indeed, the various A_i's will not be inspected on the same lines, but this does not matter, because the property of being a low-degree polynomial is preserved when "shifted" (under S).

[24] Again, this is an oversimplification but suffices to convey the main idea of our solution.

[25] It turns out that for constant m (e.g., $m = 2$) this does not pose a problem. However, a constant m would suffice only for proving a slightly weaker version of Theorem 1.2 (where $o(\log \log n)$ is replaced by $\log \log n$) and not for proving Theorem 1.3, which requires setting and $m = \log^\varepsilon n$ for constant $\varepsilon > 0$.

Tightening the gap between robustness and proximity. The above description suffices for deriving a weaker version of Theorem 3.1 in which the robustness is only, say, $\delta/3$ rather than $(1-\gamma)\delta$ for a parameter γ that may be set as low as $1/\mathrm{poly}(\log n)$. Such a weaker result yields a weaker version of Theorem 3.3 in which the query complexity is exponentially larger (e.g., for proof length $\exp(o(\log\log n)^2)\cdot n$, we would have obtained query complexity $\exp(o(\log\log n)) = \log^{o(1)} n$ rather than $o(\log\log n)$); see the comment at the end of section 3.1. To obtain the stronger bound on the robustness parameter, we take a closer look at the conjunction of the standard PCP test and the proximity test. The PCP test can be shown to have constant robustness $c > 0$, whereas the proximity test can be shown to have robustness $\delta' \triangleq (1-\gamma)\delta$. When combining the two tests, we obtain robustness equal to $\min(\alpha c, (1-\alpha)\delta')$, where α is the relative length of queries used in the PCP test (as a fraction of the total number of queries). A natural choice, which yields the weaker result, is to weight the queries (or replicate the smaller part) so that $\alpha = 1/2$. (This yields robustness of approximately $\min(c,\delta')/2$.) In order to obtain the stronger bound, we assign weights such that $\alpha = \gamma$, and obtain robustness $\min(\gamma c, (1-\gamma)\delta') > \min(\Omega(\gamma), (1-2\gamma)\delta)$, which simplifies to $(1-2\gamma)\delta$ for $\delta < \gamma/O(1)$. (The above description avoids the fact that the PCP test has constant soundness error, but the soundness error can be decreased to γ by using sequential repetitions while paying a minor cost in randomness and *while approximately preserving the robustness*. We comment that the proximity test, as is, has soundness error γ.)

6. A randomness-efficient PCP. In this section, we present a "vanilla" version (Theorem 6.1) of Theorem 3.1. More specifically, we construct a regular PCP for CktSAT (i.e., a robust PCPP without either the robustness or proximity properties). We favor this construction over earlier PCP constructions in the fact that it is very efficient in randomness. As mentioned earlier, this theorem suffices to prove Theorem 1.1.

THEOREM 6.1. *There exists a universal constant* $0 < \varepsilon < 1$ *such that the following holds. Suppose* $m \in \mathbb{Z}^+$ *satisfies* $m \le \log n/\log\log n$. *Then there exists a PCP for* CktSAT *(for circuits of size* n*) with the following parameters:*

- *randomness* $(1-\frac{1}{m})\log n + O(m\log m) + O(\log\log n)$,
- *query complexity* $q = O(m^2 n^{1/m}\log^2 n)$ *and decision complexity* $\widetilde{O}(q)$,
- *perfect completeness,*
- *and soundness error* $1-\varepsilon$.

The construction of the PCP for CktSAT proceeds in three steps. First, we transform the input circuit φ into a well-structured circuit φ' along the lines of Polishchuk and Spielman [PS94, Spi95] (section 6.1). φ' is only slightly larger than φ, but has an algebraic structure that will be crucial to our verification process. Any legal assignment to the gates of φ (i.e., one that preserves the functionality of the gates of φ) can be transformed into a legal assignment to φ'. The important property of φ' is the following: If we encode an assignment to the gates of φ' using a specific sequence of Reed–Muller codewords (i.e., low-degree polynomials), then the legality of the assignment can be locally verified (by reading a small random portion of the encoding). The encoding via low-degree polynomials (and resulting local tests) is as in Harsha and Sudan [HS00] and is described in section 6.2. Thus, our PCP verifier will essentially test (i) that the encoding of the purported satisfying assignment to φ' is formed of low-degree polynomials (this part will be done using the randomness-efficient low-degree test of Ben Sasson et al. [BSVW03]); and (ii) that the assignment is legal. Section 6.3 describes the construction of the PCP verifier and section 6.4

analyzes its properties. Most of the above results are implicit in the literature, but carefully abstracting the results and putting them together helps us in significantly reducing the randomness of the PCP verifier.

6.1. Well-structured Boolean circuits. The main problem with designing a randomness-efficient PCP verifier directly for CktSAT is that we need to encode the assignment to all gates of the input circuit using certain Reed–Muller based codes, in such a way that will allow us to locally verify the legality of all gates of the circuit, using only the encoded assignment. In order to do this, we require the circuit to have a well-behaved structure (amenable to our specific encoding and verification demands). Of course, an arbitrary circuit does not necessarily have this structure, but we are lucky to have the technology to overcome this. More to the point, we can restructure any circuit into a well-behaved circuit that will suit our needs. The natural encoding (used, e.g., in the Hadamard-based PCP, Appendix A) incurs a quadratic blow-up in length. To get over this problem, Polishchuk and Spielman [PS94, Spi95] introduced a different, more efficient restructuring process that embeds the input circuit into well-structured graphs known as de Bruijn graphs. Indeed, the blow-up in circuit size using these circuits is merely by a logarithmic multiplicative factor, and their usefulness for the local verification of legal assignments will become evident later (in section 6.2). As in Polishchuk and Spielman [PS94, Spi95], we embed the input circuit into wrapped de Bruijn graphs (see Definition 6.2). We use a slightly different definition of de Bruijn graphs, more convenient for our purposes, than that used in [PS94, Spi95]. However, it can be easily checked that these two definitions yield isomorphic graphs. The main advantage with the de Bruijn graphs is that the neighborhood relations can be expressed very easily using simple bit-operations like cyclic-shifts and bit-flips. In [PS94, Spi95] the vertex set of these graphs is *identified* with a vector space. We instead work with a strict embedding of these graphs in a vector space, where the vertices are a *strict* subset of the vector space. The benefit of both approaches is that the neighborhood functions can be expressed as an affine functions (see section 6.2 for more details). The reason for our approach will be explained at the end of section 6.2.

Definition 6.2. *The wrapped de Bruijn graph $\mathcal{G}_{N,l}$ is a directed graph with l layers, each with 2^N nodes which are represented by N-bit strings. The layers are numbered $0, 1, \ldots, l-1$. The node represented by $v = (b_0, \ldots, b_{i^*}, \ldots, b_{N-1})$ in layer i has edges pointing to the nodes represented by $\Gamma_{i,0}(v) = (b_0, \ldots, b_{i^*}, \ldots, b_{N-1})$ and $\Gamma_{i,1}(v) = (b_0, \ldots, b_{i^*} \oplus 1, \ldots, b_{N-1})$ in layer $(i+1)$ modulo l, where i^* is i modulo N and $a \oplus b$ denotes the sum of a and b modulo 2. See Figure 1 for an example.*

We now describe how to embed a circuit into a wrapped de Bruijn graph (see Figure 2 for a simple example). Given a circuit C with n gates (including both input and output gates), we associate with it a wrapped de Bruijn graph $\mathcal{G}_{N,l}$, where $N = \log n$ and $l = 5N = 5 \log n$. We then associate the nodes in layer 0 with the gates of the circuit. Now, we wish to map each wire in the circuit to a path in $\mathcal{G}_{N,l}$ between the corresponding nodes of layer 0. Standard packet-routing techniques (see [Lei92]) can be used to show that if the number of layers l is at least $5N$, then such a routing can be done with edge-disjoint paths. (Recall that we work with circuits whose fan-in and fan-out are 2.)

Thus, we can find "switches" for each of the nodes in layers $1, \ldots, l-1$ of the graph such that the output of each gate (i.e., node in layer 0) is routed to the input of the gates that require it. Each node has two inputs and two outputs, and thus there is a constant number of switches routing incoming edges to outgoing ones (see Figure 3).

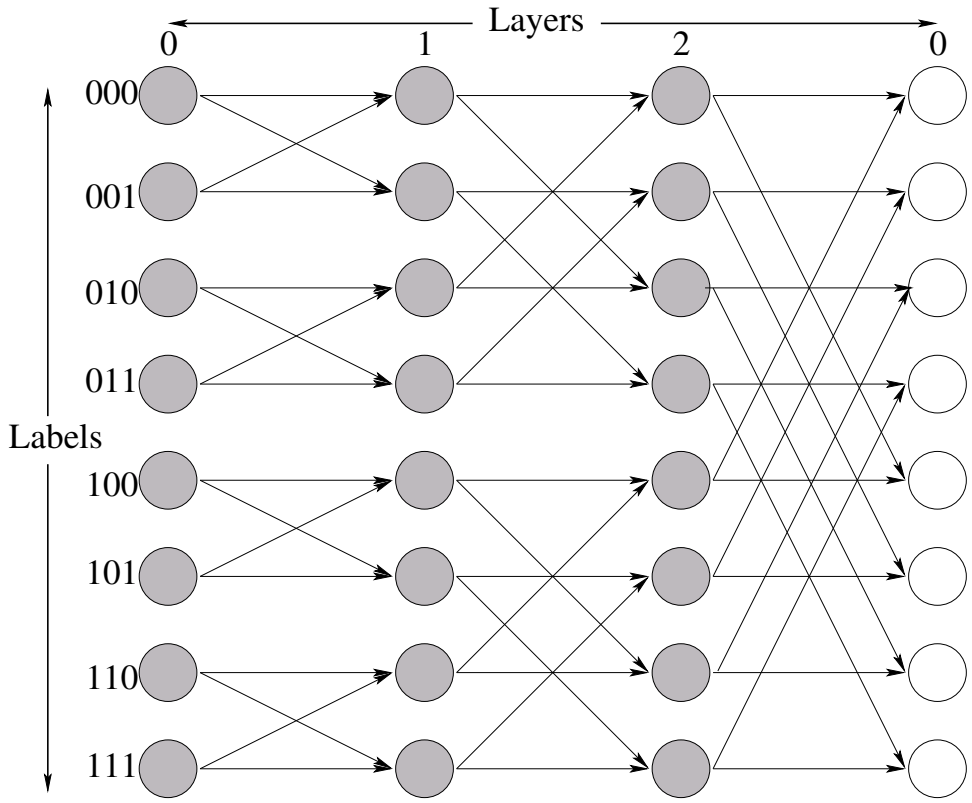


FIG. 1. *The wrapped de Bruijn graph $\mathcal{G}_{3,3}$. Notice the first and last layers are the same.*

For nodes in layer 0, instead of specifying a switch, we specify the functionality of the Boolean gate associated to that node in the circuit (e.g., AND, OR, PARITY, NOT, INPUT, OUTPUT). Actually unary gates (such as NOT and OUTPUT) have two forms (NOT, NOT′, OUTPUT, OUTPUT′) in order to specify which of the two incoming edges in the de Bruijn graph to use.

This specifies the embedding of the input circuit into a well-structured circuit (based on a de Bruijn graph). More precisely, let $\mathcal{C} = \{\text{type of switching actions}\} \cup \{\text{type of Boolean gates}\}$ be the set of allowable gates of the well-structured circuit (see Figure 3). Given a circuit on n gates, we can construct, in polynomial time, a wrapped de Bruijn graph $\mathcal{G}_{N,l}$ (where $N = \log n$ and $l = 5 \log N$) and l functions $T_0, T_1, \ldots, T_{l-1} : \{0,1\}^N \to \mathcal{C}$, where each function T_i is a specification of the gates of layer i (i.e., a specification of the switching action or Boolean functionality).

We now demonstrate how to translate a proof that a circuit is satisfiable into an assignment that satisfies the embedded circuit. A proof that a circuit is satisfiable consists of an assignment of 0's and 1's to the inputs and the gates of the circuit such that each gate's output is consistent with its inputs and the output gate evaluates to 1. The corresponding assignment to the embedded circuit consists of an assignment of 0's and 1's to the edges entering and leaving the nodes of the wrapped de Bruijn graph that is consistent with the functionality of the gates (in layer 0) and the switching actions of the nodes (in the other layers). Since we are assigning values to nodes of the embedded graph (and not their edges), the assignment actually associates a 4-tuple

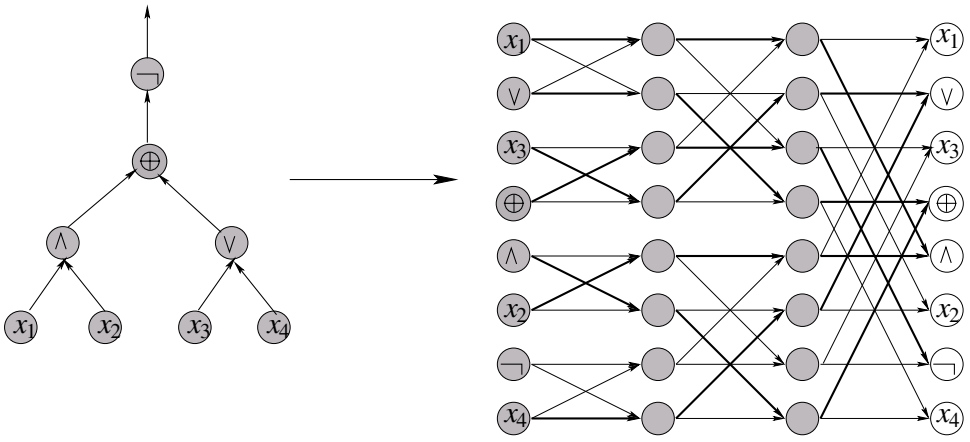


FIG. 2. *Embedding of a circuit into* $\mathcal{G}_{3,3}$. *In this example all paths between nodes at the* 0 *layer are vertex disjoint. For general circuits we merely need edge disjoint paths.*

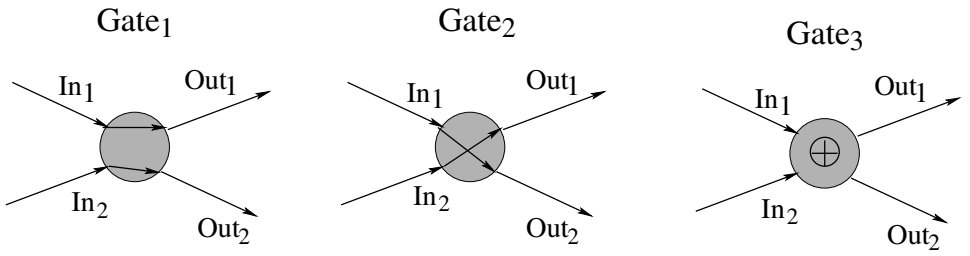


FIG. 3. *Some gates of a well-structured circuit. Gates* 1–2 *are switching gates, and gate* 3 *sits in layer* 0 *and computes the parity (xor) function.*

of 0's and 1's to each of the nodes in the graph indicating the value carried by the four edges incident at that node (two incoming and two outgoing). More formally, the embedded assignment is given by a set of l functions $A_0, A_1, \ldots, A_{l-1}$, where each function $A_i : \{0,1\}^N \to \{0,1\}^4$ specifies the values carried by the four edges incident at that vertex.

We now list the constraints on the embedded circuit that assure us that the only legal assignments are the ones that correspond to legal satisfying assignments of the original circuit, i.e., assignments that correctly propagate along the edges of the circuit, correctly compute the value of every gate, and produce a 1 at the output gate.

DEFINITION 6.3. *The* assignment constraints *for each node of the well-structured circuit require that*

- *the two outgoing values at the node are propagated correctly to the incoming values of its neighbors at the next level;*
- *for nodes at layers* $\neq 0$, *the two outgoing values have the unique values dictated by the incoming values and the switching action;*
- *for non-OUTPUT nodes in layer* 0, *both outgoing values equal the unique value dictated by the gate functionality and the incoming values (the INPUT functionality merely requires that the two outgoing values are equal to each other);*
- *for nodes in layer* 0 *with an OUTPUT functionality, the appropriate incoming*

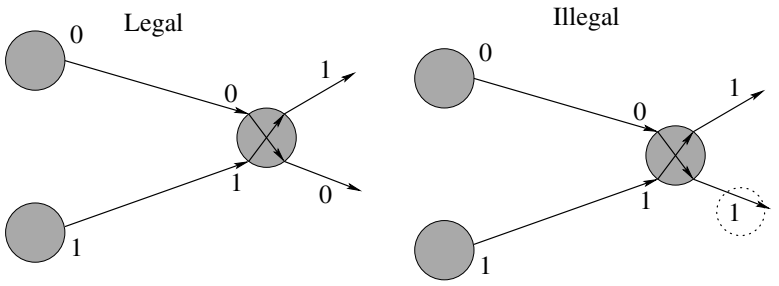


FIG. 4. *Example of legal and illegal assignments. The two vertices on the left are the inputs (at layer $i-1$) to a gate at layer i. Recall that assignments evaluate each incoming and outgoing edge of a gate.*

value equals 1.

Let $\psi : \mathcal{C} \times (\{0,1\}^4)^3 \to \{0,1\}$ be the Boolean function such that $\psi(t, a, a_0, a_1) = 0$ iff a node whose T-gate is t, A-assignment is a, and whose neighbors in the next layer have A-assignments a_0 and a_1, respectively, satisfies the aforementioned assignment constraints. See Figure 4 *for an example of legal and illegal assignments.*

Observe that the definition of ψ is independent of N, the assignments A_i, and gates T_i. By definition, the assignment $A = (A_0, \ldots, A_{l-1})$ is legal for an embedded circuit defined by $T_0, \ldots, T_{l-1}$ iff for every layer i and every node v in layer i,

$$\psi\Big(T_i(v), A_i(v), A_{i+1}\big(\Gamma_{i,0}(v)\big), A_{i+1}\big(\Gamma_{i,1}(v)\big)\Big) = 0.$$

We are now ready to formally define the well-structured circuit satisfiability problem (STRUCTURED-CKTSAT).

DEFINITION 6.4. *The problem* STRUCTURED-CKTSAT *has as its instances $\langle \mathcal{G}_{N,l}, \{T_0, T_1, \ldots, T_{l-1}\}\rangle$, where $\mathcal{G}_{N,l}$ is a wrapped de Bruijn graph with l layers and $T_i : \{0,1\}^N \to \mathcal{C}$ is a specification of the node types of layer i of the graph (T_i's are specified by a table of values).*

$\langle \mathcal{G}_{N,l}, \{T_0, \ldots, T_{l-1}\}\rangle \in$ STRUCTURED-CKTSAT *if there exists a set of assignments $A_0, A_1, \ldots, A_{l-1}$, where $A_i : \{0,1\}^N \to \{0,1\}^4$ is an assignment to the nodes of layer i of $\mathcal{G}_N$ such that for all layers i and all nodes v in layer i,*

$$\psi\Big(T_i(v), A_i(v), A_{i+1}\big(\Gamma_{i,0}(v)\big), A_{i+1}\big(\Gamma_{i,1}(v)\big)\Big) = 0.$$

The above discussion also demonstrates the existence of a reduction from CKTSAT to STRUCTURED-CKTSAT, which does not blow up the length of the target instance by more than a logarithmic multiplicative factor.

PROPOSITION 6.5. *There exists a polynomial-time reduction $\mathcal{R}$ from* CKTSAT *to* STRUCTURED-CKTSAT *such that for any circuit C, it holds that $C \in$* CKTSAT *iff $\mathcal{R}(C) \in$* STRUCTURED-CKTSAT. *Moreover, if C is a circuit of size n, then $\mathcal{R}(C) = \langle \mathcal{G}_{N,l}, \{T_0, \ldots, T_{l-1}\}\rangle$, where $N = \lceil \log n \rceil$ and $l = 5N$.*

Remark 6.6. The above reduction, though known to take polynomial time (via routing techniques), is not known to be of almost linear time.

Remark 6.7. We observe that if C is a satisfiable circuit, then any set of assignments $A_0, \ldots, A_l$ proving that the reduced instance $\mathcal{R}(C) = \langle \mathcal{G}_{N,l}, \{T_0, \ldots, T_{l-1}\}\rangle$ is a YES instance of STRUCTURED-CKTSAT contains within it a satisfying assignment

to the circuit C. Specifically, let I be the set of nodes in layer 0 that have gate functionality INPUT associated with them. Then the assignment A_0 restricted to the set of nodes I (i.e., $A_0|_I$) contains a satisfying assignment. More precisely, the satisfying assignment is obtained by concatenating the third bit (i.e., first outgoing bit) of $A_0|_i \in \{0,1\}^4$ for all $i \in I$. Conversely, every satisfying assignment w to C can be extended to $A_0, \ldots, A_{l-1}$ such that $A_0|_I$ contains w. This is done by computing the values of all gates in the computation of $C(w)$, setting the outgoing bits of A_0 according to these values, and routing them throughout $\mathcal{G}_{N,l}$ according to the switching actions to obtain $A_1, \ldots, A_{l-1}$ and the incoming bits of A_0. This observation will be vital while constructing PCPPs (see section 7).

Remark 6.8. Suppose the input circuit C is a *linear* circuit, in the sense that all gates are INPUT, OUTPUT, or PARITY gates, and the OUTPUT gates test for 0 rather 1 (see Definition 8.13). Then it can be verified that the transformation mapping satisfying assignments w of C to legal assignments $A_0, \ldots, A_{l-1}$ of $\mathcal{R}(C)$ is $\mathbb{F}_2$-linear. The reason is that each gate in the computation of $C(w)$ is an $\mathbb{F}_2$-linear function of w. These remarks will be used in the coding applications, to obtain linear codes (see section 8.4 for more information).

6.2. Arithmetization. In this section, we construct an algebraic version of Structured-CktSAT by arithmetizing it along the lines of Harsha and Sudan [HS00]. The broad overview of the arithmetization is as follows: We embed the nodes in each layer of the wrapped de Bruijn graph $\mathcal{G}_{N,l}$ into a vector space and extend the gate specifications and assignments to low-degree polynomials over this space. Finally, we express the assignment constraints (see Definition 6.3) as a pair of polynomial identities satisfied by these polynomials.

First, we have some notation. Let m be a parameter. Set h such that $h = N/m$, where 2^N is the number of nodes in each layer of the de Bruijn graph. Choose a finite extension field $\mathbb{F}$ of $\mathbb{F}_2$ of size roughly $c_F m^2 2^h = c_F m^2 2^{N/m}$, where c_F is a suitably large constant to be specified later. Specifically, take $\mathbb{F} = \mathbb{F}_2^{\mathcal{U}} = \mathbb{F}_{2^{\mathcal{U}}}$ for $\mathcal{U} = h + 2\log m + \log c_F$. Let $\{e_0, e_1, \ldots, e_{\mathcal{U}-1}\}$ be a basis of $\mathbb{F}$ over $\mathbb{F}_2$. Set H to be a subspace of $\mathbb{F}_2^{\mathcal{U}}$ (and hence a subset of $\mathbb{F}$) spanned by $\{e_0, \ldots, e_{h-1}\}$. Note that H^m is a subset of the space $\mathbb{F}^m$. Furthermore, $|H^m| = 2^N$. Hence, we can embed each layer of the graph $\mathcal{G}_{N,l}$ in $\mathbb{F}^m$ by identifying the node $v = (b_0, \ldots, b_{N-1}) \in \{0,1\}^N$ with the element $(b_0 e_0 + \cdots + b_{h-1} e_{h-1}, b_h e_0 + \cdots + b_{2h-1} e_{h-1}, \ldots, b_{(m-1)h} e_0 + \cdots + b_{mh-1} e_{h-1})$ of H^m. Henceforth, we use both representations (N-bit string and element of H^m) interchangeably. The representation will be clear from the context.

Any assignment $S : H^m \to \mathbb{F}$ can be interpolated to obtain a polynomial $\tilde{S} : \mathbb{F}^m \to \mathbb{F}$ of degree at most $|H|$ in each variable (and hence a total degree of at most $m|H|$) such that $\tilde{S}|_{H^m} = S$ (i.e., the restriction of $\tilde{S}$ to H^m coincides with the function S). Conversely, any polynomial $\tilde{S} : \mathbb{F}^m \to \mathbb{F}$ can be interpreted as an assignment from H^m to $\mathbb{F}$ by considering the function restricted to the subdomain H^m.

Recall that $\mathcal{C}$ and $\{0,1\}^4$ are the set of allowable gates and assignments given by the gate functions T_i and assignments A_i in the Structured-CktSAT problem. We identify $\mathcal{C}$ with a fixed subset of $\mathbb{F}$ and identify $\{0,1\}^4$ with the set of elements spanned by $\{e_0, e_1, e_2, e_3\}$ over $\mathbb{F}_2$. With this identification, we can view the assignments A_i and gates T_i as functions $A_i : H^m \to \mathbb{F}$ and $T_i : H^m \to \mathbb{F}$, respectively. Furthermore, we can interpolate these functions, as mentioned above, to obtain polynomials $\tilde{A}_i : \mathbb{F}^m \to \mathbb{F}$ and $\tilde{T}_i : \mathbb{F}^m \to \mathbb{F}$ of degree at most $m|H|$ over $\mathbb{F}$.

We now express the neighborhood functions of the graph in terms of affine functions over $\mathbb{F}^m$. This is where the nice structure of the wrapped de Bruijn graph will be

useful. For any positive integer i, define affine transformations $\tilde{\Gamma}_{i,0}, \tilde{\Gamma}_{i,1} : \mathbb{F}^m \to \mathbb{F}^m$ as follows: $\tilde{\Gamma}_{i,0}$ is the identity function. For $\tilde{\Gamma}_{i,1}$, first let $t = \lfloor i/h \rfloor \bmod m$ and $u = i \bmod h$. Then $\tilde{\Gamma}_{i,1}(z_0, \ldots, z_{m-1}) = (z_0, \ldots, z_{t-1}, z_t + e_u, z_{t+1}, \ldots, z_{m-1})$.[26] It can be checked from the above definition that for any layer i and node x in layer i (which we view as a point in H^m), we have $\tilde{\Gamma}_{i,j}(x) = \Gamma_{i,j}(x)$ for $j = 0, 1$. In other words, $\tilde{\Gamma}$ is an extension of the neighborhood relations of the graph $\mathcal{G}_{N,l}$ over $\mathbb{F}^m$.

Finally, we now express the assignment constraints (see Definition 6.3) as polynomial identities. The first of these identities checks that the assignments given by the assignment polynomial $\tilde{A}_i$ are actually elements of $\{0,1\}^4$ for points in H^m. For this purpose, let $\psi_0 : \mathbb{F} \to \mathbb{F}$ be the univariate polynomial of degree 2^4 given by

$$\psi_0(z) = \prod_{\alpha \in \{0,1\}^4} (z - \alpha). \tag{6.1}$$

This polynomial satisfies $\psi_0(z) = 0$ iff $z \in \{0,1\}^4$ (recall we identified $\{0,1\}^4$ with a subset of $\mathbb{F}$ spanned by $e_0, \ldots, e_3$). We check that $\psi_0(\tilde{A}_i(x)) = 0$ for all $x \in H^m$ and all layers i. We then arithmetize the rule ψ (from Definition 6.3) to obtain a polynomial $\psi_1 : \mathbb{F}^4 \to \mathbb{F}$. In other words, $\psi_1 : \mathbb{F}^4 \to \mathbb{F}$ is a polynomial such that $\psi_1(t, a, a_0, a_1) = \psi(t, a, a_0, a_1)$ for all $(t, a, a_0, a_1) \in \mathcal{C} \times (\{0,1\}^4)^3$. The degree of ψ_1 can be made constant, because $|\mathcal{C}|$ and $|\{0,1\}^4|$ are constant.[27] The two polynomial identities we would like to check are $\psi_0(\tilde{A}_i(x)) = 0$ and $\psi_1(\tilde{T}_i(x), \tilde{A}_i(x), \tilde{A}_{i+1}(\tilde{\Gamma}_{i,0}(x)), \tilde{A}_{i+1}(\tilde{\Gamma}_{i,1}(x))) = 0$ for all $x \in H^m$. For notational convenience, we express these two conditions together as a pair of polynomials $\psi' = (\psi_0, \psi_1) : \mathbb{F}^4 \to \mathbb{F}^2$ such that $\psi'(x_1, x_2, x_2, x_4) = (\psi_0(x_2), \psi_1(x_1, x_2, x_3, x_4))$.[28] Let κ be the maximum of the degree of these two polynomials. In order to make these polynomial identities sufficiently redundant, we set c_F to be a sufficiently large constant (say 100) such that $\kappa m^2 2^h / |\mathbb{F}|$ is low.

We have thus reduced Structured-CktSAT to an algebraic consistency problem, which we shall call the AS-CktSAT(Algebraic-Structured-CktSAT) problem.[29]

Definition 6.9. *The promise problem* AS-CktSAT = (AS-CktSAT$^{\text{YES}}$, AS-CktSAT$^{\text{NO}}$) *has as its instances* $\langle 1^n, 1^m, \mathbb{F}, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle$, *where* $\mathbb{F}$ *is a finite extension field of* $\mathbb{F}_2$ *(i.e.,* $\mathbb{F} = \mathbb{F}_{2^\upsilon}$ *for some* υ*),* H *an* $\mathbb{F}_2$*-linear subspace of* $\mathbb{F}$*, and* $\tilde{T}_i : \mathbb{F}^m \to \mathbb{F}$ *for* $i = 0, \ldots, l-1$ *a sequence of polynomials of degree* d*, where* $|H| = n^{1/m}$*,*

[26] An alternate description of $\tilde{\Gamma}_{i,1}$ is as follows: Since $\mathbb{F} = \mathbb{F}_2^\upsilon$, we can view $\mathbb{F}^m$ as an $m\upsilon$-dimensional space over $\mathbb{F}_2$. Hence, any vector $(z_0, \ldots, z_{m-1})$ can be written as $(b_{0,0}, \ldots, b_{0,\upsilon-1}, b_{1,0}, \ldots, b_{1,\upsilon-1}, \ldots, b_{m-1,0}, \ldots, b_{m-1,\upsilon-1})$. Furthermore, we note that for any vector $(z_0, \ldots, z_{m-1})$ in H^m, $b_{r,s} = 0$ for all $s \geq h$ and all r. It can now be checked that $\tilde{\Gamma}_{i,1}$ is the affine transformation that flips the bit $b_{t,u}$, where $t = \lfloor i/h \rfloor \bmod m$ and $u = i \bmod h$.

[27] Notice that we do not specify ψ_1 uniquely at this stage. Any choice of a constant-degree polynomial will work in this section, but to enforce linearity, we will use a somewhat nonstandard choice in section 8.4. Specifically, we argue that if C is a linear circuit, then ψ_1 can be picked to be an $\mathbb{F}_2$-linear transformation, and we point out that ψ_0 is an $\mathbb{F}_2$-linear transformation. For more details see section 8.4.

[28] An alternative approach (which we do not follow) to combine ψ_0 and ψ_1 into a single polynomial ψ was suggested to us by Sergey Yekhanin and Jaikumar Radhakrishnan. Let $p(x)$ be a monic, quadratic, irreducible polynomial over the field $\mathbb{F}$. Let $Q(x, y) = y^2 \cdot p(x, y)$. Now Q satisfies the property that $Q(a, b) = 0$ iff $a = b = 0$. (Proof: Q is homogeneous and so $Q(0, 0) = 0$; if $b = 0$, $Q(a, 0) = a^2$ (since p is monic) and so is zero only if $a = 0$. If $b \neq 0$, then $Q(a, b) = 0$ only if $p(a/b) = 0$, but p has no roots in $\mathbb{F}$.) Now define $\psi(x) = Q(\psi_0(x), \psi_1(x))$. For instance, over fields of odd characteristic, $\psi(x) = \psi_0^2(x) - \alpha \cdot \psi_1^2(x)$, where α is a nonsquare in $\mathbb{F}$ will work. ψ can be thought of as an "algebraic AND" of ψ_0 and ψ_1, since ψ' satisfies the property that $\psi(x) = 0$ iff $\psi_0(x) = 0$ and $\psi_1(x) = 0$.

[29] AS-CktSAT is actually a promise problem.

$d = m \cdot |H|$, and $|\mathbb{F}| = c_F \cdot md$. The field $\mathbb{F}$ is specified by an irreducible polynomial $p(x)$ of degree f over $\mathbb{F}_2$, H is taken to be spanned by the first $h = \log |H|$ canonical basis elements, and each of the polynomials $\tilde{T}_i$ is specified by a list of coefficients as follows:

- *$\langle 1^n, 1^m, \mathbb{F}, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle \in$ AS-CktSAT$^{\text{YES}}$ if there exists a sequence of degree d polynomials $\tilde{A}_i : \mathbb{F}^m \to \mathbb{F}, i = 0, \ldots, l-1$, such that for all $i = 0, \ldots, l-1$ and all $x \in H^m$,*

$$\psi'\left(\tilde{T}_i(x), \tilde{A}_i(x), \tilde{A}_{i+1}\big(\tilde{\Gamma}_{i,0}(x)\big), \tilde{A}_{i+1}\big(\tilde{\Gamma}_{i,1}(x)\big)\right) = (0,0).$$

- *$\langle 1^n, 1^m, \mathbb{F}, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle \in$ AS-CktSAT$^{\text{NO}}$ if for all functions $\tilde{A}_i : \mathbb{F}^m \to \mathbb{F}, i = 0, \ldots, l-1$, there exists an $i \in \{0, \ldots, l-1\}$ and $x \in H^m$ such that*

$$\psi'\left(\tilde{T}_i(x), \tilde{A}_i(x), \tilde{A}_{i+1}\big(\tilde{\Gamma}_{i,0}(x)\big), \tilde{A}_{i+1}\big(\tilde{\Gamma}_{i,1}(x)\big)\right) \neq (0,0),$$

where the $\tilde{\Gamma}_{i,j}$'s and ψ' are as defined earlier. (Recall that the $\tilde{\Gamma}$'s are linear functions while ψ' represents a pair of polynomials of degree at most κ.)

From the above discussion we have the following reduction from Structured-CktSAT to AS-CktSAT.

Proposition 6.10. *There exists a polynomial-time computable function $\mathcal{R}$ mapping any instance $\mathcal{I} = \langle \mathcal{G}_{N,l}, \{T_0, T_1, \ldots, T_{l-1}\}\rangle$ of* Structured-CktSAT *and parameter $m \leq \log n / \log\log n$ (where $n = |\mathcal{I}|$) to an instance $\mathcal{R}(\mathcal{I}, 1^m)$ of* AS-CktSAT *such that*

$$\mathcal{I} \in \text{Structured-CktSAT} \implies \mathcal{R}(\mathcal{I}, 1^m) \in \text{AS-CktSAT}^{\text{YES}},$$
$$\mathcal{I} \notin \text{Structured-CktSAT} \implies \mathcal{R}(\mathcal{I}, 1^m) \in \text{AS-CktSAT}^{\text{NO}}.$$

Moreover, if $\mathcal{R}(\mathcal{I}, 1^m) = \langle 1^{n'}, 1^{m'}, \mathbb{F}, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l'-1}\}\rangle$, then $n' = 2^N$ (the number of nodes in each layer of the de Bruijn graph $\mathcal{G}_{N,l}$), $m' = m$, and $l' = l$ (the number of layers in the de Bruijn graph).

Combining Propositions 6.5 and 6.10, we have the following.

Proposition 6.11. *There exists a polynomial-time computable function $\mathcal{R}$ mapping any circuit C and parameter $m \leq \log n / \log\log n$ (where $n = |C|$) to an instance $\mathcal{R}(C, 1^m)$ of* AS-CktSAT *such that $C \in$ CktSAT $\iff \mathcal{R}(C, 1^m) \in$ AS-CktSAT.*

Moreover, if C is a circuit of size n, then $\mathcal{R}(C, 1^m) = \langle 1^{n'}, 1^{m'}, \mathbb{F}, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l'-1}\}\rangle$ where $n' = \Theta(n)$, $m' = m$, and $l' \leq 5 \log n'$. Thus, $|\mathcal{R}(C, 1^m)| = O((c_F m^2)^m \log n) \cdot |C|$.

Remark 6.12. Following Remark 6.7, if C is a satisfiable circuit, then any set of polynomials $\tilde{A}_0, \ldots, \tilde{A}_{l-1}$ proving that the reduced instance $\mathcal{R}(C, 1^m) = \langle 1^n, 1^m, \mathbb{F}, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle$ is a YES instance of AS-CktSAT contains within it a satisfying assignment to the circuit C. Specifically, the set I (of layer 0 nodes with INPUT functionality in $\mathcal{G}_{N,l}$) from Remark 6.7 can now be viewed as a subset $I \subseteq H^m$. Then the polynomial $\tilde{A}_0 : \mathbb{F}^m \to \mathbb{F}$ restricted to the set I (i.e., $\tilde{A}_0|_I$) contains a satisfying assignment (again as a concatenation of third-bits). Conversely, every satisfying assignment w to C can be extended to a set of polynomials $\tilde{A}_0, \ldots, \tilde{A}_{l-1}$ such that $\tilde{A}_0|_I$ contains w. This is done by taking low-degree extensions of the functions $A_0, \ldots, A_{l-1}$ from Remark 6.7.

Remark 6.13. Following Remark 6.8, if C is a linear circuit, then the mapping of satisfying assignments w of C to polynomials $\tilde{A}_0, \ldots, \tilde{A}_{l-1}$ satisfying $\mathcal{R}(C)$ is $\mathbb{F}_2$-linear. This is due to Remark 6.8, the association of $\{0,1\}^4$ with the linear space

spanned by $\{e_0, e_1, e_2, e_3\}$ in $\mathbb{F}$, and from the fact that the interpolation from A_i to $\tilde{A}_i$ is $\mathbb{F}$-linear and hence $\mathbb{F}_2$-linear. For more information see section 8.4.

Comment. Recall that the arithmetization was obtained by considering low-degree extensions over $\mathbb{F}^m$ of functions from H^m to H. If H were a subfield of the field $\mathbb{F}$, this step would have caused a quadratic blow-up, and we avoid this problem by not insisting that H be a field. In [PS94, Spi95], H is a field and $\mathbb{F} = H^2$ is an extension of it, but the PCP system refers only to an $O(|H|)$-sized subset of $\mathbb{F}$. We cannot take this approach because we will be using a total low-degree test, which needs to refer to the entire vector space $\mathbb{F}^m$. In contrast, in [PS94, Spi95] an individual low-degree test is used, which can work with a subset of $\mathbb{F}^m$.

6.3. The PCP verifier. We design a PCP verifier for CktSAT via the reduction to AS-CktSAT based on the randomness-efficient low-degree tests of Ben-Sasson et al. [BSVW03]. Given a circuit C, the verifier reduces it to an instance of the problem AS-CktSAT (Proposition 6.11). The proof consists of a sequence of oracles $\tilde{A}_i : \mathbb{F}^m \to \mathbb{F}$ for $i = 0, \ldots, l-1$ and an auxiliary sequence of oracles $P_{i,j} : \mathbb{F}^m \to \mathbb{F}^2$ for $i = 0, \ldots, l-1$ and $j = 0, \ldots, m$. For each i and j, we view the auxiliary oracle $P_{i,j} : \mathbb{F}^m \to \mathbb{F}^2$ as a pair of functions $P_{i,j}^{(0)} : \mathbb{F}^m \to \mathbb{F}$ and $P_{i,j}^{(1)} : \mathbb{F}^m \to \mathbb{F}$ (i.e., $P_{i,j}(x) = (P_{i,j}^{(0)}(x), P_{i,j}^{(1)}(x))$). This auxiliary sequence of oracles helps the verifier to check that the functions $\tilde{A}_i$ satisfy condition ψ' (see Definition 6.9).

The verifier expects the first subsequence of auxiliary oracles $P_{i,0}(\cdot)$ for $i = 0, \ldots, l-1$, to satisfy the following relation:

$$P_{i,0}(x) = \psi'\left(\tilde{T}_i(x), \tilde{A}_i(x), \tilde{A}_{i+1}\big(\tilde{\Gamma}_{i,0}(x)\big), \tilde{A}_{i+1}\big(\tilde{\Gamma}_{i,1}(x)\big)\right) \quad \forall x \in \mathbb{F}^m. \tag{6.2}$$

Furthermore, it expects $P_{i,0}(x) = 0$ for every $x \in H^m$. Indeed, by Definition 6.9, we have the following.

Lemma 6.14.

1. *If* $\langle 1^n, 1^m, \mathbb{F}, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle$ *is a YES instance of* AS-CktSAT*, satisfied by polynomials* $\tilde{A}_0, \ldots, \tilde{A}_{l-1}$*, and* $P_{0,0}, \ldots, P_{l-1,0}$ *are defined according to* (6.2)*, then* $P_{i,0}(x) = (0,0)$ *for all* $x \in H^m$*.*

2. *If* $\langle 1^n, 1^m, \mathbb{F}, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle$ *is a NO instance of* AS-CktSAT*, then for any sequences of functions* $\tilde{A}_0, \ldots, \tilde{A}_{l-1}$*,* $P_{0,0}, \ldots, P_{l-1,0}$*, either* (6.2) *fails to hold for some* i *or* $P_{i,0}(x) \neq (0,0)$ *for some* i *and some* $x \in H^m$*.*

Recalling that the degree of the constraint ψ' (see Definition 6.9) is at most κ and that the $\tilde{A}_i$'s are of degree at most $d = m \cdot |H|$, we observe that the $P_{i,0}$'s can be taken to be of degree at most κd in item 1 above.

As mentioned above, the verifier now needs to check that the functions $P_{i,0}$ vanish on the set H^m. For this we use a "zero-propagation test" based on the sum-check protocol of Lund et al. [LFKN92]. Specifically, the verifier expects the remaining set of auxiliary oracles $P_{i,j} = (P_{i,j}^{(0)}, P_{i,j}^{(1)})$ ($i = 0, \ldots, l-1$ and $j = 1, \ldots, m$) to satisfy the following relations: Let $H = \{h_0, \ldots, h_{|H|-1}\}$ be some fixed enumeration of the elements in H. For all $b \in \{0,1\}$,

$$P_{i,j}^{(b)}\left(\underbrace{x_1, \ldots, x_{j-1}}, x_j, \underbrace{x_{j+1}, \ldots, x_m}\right) = \sum_{k=0}^{|H|-1} P_{i,j-1}^{(b)}\left(\underbrace{x_1, \ldots, x_{j-1}}, h_k, \underbrace{x_{j+1}, \ldots, x_m}\right) x_j^k \quad \forall (x_1, \ldots, x_m) \in \mathbb{F}^m. \tag{6.3}$$

These relations ensure that for all i and $j \geq 1$, $P_{i,j}(\cdot)$ vanishes on $\mathbb{F}^j \times H^{m-j}$ iff the function $P_{i,j-1}(\cdot)$ vanishes on $\mathbb{F}^{j-1} \times H^{m-j+1}$. For future reference, we (re)state this fact as a lemma.

LEMMA 6.15. $P^{(b)}_{i,j}|_{\mathbb{F}^j \times H^{m-j}} \equiv 0 \iff P^{(b)}_{i,j-1}|_{\mathbb{F}^{j-1} \times H^{m-j+1}} \equiv 0$.

Thus, for all i, $P_{i,m}$ vanishes on the entire space $\mathbb{F}^m$ iff $P_{i,0}$ vanishes on H^m. Also, as $P^{(b)}_{i,0}$ has degree at most κd in each variable, so does $P^{(b)}_{i,j}$ for each i and j. Hence, the degree of $P^{(b)}_{i,j}$ is at most κd.

Thus, the verifier needs to make the following checks:

- LOW-DEGREE TEST.
  For $i = 0, \ldots, l-1$ and $j = 0, \ldots, m$, the functions $\tilde{A}_i$ are polynomials of degree at most $d = m \cdot |H|$ and the functions $P_{i,j}$ are pairs of polynomials of degree at most κd.
- EDGE-CONSISTENCY TEST.
  For $i = 0, \ldots, l-1$, the functions $P_{i,0}$ obey (6.2).
- ZERO-PROPAGATION TEST.
  For $i = 0, \ldots, l-1$ and $j = 1, \ldots, m$, the functions $P_{i,j}$ satisfy (6.3).
- IDENTITY TEST.
  For $i = 0, \ldots, l-1$, the functions $P_{i,m}$ are identically zero on the entire domain $\mathbb{F}^m$.

The low-degree test in most earlier construction of PCP verifiers is performed using the "line-point" test. The "line-point" low degree test first chooses a random line, a random point on this line, and checks if the restriction of the function to the line (given by a univariate polynomial) agrees with the value of the function at the point. A random line l is typically chosen by choosing two random points $x, y \in \mathbb{F}^m$ and setting $l = l_{x,y} = \{x + ty | t \in \mathbb{F}\}$. However, this requires $2m \log |\mathbb{F}|$ bits of randomness, which is too expensive for our purposes. We save on randomness by using the low-degree test of Ben-Sasson et al. [BSVW03] based on small-biased spaces (see Appendix B for more details). The low-degree test of [BSVW03] uses pseudorandom lines instead of totally random lines in the following sense: The pseudorandom line $l = l_{x,y}$ is chosen by choosing the first point x at random from $\mathbb{F}^m$, while the second point y is chosen from a λ-biased subset S_λ of $\mathbb{F}^m$. This needs only $\log |S_\lambda| + \log |\mathbb{F}|^m$ bits of randomness. We further save on randomness by the use of *canonical lines*.[30] Consider any pseudorandom line $l = l_{x,y}$, where $x \in \mathbb{F}^m$ and $y \in S_\lambda$. We observe that for every $x' \in l$, we have $l_{x',y} = l_{x,y}$. In other words, $|\mathbb{F}|$ different choices of random bits lead to the same line $l_{x,y}$. We prevent this redundancy by representing each line in a canonical manner. A canonical line is chosen by first choosing a random point y from the λ-biased set S_λ. We view this y as specifying the direction (i.e., slope) of the line. This direction partitions the space $\mathbb{F}^m$ into $|\mathbb{F}|^{m-1}$ parallel lines (each with direction y). We enumerate these lines arbitrarily and select one of them uniformly at random. Thus, choosing a random canonical line costs only $\log |S_\lambda| + \log |\mathbb{F}|^{m-1}$ bits of randomness. A further point to be noted is that we perform a "line" test instead of the regular line-point test: The test queries the function for all points along the canonical line $l_{x,y}$ and verifies that the restriction of the function to this line is a low-degree polynomial.

Having performed the low-degree test (i.e., verified that the polynomials $\tilde{A}_i$'s and P_{ij}'s are close to low-degree polynomials), the verifier then performs the NODE-

[30] It is to be noted that the canonical representation of lines has been used either implicitly or explicitly in the soundness analysis of all earlier uses of the low-degree test. However, this is the first time that the canonical representation is used to save on the number of random bits.

Consistency Test, Zero-Propagation Test, and Identity Test by choosing a suitable small-sized sample in the entire space and checking if the corresponding condition is satisfied on that sample. For the Zero-Propagation Test indeed the natural sample is an axis-parallel line. For the Edge-Consistency Test and Identity Test, the sample we use is any set of $|\mathbb{F}|$ points selected from a partition of $\mathbb{F}^m$ into $|\mathbb{F}|^{m-1}$ equal sets.

We are now ready to formally describe the PCP verifier for CktSAT. We parameterize the PCP verifier in terms of m, the number of dimensions in our intermediate problem AS-CktSAT, and λ, the parameter of the λ-biased sets of $\mathbb{F}^m$ required for the low-degree tests of Ben-Sasson et al. [BSVW03]. We rely on the fact that λ-biased subsets of $\mathbb{F}^m$ of size at most $\text{poly}(\log|\mathbb{F}|^m, 1/\lambda)$ can be constructed efficiently [NN93, AGHP92].

PCP $\text{Verifier}_{m,\lambda}^{\tilde{A}_i, P_{i,j}; i=0,\ldots,l-1; j=0,\ldots,m}(C)$.

1. Use Proposition 6.11 to reduce the instance C of CktSAT, using parameter m, to an instance $\langle 1^n, 1^m, \mathbb{F}, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle$ of AS-CktSAT, and set $d = m \cdot |H|$.
   Notation. We let $S_\lambda \subset \mathbb{F}^m$ be a λ-biased set of size at most $(\frac{\log|\mathbb{F}|^m}{\lambda})^2$ [AGHP92]. Let $\mathbb{F}^m = \biguplus_{\eta=1}^{|\mathbb{F}|^{m-1}} U_\eta$ and $\mathbb{F}^m = \biguplus_{\eta=1}^{|\mathbb{F}|^{m-1}} V_\eta$ be two arbitrary partitions of the space $\mathbb{F}^m$ into $|\mathbb{F}|$-sized sets each.
2. Choose a random string R of length $\log(|S_\lambda| \cdot |\mathbb{F}|^{m-1})$. (Note: We reuse R in all tests, but only the Low-Degree Test utilizes the full length of R.)
3. Low-Degree Test.
   Use random string R to determine a random canonical line $\mathcal{L}$ in $\mathbb{F}^m$ using the λ-biased set S_λ.
   For $i = 0, \ldots, l-1$,
   query oracle $\tilde{A}_i$ on all points along the line $\mathcal{L}$ and reject if the restriction $\tilde{A}_i$ to $\mathcal{L}$ is not a (univariate) polynomial of degree at most d.
   For $i = 0, \ldots, l-1$, $j = 0, \ldots, m$, and $b \in \{0,1\}$,
   query oracle $P_{i,j}^{(b)}$ on all points along the line $\mathcal{L}$ and reject if the restriction of $P_{i,j}^{(b)}$ to $\mathcal{L}$ is not a (univariate) polynomial of degree at most κd.
4. Edge-Consistency Test.
   Use the random string R to determine a random set U_η of the partition $\mathbb{F}^m = \biguplus_{\eta=1}^{|\mathbb{F}|^{m-1}} U_\eta$.
   For $i = 0, \ldots, l-1$,
   for all $x \in U_\eta$, query $P_{i,0}(x), \tilde{A}_i(x), \tilde{A}_{i+1}(\tilde{\Gamma}_{i,0}(x))$, and $\tilde{A}_{i+1}(\tilde{\Gamma}_{i,1}(x))$ and reject if (6.2) is not satisfied.
5. Zero-Propagation Test.
   For $i = 0, \ldots, l-1$, $j = 1, \ldots, m$, and $b \in \{0,1\}$,
   use random string R to determine a random jth axis-parallel line in $\mathbb{F}^m$ of the form $\mathcal{L} = \{(a_1, \ldots, a_{j-1}, X, a_{j+1}, \ldots, a_m) : X \in \mathbb{F}\}$. Query $P_{i,j-1}^{(b)}$ and $P_{i,j}^{(b)}$ along all the points in $\mathcal{L}$. Reject if either the restriction of $P_{i,j-1}^{(b)}$ or $P_{i,j}^{(b)}$ to $\mathcal{L}$ is not a univariate polynomial of degree at most κd or if any of the points on the line $\mathcal{L}$ violate (6.3).
6. Identity Test.
   Use the random string R to determine a random set V_η of the partition

$\mathbb{F}^m = \biguplus_{\eta=1}^{|\mathbb{F}|^{m-1}} V_\eta$. For $i = 0, \ldots, l-1$,
for all $x \in V_\eta$, query $P_{i,m}(x)$. Reject if any of these $P_{i,m}(x)$ are not $(0,0)$.
Accept if none of the above tests reject.

Remark 6.16.

1. The Low-Degree Test requires $\log(|S_\lambda| \cdot |\mathbb{F}|^{m-1})$ random bits to generate a canonical line in $\mathbb{F}^m$ using the λ-biased set, while each of the other tests requires at most $\log(|\mathbb{F}|^{m-1})$ bits of randomness. Hence, the string R suffices for each of the tests. For the settings of parameters we use, $\log(|S_\lambda| \cdot |\mathbb{F}|^{m-1})$ is typically significantly smaller than $\log(|\mathbb{F}|^m)$, which we would not be able to afford.

2. The Edge-Consistency Test and Identity Test in the "standard" sense are usually performed by selecting a random point in the space $\mathbb{F}^m$ and checking whether the corresponding condition is satisfied. However, we state these tests in a "nonstandard" manner using partitions of the space $\mathbb{F}^m$ into $|\mathbb{F}|$-sized tests so that these tests can be easily adapted when we construct the robust PCP verifier (see section 8). The nonstandard tests are performed in the following manner: Choose a random set in the partition and perform the standard test for each point in the set. At present, we can work with any partition of $\mathbb{F}^m$; however, we will later need specific partitions to get "robustness."

6.4. Analysis of the PCP Verifier**.** We now analyze the PCP Verifier above. The analysis below assumes that the parameters satisfy $m \le \log n / \log\log n$ and $\lambda \le 1/c \log n$ for a sufficiently large constant c. Theorem 6.1 can be deduced by setting $\lambda = 1/c \log n$.

Complexity. The PCP Verifier makes $O(lm|\mathbb{F}|) = O(m^3 n^{1/m} \log n)$ queries, each of which expects as an answer an element of $\mathbb{F}$ or $\mathbb{F}^2$ (i.e., a string of length $O(\log|\mathbb{F}|)$). Hence, the total (bit) query complexity is $O(lm|\mathbb{F}| \log|\mathbb{F}|) = O(lm \cdot c_F m^2 n^{1/m} \log(c_F m^2 n^{1/m}))$. Recalling that $l = 5 \log n$, this quantity is at most $O(m^2 n^{1/m} \log^2 n)$ for $m \le \log n$. For the decision complexity, we note that the main computations required are (a) testing whether a function is a low-degree univariate polynomial over $\mathbb{F}$ (for Low-Degree Test and Zero-Propagation Test), (b) evaluating ψ' on $|\mathbb{F}|$ quadruples of points (for Edge-Consistency Test), and (c) univariate polynomial interpolation and evaluation (for testing (6.3) in Zero-Propagation Test). We now argue that each of these can be done with a nearly linear ($\widetilde{O}(|\mathbb{F}|)$) number of operations over $\mathbb{F}$, yielding a nearly linear ($\widetilde{O}(q)$) decision complexity overall. Each evaluation of ψ' can be done with a constant number of $\mathbb{F}$-operations because ψ' is of constant degree. Polynomial interpolation and evaluation can be done with a nearly linear number of $\mathbb{F}$-operations by [SS71, Sch77], and testing whether a function is of low degree reduces to polynomial interpolation (this is achieved by interpolating the function to represent it as a polynomial of degree $|\mathbb{F}|-1$ and checking that the high-degree coefficients are zero). Each $\mathbb{F}$-operation can be done with $\widetilde{O}(\log|\mathbb{F}|)$ bit operations, using the polynomial multiplication algorithm of [SS71, Sch77] (over $\mathbb{F}_2$).

The number of random bits used by the verifier is exactly $\log(|S_\lambda| \cdot |\mathbb{F}|^{m-1})$. Let $n' = |\mathbb{F}|^m$. Then $\log(|S_\lambda| \cdot |\mathbb{F}|^{m-1}) = \left(1 - \frac{1}{m}\right) \log n' + \log(\mathrm{poly}(\frac{\log n'}{\lambda})) = \left(1 - \frac{1}{m}\right) \log n' + O(\log\log n') + O\left(\log\left(\frac{1}{\lambda}\right)\right)$. Now, $n' = (c_F m^2)^m n$. Hence, $\log n' = \log n + 2m \log m + O(m)$ and $\log\log n' = \log\log n + O(\log m)$. Thus, the total randomness is at most $\left(1 - \frac{1}{m}\right) \log n + O(m \log m) + O(\log\log n) + O\left(\log\left(\frac{1}{\lambda}\right)\right)$.

We summarize the above observations in the following proposition for future ref-

erence.

PROPOSITION 6.17. *The randomness, query, and decision complexities of* PCP VERIFIER *are* $r = \left(1 - \frac{1}{m}\right)\log n + O(m \log m) + O(\log\log n) + O\left(\log\left(\frac{1}{\lambda}\right)\right)$, $q = O(m^2 n^{1/m} \log^2 n)$ *and* $d = \widetilde{O}(q)$, *respectively.*

Completeness. If C is satisfiable, then the reduction reduces it to a YES instance of AS-CKTSAT. Then by definition there exist polynomials $\tilde{A}_i$ that satisfy constraint ψ'. Setting $P_{i,j}$ according to (6.2) and (6.3), we notice that the verifier accepts with probability one.

Soundness. To prove the soundness, we need to prove that if C is not satisfiable then the verifier accepts with probability bounded away from 1. We will prove a stronger statement. Recall from Remark 6.12 that the function $\tilde{A}_0 : \mathbb{F}^m \to \mathbb{F}$ supposedly has the satisfying assignment embedded within it. Let $I \subset \mathbb{F}^m$ be the set of locations in $\mathbb{F}^m$ that contains the assignment (i.e., $\tilde{A}_0|_I$ is supposedly the assignment).

LEMMA 6.18. *There exists a constant c and a constant $0 < \varepsilon_0 < 1$ such that for all ε, m, λ satisfying $\varepsilon \le \varepsilon_0$, $m \le \log n / \log\log n$, and $\lambda \le 1/c \log n$, the following holds. If $\tilde{A}_0$ is 4ε-far from every polynomial $\widehat{A}_0$ of degree md such that $C(\widehat{A}_0|_I) = 1$, then for all proof oracles $\{\tilde{A}_i\}$ and $\{P_{i,j}\}$, the verifier accepts with probability at most $1 - \varepsilon$.*

Proof. Let α be the universal constant from Theorem B.4. Set $\varepsilon_0 = \min\{\alpha, \frac{1}{22}\}$. Let $d = m2^h$, and choose c_F to be a large enough constant such that $\kappa md/|\mathbb{F}| = \kappa/c_F \le \varepsilon_0$. Suppose each of the functions $\tilde{A}_i$ are 4ε-close to some polynomial of degree md and each of the functions $P_{i,j}^{(b)}$ is 4ε-close to some polynomial of κmd. If this were not the case, then by Theorem B.4, LOW-DEGREE TEST accepts with probability at most $1 - \varepsilon$ for the polynomial that is 4ε-far. It can be verified that the parameters satisfy the requirements of Theorem B.4 for sufficiently large choices of the constants c_F and c and sufficiently small ε.

For each $i = 0, \ldots, l-1$, let $\widehat{A}_i : \mathbb{F}^m \to \mathbb{F}$ be the polynomial of degree at most md that is 4ε-close to $\tilde{A}_i$. Similarly, for each $i = 0, \ldots, l-1, j = 0, \ldots, m$ and $b \in \{0,1\}$, let $\widehat{P}_{i,j}^{(b)}$ be the polynomial of degree at most κmd that is 4ε-close to $P_{i,j}^{(b)}$. Such polynomials are uniquely defined since every two polynomials of degree κmd disagree in at least a $1 - \frac{\kappa md}{|\mathbb{F}|} \ge 1 - \varepsilon_0 > 8\varepsilon$ fraction of points. As in the case of the $P_{i,j}$'s, let $\widehat{P}_{i,j} : \mathbb{F}^m \to \mathbb{F}^2$ be the function given by $\widehat{P}_{i,j}(x) = (\widehat{P}_{i,j}^{(0)}(x), \widehat{P}_{i,j}^{(1)}(x))$.

By hypothesis, $\widehat{A}_0|_I$ does not satisfy C. Then, by Lemmas 6.14 and 6.15, at least one of the following must hold.

(a) *There exists $i \in \{0, \ldots, l-1\}$ and $b \in \{0,1\}$ such that $\widehat{P}_{i,m}^{(b)} \not\equiv 0$.*
Then for this i, IDENTITY TEST fails unless a random set V_η is chosen such that for all $x \in V_\eta$, $P_{i,m}^{(b)}(x) = 0$. Hence, it must be the case that for all $x \in V_\eta$, either $P_{i,m}^{(b)}(x) \ne \widehat{P}_{i,m}^{(b)}(x)$ or $\widehat{P}_{i,m}^{(b)}(x) = 0$. Since the $V_\eta's$ form a partition of $\mathbb{F}^m$, the probability of this occurring is upper-bounded by the probability that a random $x \in \mathbb{F}^m$ satisfies either $P_{i,m}^{(b)}(x) \ne \widehat{P}_{i,m}^{(b}(x)$ or $\widehat{P}_{i,m}^{(b)}(x) = 0$. This probability is at most $4\varepsilon + \frac{\kappa md}{|\mathbb{F}|} = 4\varepsilon + \frac{\kappa}{c_F} \le 5\varepsilon_0$, where we use the fact that $\widehat{P}_{i,m}^{(b)}$ is 4ε-close to $P_{i,m}^{(b)}$ and that a nonzero polynomial of degree κmd vanishes on at most a $\kappa md/|\mathbb{F}|$ fraction of points.

(b) *There exists $i \in \{0, \ldots, l-1\}$ such that $\widehat{P}_{i,0}$, $\widehat{A}_i$, and $\widehat{A}_{i+1}$ do not obey* (6.2). In other words, $\widehat{P}_{i,0}(x) \not\equiv \psi'(\tilde{T}_i(x), \widehat{A}_i(x), \widehat{A}_{i+1}(\tilde{\Gamma}_{i,0}(x)), \widehat{A}_{i+1}(\tilde{\Gamma}_{i,1}(x)))$.
Then for this i, EDGE-CONSISTENCY TEST fails unless a random partition U_η is chosen such that for all $x \in U_\eta$, $P_{i,0}(x) = \psi'(\tilde{T}_i(x), \tilde{A}_i(x), \tilde{A}_{i+1}(\tilde{\Gamma}_{i,0}(x))$,

$\tilde{A}_{i+1}(\tilde{\Gamma}_{i,1}(x)))$. Hence, it must be the case that for every $x \in U_\eta$, one of the following (six) holds:

$$P^{(0)}_{i,0}(x) \neq \widehat{P}^{(0)}_{i,0}(x); \qquad P^{(1)}_{i,0}(x) \neq \widehat{P}^{(1)}_{i,0}(x); \qquad \tilde{A}_i(x) \neq \widehat{A}_i(x);$$
$$\tilde{A}_{i+1}(\tilde{\Gamma}_{i,0}(x)) \neq \widehat{A}_{i+1}(\tilde{\Gamma}_{i,0}(x)); \qquad \tilde{A}_{i+1}(\tilde{\Gamma}_{i,1}(x)) \neq \widehat{A}_{i+1}(\tilde{\Gamma}_{i,1}(x));$$
$$\widehat{P}_{i,0}(x) = \psi'(\tilde{T}_i(x), \widehat{A}_i(x), \widehat{A}_{i+1}(\tilde{\Gamma}_{i,0}(x)), \widehat{A}_{i+1}(\tilde{\Gamma}_{i,1}(x))).$$

The probability of this happening is at most the probability that a random $x \in \mathbb{F}^m$ satisfies these conditions, which is at most $5 \cdot 4\varepsilon + \frac{\kappa m d}{|\mathbb{F}|} \leq 21\varepsilon_0$.

(c) *For some $i = 0, \ldots, l-1$, $j = 1, \ldots, m$, and $b \in \{0,1\}$, $\widehat{P}^{(b)}_{i,j}$ does not obey* (6.3).

In other words, $\widehat{P}^{(b)}_{i,j}(\ldots, x_j, \ldots) \not\equiv \sum_{k=0}^{|H|-1} \widehat{P}^{(b)}_{i,j-1}(\ldots, h_j, \ldots) x_j^k$. Then, for this i, j, Zero-Propagation Test rejects unless a random axis-parallel line $\mathcal{L}$ is chosen such that both $P^{(b)}_{i,j}|_{\mathcal{L}}$ and $P^{(b)}_{i,j-1}|_{\mathcal{L}}$ are polynomials of degree at most κd and for every $x \in \mathcal{L}$, $P^{(b)}_{i,j}(\ldots, x, \ldots) = \sum_{k=0}^{|H|-1} P^{(b)}_{i,j-1}(\ldots, h_k, \ldots) x^k$. Suppose we have that for all $x \in \mathcal{L}$, $P^{(b)}_{i,j}(x) = \widehat{P}^{(b)}_{i,j}(x)$ and $P^{(b)}_{i,j-1}(x) = \widehat{P}^{(b)}_{i,j-1}(x)$. Then, it must be the case that for all $x \in \mathcal{L}$, $\widehat{P}^{(b)}_{i,j}(\ldots, x, \ldots) = \sum_{k=0}^{|H|-1} \widehat{P}^{(b)}_{i,j-1}(\ldots, h_k, \ldots) x^k$. Since the axis-parallel lines cover $\mathbb{F}^m$ uniformly, the probability of this occurring is at most the probability of a random $x \in \mathbb{F}^m$ satisfying this condition, which is at most $\frac{\kappa m d}{c_F} \leq \varepsilon$. The probability that both $P^{(b)}_{i,j}|_{\mathcal{L}}$ and $P^{(b)}_{i,j-1}|_{\mathcal{L}}$ are polynomials of degree κd and either $P^{(b)}_{i,j}|_{\mathcal{L}} \neq \widehat{P}^{(b)}_{i,j}|_{\mathcal{L}}$ or $P^{(b)}_{i,j-1}|_{\mathcal{L}} \neq P^{(b)}_{i,j-1}|_{\mathcal{L}}$ is at most $2 \cdot 4\varepsilon/(1-\varepsilon_0) \leq 9\varepsilon_0$, since $P^{(b)}_{i,j}$ and $P^{(b)}_{i,j-1}$ are 4ε-close to $\widehat{P}^{(b)}_{i,j}$ and $\widehat{P}^{(b)}_{i,j-1}$, respectively, and any two distinct polynomials of degree $\kappa m d$ disagree on at least a $1 - \kappa m d/|\mathbb{F}| \geq 1 - \varepsilon_0$ fraction of points. Hence, Zero-Propagation Test accepts with probability at most $10\varepsilon_0$.

We thus have that the verifier accepts with probability at most $\max\{1-\varepsilon, 5\varepsilon_0, 21\varepsilon_0, 10\varepsilon_0\} = 1 - \varepsilon$. ∎

Proof of Theorem 6.1. Theorem 6.1 is proved using PCP Verifier defined in this section setting $\lambda = 1/c \log n$. Step 1 of the verifier reduces the instance C of CktSAT to an instance $\langle 1^{n'}, 1^m, \mathbb{F}, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle$ of AS-CktSAT. We have from Proposition 6.11 that $n' = \Theta(n)$ and $l = O(\log n)$, where n is the size of the input circuit C. Setting $n = n'$ in Proposition 6.17, we have that the randomness, query and decision complexity of the verifier are as claimed in Theorem 6.1. The soundness of the verifier follows from Lemma 6.18. ∎

7. A randomness-efficient PCPP. In this section, we modify the PCP for CktSAT and construct a PCPP for CktVal while maintaining all the complexities. (Recall that, by Proposition 2.4, the latter implies the former.) We do so by adding a proximity test to PCP Verifier defined in section 6.3. This new proximity test, as the name suggests, checks the closeness of the input to the satisfying assignment that is assumed to be encoded in the proof oracle (see Remark 6.12). This check is done by locally decoding a bit (or several bits) of the input from its encoding and comparing it with the actual input oracle.

Theorem 7.1. *There exists universal constants c and $0 < \varepsilon < 1$ such that the following holds for all $n, m \in \mathbb{Z}^+$, and $0 < \delta < 1$ satisfying $n^{1/m} \geq m^{cm}/\delta^3$. There exists a PCPP for* CktVal *(for circuits of size n) with the following parameters:*

- *randomness* $\left(1 - \frac{1}{m}\right) \log n + O(m \log m) + O(\log \log n) + O(\log(1/\delta))$,

- *query complexity* $q = O(m^2 n^{1/m} \log^2 n)$ *and decision complexity* $d = \widetilde{O}(q)$,
- *perfect completeness,*
- *soundness error* $1 - \varepsilon$ *for proximity parameter* δ.

Note that the condition $n^{1/m} \geq m^{cm}/\delta^3$ (made in Theorem 7.1) implies the condition $m \leq \log n / \log\log n$ stated in Theorem 6.1. Thus, the PCPP of Theorem 7.1 works only when n, the size of the circuit, is not too small (more, precisely, when $n \geq m^{cm^2}/\delta^{3m}$). As explained in section 3, when applying multiple proof compositions, we need (at the last compositions) PCPPs that work for even smaller values of n. For this purpose, we construct the following PCPP that works for small values of n. This PCPP, however, performs relatively poorly with respect to randomness (i.e., it has randomness complexity $O(\log n)$ rather than $(1 - o(1)) \log_2 n$). This will not be a concern for us since this verifier (or rather the robust version of this verifier) is used only in the inner levels of composition.

THEOREM 7.2. *For all* $n \in \mathbb{Z}^+$ *and* $\delta \in (0, 1)$, CKTVAL *has a PCPP (for circuits of size* n*) with the following parameters:*

- *randomness* $O(\log n)$*;*
- *decision complexity* $\operatorname{poly}\log n$*, which also upper-bounds the query complexity;*
- *perfect completeness; and*
- *soundness error* $1 - \Omega(\delta)$ *for proximity parameter* δ.

Preliminaries. Recall that a PCPP verifier is supposed to work as follows: The verifier is given explicit access to a circuit C with n gates on k input bits and oracle access to the input w in the form of an input oracle $W : [k] \to \{0, 1\}$. The verifier should accept W with probability 1 if it is a satisfying assignment and accept it with probability at most $1 - \varepsilon$ if it δ-far from any satisfying assignment.

For starters, we assume that $k \geq n/5$. In other words, the size of the input w is linear in the size of the circuit C. The reason we need this assumption is that we wish to verify the proximity of w to a satisfying assignment, but our proofs encode the assignment to all n gates of the circuit, and thus it better be the case that w is a nonnegligible fraction of the circuit. This assumption is not a major restriction, because if this is not the case then we work with the modified circuit C' and input w' that are defined as follows: For $t = \lceil n/k \rceil$, the circuit C' has $n' = n + 3tk$ gates and $k' = tk$ input bits such that $C'(w') = 1$ iff $w' = w^t$ for some w such that $C(w) = 1$; that is, C' checks if its input consists of t copies of some satisfying assignment of C. (It can be verified that C' can indeed be implemented on a circuit of size $n + 3tk$.) We choose t such that $k' \geq n'/10$. However, note that the input oracle W cannot be altered. So the verifier emulates the input w' using the original input oracle $W : [k] \to \{0, 1\}$ in a straightforward manner; that is, it defines $W' : [tk] \to \{0, 1\}$ such that $W'(i) \triangleq W(((i-1) \bmod k) + 1)$. Indeed, in view of the way W' is emulated based on W, testing that W' is a repetition of some k-bit string makes no sense. This test is incorporated into C' in order to maintain the distance features of C; that is, if w is δ-far from satisfying C, then $w' = w^t$ is δ-far from satisfying C' (without having C' explicitly run C on all t copies of w, because that would make its size larger than nt and defeat our goal of increasing the length of the input relative to the circuit size). We state this fact as a proposition for future reference.

PROPOSITION 7.3. *There exists a generic transformation from* CKTVAL *to* CKTVAL *that maps the instance* (C, w)*, where* C *is a circuit with* n *gates and* k *input bits, to the instance* (C', w')*, where* C' *is a circuit on* $n' = n + 3tk$ *gates and* $k' = kt$ *input bits (where* $t = \lceil n/k \rceil$*) defined as follows:* $C'(w') = 1$ *iff* $w' = w^t$ *for some* w *such that* $C(w) = 1$ *and* $w' = w^t$*. This transformation increases the length of the input oracle*

compared to the proof oracle (here, the values of all gates in C). The transformation preserves the relative distance to the set of satisfying assignments; that is, if w is δ-far from the set of satisfying assignments of C, then $w' = w^t$ is δ-far from the satisfying assignments of C'.

We first describe PCPP Verifier which proves Theorem 7.1 and later describe ALMSS PCPP Verifier which proves Theorem 7.2.

7.1. The construction of PCPP Verifier (Theorem 7.1). As in the case of PCP Verifier described in section 6.3, PCPP Verifier is constructed by reducing the input circuit C, an instance of CktSAT, using parameter m, to an instance $\langle 1^n, 1^m, \mathbb{F}, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle$ of AS-CktSAT. The proof oracle for PCPP Verifier is the same as that of the PCP Verifier (i.e., the proof oracle consists of a sequence of functions $\tilde{A}_i : \mathbb{F}^m \to \mathbb{F}, i = 0, \ldots, l-1$, and $P_{i,j} : \mathbb{F}^m \to \mathbb{F}^2, i = 0, \ldots, l-1, j = 0, \ldots, m$, where $l = 5 \log n$).

Recall that the function $\tilde{A}_0 : \mathbb{F}^m \to \mathbb{F}$ is assumed to contain within it an assignment (see Remarks 6.7 and 6.12). Let $I \subseteq H^m \subset \mathbb{F}^m$ be the set of locations in $\mathbb{F}^m$ that contain the assignment. PCPP Verifier, in addition to the tests of PCP Verifier, performs the following Proximity Test to check if the assignment given by $\tilde{A}_0|_I$ matches the input oracle W. Specifically, we have the following.

PCPP $\text{Verifier}^{W;\ \tilde{A}_i, P_{i,j}; i=0,\ldots,l-1; j=0,\ldots,m}_{m,\lambda,\delta}(C)$.

1. Run PCP $\text{Verifier}^{W;\ \tilde{A}_i, P_{i,j}}_{m,\lambda}(C)$ and reject if it rejects.
   Let R be the random string generated during the execution of this step.
2. Proximity Test.
   Use random string R to determine a random canonical line $\mathcal{L}$ in $\mathbb{F}^m$ using the λ-biased set S_λ. Query oracle $\tilde{A}_0$ on all points along the line $\mathcal{L}$ and reject if the restriction $\tilde{A}_0$ to $\mathcal{L}$ is not a polynomial of degree at most $d = m \cdot |H|$. Query the input oracle W on all locations corresponding to those in $I \cap \mathcal{L}$ and reject if W disagrees with $\tilde{A}_0$ on any of the locations in $I \cap \mathcal{L}$.

By inspection, the proximity test increases the query and decision complexities by (even less than) a constant factor. For the randomness complexity, the randomness is used only to generate a random canonical line (as in PCP Verifier), so the randomness complexity is $\log(|\mathbb{F}|^{m-1} \cdot |S_\lambda|)$ as before. However, in order to prove soundness, we need to assume not only that $\lambda \leq 1/c \log n$ for some constant c (as before), but also that $\lambda \leq \delta^3/m^{cm}$.[31] Thus, setting $\lambda = \min\{1/c \log n, \delta^3/m^{cm}\}$, the randomness complexity increases by at most $O(m \log m) + O(\log(1/\delta))$, as claimed in Theorem 7.1. Summarizing the above observations for future reference, we have the following proposition.

Proposition 7.4. *The randomness, query, and decision complexities of* PCPP Verifier *are* $r = \left(1 - \frac{1}{m}\right) \log n + O(m \log m) + O(\log \log n) + O(\log(1/\delta))$, $q = O(m^2 n^{1/m} \log^2 n)$, *and* $d = \widetilde{O}(q)$, *respectively.*

It is straightforward to check perfect completeness of this verifier. To prove soundness, we observe that if the input W is δ-far from satisfying the circuit, then one of the following must happen: (1) The verifier detects an inconsistency in the proof oracle or (2) the input oracle does not match the encoding of the input in the proof oracle. In the case of the former, we prove soundness by invoking Lemma 6.18, while in the latter case, we prove soundness by analyzing the proximity test. These ideas are

[31] Actually, for the proximity test we need only $\lambda \leq \delta/m^{cm}$; however, to prove robustness of the proximity test (see section 8.1) we require $\lambda \leq \delta^3/m^{cm}$.

explained in detail in the following lemma which proves the soundness of the verifier.

LEMMA 7.5. *There exists a constant c and a constant $\varepsilon > 0$ such that for all m, λ, δ satisfying $n \geq 8000|\mathbb{F}|^{m-1}/\delta^3$, $\lambda \leq 1/c \log n$, and $\lambda \leq \delta/m^{cm}$, the following holds. If the input w given by the input oracle $W : [k] \to \{0,1\}$ is δ-far from satisfying the circuit, then for any proof oracle the verifier rejects W with probability at least ε.*

Proof. Set ε to be the constant ε_0 in Lemma 6.18.

Case (i). $\tilde{A}_0$ is not 4ε-close to any polynomial $\widehat{A}_0$ of degree md such that $C(\widehat{A}_0|_I) = 1$. Then by Lemma 6.18, we conclude that the verifier rejects with probability at least ε.

Case (ii). $\tilde{A}_0$ is 4ε-close to some polynomial $\widehat{A}_0$ of degree md such that $C(\widehat{A}_0|_I) = 1$. Since w is δ-far from any satisfying assignment, the assignment given by $\widehat{A}_0|_I$ must be at least δ-far from w. Let $B \subset \mathbb{F}^m$ denote the set of locations in I where the assignment given by $\widehat{A}_0$ disagrees with w (i.e., $B = \{x \in I | \widehat{A}_0(x)$ disagrees with w at $x\}$). Hence, $|B|/|I| \geq \delta$. Since $|I| = k \geq n/5$, we have $|B| \geq \delta n/5$. Consider the following two events.

Event I. $\tilde{A}_0|_{\mathcal{L}}$ is 5ε-far from $\widehat{A}_0|_{\mathcal{L}}$.

By the sampling lemma (Lemma B.3) with $\mu = 4\varepsilon$ and $\zeta = \varepsilon$, this event occurs with probability at most $\left(\frac{1}{|\mathbb{F}|} + \lambda\right) \cdot \frac{4\varepsilon}{\varepsilon^2} \leq \frac{1}{4}$ since $|\mathbb{F}|, \frac{1}{\lambda} \geq 32/\varepsilon$.

Event II. $B \cap \mathcal{L} = \emptyset$.

Again by the sampling lemma (Lemma B.3) with $\mu = \zeta = \frac{|B|}{|\mathbb{F}^m|}$, this event occurs with probability at most $\left(\frac{1}{|\mathbb{F}|} + \lambda\right) \cdot \frac{|\mathbb{F}^m|}{|B|} = \left(\frac{1}{|\mathbb{F}|} + \lambda\right) \cdot \frac{5|\mathbb{F}^m|}{\delta n} \leq \frac{1}{4}$, where the last inequality follows because $n \geq 8000|\mathbb{F}|^{m-1}/\delta^3 \geq 40|\mathbb{F}|^{m-1}/\delta$ and $\lambda \leq \delta/(40(c_F m^2)^m)$.

Suppose Event I does not occur. Then, if $\widehat{A}_0|_{\mathcal{L}} \neq \tilde{A}_0|_{\mathcal{L}}$, PROXIMITY TEST rejects since then $\tilde{A}_0|_{\mathcal{L}}$ cannot be a polynomial of degree at most d, as it is 5ε-close to the polynomial $\widehat{A}_0$ and hence cannot be closer to any other polynomial (as $5\varepsilon \leq \frac{1}{2}(1 - \frac{d}{|\mathbb{F}|}) = \frac{1}{2}(1 - \frac{1}{c_F})$). Now if $\widehat{A}_0|_{\mathcal{L}} = \tilde{A}_0|_{\mathcal{L}}$ and Event II does not occur, then PROXIMITY TEST detects a mismatch between the input oracle W and $\tilde{A}_0|_{\mathcal{L}}$. Hence, if both Event I and Event II do not occur, then the test rejects.

Thus, the probability of the test accepting in this case is at most the probability of either Event I or Event II occurring, which is at most $1/2$. Thus, the probability that the verifier accepts is at most $\max\left\{1 - \varepsilon, \frac{1}{2}\right\} = 1 - \varepsilon$. This completes the proof of the lemma. ∎

Proof of Theorem 7.1. Theorem 7.1 is proved using PCPP VERIFIER defined in this section, setting $\lambda = \min\{1/c \log n, \delta^3/m^{cm}\}$. The randomness and decision (resp., query) complexities follow from Proposition 7.4. The only fact to be verified is the soundness of the verifier. By the hypothesis of Theorem 7.1, $n^{1/m} \geq m^{cm}/\delta^3$ for a suitably large constant c. This implies that $n^{1/m} \geq 8000(c_F m^2)^{m-1}/\delta^3$ or, equivalently, $n \geq 8000|\mathbb{F}|^{m-1}/\delta^3$. Hence, Lemma 7.5 applies and we have that the verifier has soundness error $1 - \varepsilon$ for proximity parameter δ. This proves Theorem 7.1. ∎

7.2. The ALMSS-type PCPP (Theorem 7.2). We now turn to designing a PCPP that proves Theorem 7.2. We call this ALMSS PCPP VERIFIER as it has parameters similar to the "parallelized" PCPs of [ALM+98]. ALMSS PCPP VERIFIER is identical to PCPP VERIFIER of Theorem 7.1 except for the fact that it has a slightly different proximity test. All other details remain the same.

ALMSS PCPP VERIFIER$_\delta^{W;\ \tilde{A}_i, P_{i,j}; i=0,\ldots,l-1; j=0,\ldots,m}(C)$.

1. Set $m = \log n / \log\log n$ and $\lambda = 1/c \log n$.

2. Run PCP VERIFIER$_{m,\lambda}^{W;\,\tilde{A}_i,P_{i,j}}(C)$ and reject if it rejects.
3. ALMSS PROXIMITY TEST.
   Choose a random position $i \stackrel{R}{\leftarrow} \{1,\ldots,k\}$ in the input and a random direction $y \in \mathbb{F}^m$. Let $x \in I$ be the point corresponding to i in H^m. Let $\mathcal{L}$ be the line through x in the direction y. Query oracle $\tilde{A}_0$ on all points along the line $\mathcal{L}$ and reject if the restriction $\tilde{A}_0$ to $\mathcal{L}$ is not a polynomial of degree at most $d = m \cdot |H|$. Query the input oracle W at location i and reject if $W[i] \neq \tilde{A}_0(x)$.

Unlike PCPP VERIFIER, we will not calculate the randomness used by this verifier exactly. An upper bound within a constant factor suffices for our purposes. The extra randomness used by ALMSS PROXIMITY TEST is $\log k + m \log |\mathbb{F}|$ (i.e., the randomness required to choose a random index in $\{1,\ldots,k\}$ and a random direction in $\mathbb{F}^m$). For our choice of m and δ, the randomness of PCP VERIFIER is at most $O(\log n)$ (see the analysis preceding Proposition 6.17). Hence, the total randomness of ALMSS PCPP VERIFIER is at most $O(\log n)$. The query and decision complexities are at most a constant times that of PCP VERIFIER which is turn is upper-bounded by $\operatorname{poly}\log n$. Summarizing the above observations for future reference, we have the following proposition.

PROPOSITION 7.6. *The randomness and decision complexities of* ALMSS PCPP VERIFIER *are* $O(\log n)$ *and* $\operatorname{poly}\log n$*, respectively.*

The soundness of the verifier is given by the following lemma.

LEMMA 7.7. *For all* $\delta \in (0,1)$*, the following holds. If the input* w *given by the input oracle* $W : [k] \to \{0,1\}$ *is* δ*-far from satisfying the circuit, then for any proof oracle the verifier rejects* W *with probability* $\Omega(\delta)$.

Proof. Let ε_0 be the constant that appears in Lemma 6.18.

Case (i). $\tilde{A}_0$ is not $4\varepsilon_0$-close to any polynomial $\widehat{A}_0$ of degree md such that $C(\widehat{A}_0|_I) = 1$. Then by Lemma 6.18, we conclude that the verifier rejects with probability at least ε_0.

Case (ii). $\tilde{A}_0$ is $4\varepsilon_0$-close to some polynomial $\widehat{A}_0$ of degree md such that $C(\widehat{A}_0|_I) = 1$. Since w is δ-far from any satisfying assignment, the assignment given by $\widehat{A}_0|_I$ must be δ-far from w. With probability greater than δ over the choice of $i \in \{1,\ldots,k\}$ (and the corresponding point $x \in I$ in H^m), we have $W[i] \neq \widehat{A}_0(x)$. If this occurs, the only way the verifier can accept is if $\tilde{A}_0|_{\mathcal{L}}$ is a degree md polynomial other than $\widehat{A}_0|_{\mathcal{L}}$. We will show that for any fixed point x in $\mathbb{F}^m$, with probability at least $1 - 16\varepsilon_0$ over the choice of random line $\mathcal{L}$ through x, $\tilde{A}_0|_{\mathcal{L}}$ cannot be a degree md polynomial different from $\widehat{A}_0|_{\mathcal{L}}$. We can then conclude that the verifier rejects with probability at least $\delta \cdot (1 - 16\varepsilon_0) = \Omega(\delta)$. Since $\mathcal{L}$ is a random line through x, every point on $\mathcal{L}$ other than x is a uniformly random point in $\mathbb{F}^m$. Recall that $\tilde{A}_0$ and $\widehat{A}_0$ are $4\varepsilon_0$-close. By a Markov argument it follows that for every fixed value of x and a random line $\mathcal{L}$ through x, with probability at least $1 - 16\varepsilon_0$, $\tilde{A}|_{\mathcal{L}\setminus\{x\}}$ and $\widehat{A}|_{\mathcal{L}\setminus\{x\}}$ are at least 1/4-close. This implies that $\tilde{A}|_{\mathcal{L}}$ cannot be a polynomial of degree md other than $\widehat{A}|_{\mathcal{L}}$ (since two distinct polynomials agree in at most md points, and $(md-1)/|\mathbb{F}| < 1/4$).

In either case, ALMSS PCPP VERIFIER rejects with probability at least $\min\{\varepsilon_0, \Omega(\delta)\} = \Omega(\delta)$. □

Conclusion. Theorem 7.2 follows from Proposition 7.6 and Lemma 7.7.

8. A randomness-efficient robust PCPP. In this section, we modify the PCPP for CKTVAL constructed in section 7 to design a robust PCPP, while essentially maintaining all complexities. Recall the definition of robustness: If the input oracle

W is δ-far from a satisfying assignment, then a "regular" PCPP verifier rejects the input for most choices of its random coins; that is, it observes an inconsistency in the (input and) proof. In contrast, for most choices of its random coins, a *robust* PCPP verifier not only notices an inconsistency in the (input and) proof but also observes that a considerable portion of the (input and) proof read by it has to be modified to remove this inconsistency.

We construct two robust PCPPs which are robust analogues of the two PCPPs presented in section 7. The first robust PCPP is the main construct (claimed in Theorem 3.1), which is the robust analogue of PCPP Verifier. The second robust PCPP is an ALMSS-type robust PCPP (claimed in Theorem 3.2), which is the robust analogue of ALMSS PCPP Verifier. Thus, we prove Theorems 3.1 and 3.2.

Overview of the proofs of Theorem 3.1 *and* 3.2. We "robustify" the PCPP verifier in three steps. Recall that a single execution of the verifier actually involves several tests (in fact $lm+2l$ Low-Degree Tests, l Edge-Consistency Tests, lm Zero-Propagation Tests, l Identity Tests, and a single proximity test (either Proximity Test or ALMSS Proximity Test, as the case may be)). In the first step (section 8.1), we observe that each of these tests is robust individually. In the second step (section 8.2), we perform a "bundling" of the queries so that a certain set of queries can always be asked together. Indeed, bundling is analogous to "parallelization" except that it does *not* involve any increase in the randomness complexity (unlike parallelization, which introduces such an increase, which although small is too big for our purposes). We stress that bundling is tailored to the specific tests, in contrast to parallelization which is generic. The aforementioned bundling achieves robustness, albeit over a much a larger alphabet. In the final step (section 8.3), we use a good error-correcting code to transform the "bundles" into regular bit-queries such that robustness over the binary alphabet is achieved.

8.1. Robustness of individual tests. For each possible random string R, PCPP Verifier (resp., ALMSS PCPP Verifier) performs several tests. More precisely, it performs $l(m+2)$ Low-Degree Tests, l Edge-Consistency Tests, lm Zero-Propagation Tests, l Identity Tests, and a single Proximity Test (resp., ALMSS Proximity Test). In this section, we prove that each of these tests is robust individually. In other words, we show that when one of these tests fails, it fails in a "robust" manner; that is, a considerable portion of the input read by the test has to be modified for the test to pass.

First, we introduce some notation. We view functions $g, g' : \mathbb{F}^m \to \mathbb{F}$ as strings of length $|\mathbb{F}|^m$ over the alphabet $\mathbb{F}$, so their relative Hamming distance $\Delta(g, g')$ is simply $\Pr_x[g(x) \neq g'(x)]$. As before, let $I \subseteq H^m \subset \mathbb{F}^m$ be the set of locations in $\mathbb{F}^m$ that contains the assignment.

Let $0 < \varepsilon < 1$ be a small constant to be specified later. As before, for $i = 0, \ldots, l-1$, $j = 0, \ldots, m$ and $b \in \{0,1\}$, let $\widehat{A}_i$ (resp., $\widehat{P}^{(b)}_{i,j}$) be the closest polynomials of degree md (resp., κmd) to $\tilde{A}_i$ and $P_{i,j}$, respectively. (If there is more than one polynomial, choose one arbitrarily.) The proof of the soundness of the PCPP verifiers, PCPP Verifier and ALMSS PCPP Verifier (see section 6 and 7), was along the following lines: If the input oracle $W : [k] \to \{0,1\}$ is δ-far from satisfying the circuit, then one of the following must happen (changing ε by a factor of 2):

1. There exists $i \in \{0, \ldots, l-1\}$ such that $\tilde{A}_i$ is 8ε-far from every degree md polynomial or there exists $i \in \{0, \ldots, l-1\}$, $j \in \{0, \ldots, m\}$, and $b \in \{0,1\}$ such that $P^{(b)}_{i,j}$ is 8ε-far from every degree κmd polynomial. In this case,

LOW-DEGREE TEST detects the error with probability at least 2ε.

2. There exists $i \in \{0, \ldots, l-1\}$ and $b \in \{0,1\}$, such that $\Delta(P_{i,m}^{(b)}, \widehat{P}_{i,m}^{(b)}) \leq 8\varepsilon$ and $\widehat{P}_{i,m} \not\equiv 0$. In this case, IDENTITY TEST detects the error with probability at least $1 - 10\varepsilon$.
3. There exists $i \in \{0, \ldots, l-1\}$, $j \in \{1, \ldots, m\}$, and $b \in \{0,1\}$ such that
$$\Delta(P_{i,j}^{(b)}, \widehat{P}_{i,j}^{(b)}) \leq 8\varepsilon, \Delta(P_{i,j-1}^{(b)}, \widehat{P}_{i,j-1}^{(b)}) \leq 8\varepsilon,$$
$$\text{and } \widehat{P}_{i,j}^{(b)}(\ldots, x_j, \ldots) \not\equiv \sum_{k=0}^{|H|-1} \widehat{P}_{i,j-1}^{(b)}(\ldots, h_k, \ldots) x_j^k.$$
In this case, ZERO-PROPAGATION TEST detects the error with probability at least $1 - 20\varepsilon$.
4. There exists $i \in \{0, \ldots, l-1\}$ such that
$$\Delta(P_{i,0}^{(0)}, \widehat{P}_{i,0}^{(0)}) \leq 8\varepsilon, \Delta(P_{i,0}^{(1)}, \widehat{P}_{i,0}^{(1)}) \leq 8\varepsilon, \Delta(\tilde{A}_i, \widehat{A}_i) \leq 8\varepsilon, \Delta(\tilde{A}_{i+1}, \widehat{A}_{i+1}) \leq 8\varepsilon,$$
$$\text{and } \widehat{P}_{i,0}(x) \not\equiv \psi'(\tilde{T}_i(x), \widehat{A}_i(x), \widehat{A}_{i+1}(\tilde{\Gamma}_{i,0}(x)), \widehat{A}_{i+1}(\tilde{\Gamma}_{i,1}(x))).$$
In this case, EDGE-CONSISTENCY TEST detects the error with probability at least $1 - 42\varepsilon$.
5. $\Delta(\tilde{A}_0, \widehat{A}_0) \leq 8\varepsilon$, but W and $\widehat{A}_0|_I$ disagree on at least δ fraction of the points. In this case, PROXIMITY TEST (or ALMSS PROXIMITY TEST, as the case may be) detects the error with probability at least 2ε (or $\Omega(\delta)$ in the case of ALMSS PROXIMITY TEST).

Claims 8.1–8.6 below strengthen the above analysis and show that one of the tests not only detects the error, but a significant portion of the input read by that test needs to be modified in order to make the test accept. More formally, recall that each of our tests T (randomly) generates a pair (I, D), where I is a set of queries to make to its oracle and D is the predicate to apply to the answers. For such a pair $(I, D) \leftarrow T$ and an oracle π, we define the *distance of $\pi|_I$ to T* to be the relative Hamming distance between $\pi|_I$ and the nearest satisfying assignment of D. Similarly, we say that π has *expected distance ρ from satisfying T* if the expectation of the distance of $\pi|_I$ to T over $(I, D) \overset{R}{\leftarrow} T$ equals ρ.

We then have the following claims about the robustness of the individual tests.

The robustness of LOW-DEGREE TEST can be easily inferred from the analysis of the λ-biased low-degree test due to Ben-Sasson et al. [BSVW03] as shown below.

CLAIM 8.1. *The following holds for all sufficiently small $\varepsilon > 0$. If $A : \mathbb{F}^m \to \mathbb{F}$ (resp., $P : \mathbb{F}^m \to \mathbb{F}$) is 8ε-far from every polynomial of degree md (resp., degree κmd), then the expected distance of A (resp., P) from satisfying* LOW-DEGREE TEST *with degree parameter d (resp., κd) is at least 2ε.*

Proof. Recall that LOW-DEGREE TEST chooses a random canonical line $\mathcal{L}$ and checks if $A|_{\mathcal{L}}$ is a univariate polynomial of degree d. For each canonical line $\mathcal{L}$, define $A_{\text{lines}}(\mathcal{L})$ to be the degree d univariate polynomial mapping $\mathcal{L} \to \mathbb{F}$ having maximum agreement with A on $\mathcal{L}$, breaking ties arbitrarily. The distance of $A|_{\mathcal{L}}$ to satisfying LOW-DEGREE TEST is precisely $\Delta(A|_{\mathcal{L}}, A_{\text{lines}}(\mathcal{L}))$.

The low-degree test LDT of Ben-Sasson et al. [BSVW03] works as follows (see Appendix B for more details): The test LDT has oracle access to a points-oracle $f : \mathbb{F}^m \to \mathbb{F}$ and a lines-oracle g. It chooses a random canonical line $\mathcal{L}$ using the λ-biased set, queries the lines-oracle g on the line $\mathcal{L}$, and queries the points-oracle f on a random point x on $\mathcal{L}$. It accepts iff $g(\mathcal{L})$ agrees with f at x.

By inspection, the probability that $\mathrm{LDT}^{A,A_{\mathrm{lines}}}$ rejects the points-oracle A and lines-oracle A_{lines} as defined above equals $\mathrm{E}_{\mathcal{L}}[\Delta(A|_{\mathcal{L}}, A_{\mathrm{lines}}(\mathcal{L}))]$. By Theorem B.4, if A is 8ε-far from every degree md polynomial, then $\mathrm{LDT}^{A,A_{\mathrm{lines}}}$ rejects with probability at least 2ε (for sufficiently small ε). (Recall that our parameters satisfy the conditions of Theorem B.4 for sufficiently large choices of the constants c and c_F.) Thus, A has expected distance 2ε from satisfying our Low-Degree Test, as desired. □

The intuition behind the proofs of robustness of Identity Test, Zero-Propagation Test, and Edge-Consistency Test is as follows. The key point to be noted is that the checks made by each of these tests are in the form of polynomial identities. Hence, if the functions read by these tests are close to being polynomials, then it follows from the Schwartz–Zippel lemma that the inputs read by these tests either satisfy these polynomial identities or are in fact far from satisfying them. We formalize this intuition and prove the robustness of Identity Test, Edge-Consistency Test, and Zero-Propagation Test in Claims 8.2, 8.3, and 8.4, respectively.

Claim 8.2. *The following holds for all sufficiently small $\varepsilon > 0$. If for some $i = 0, \ldots, l-1$ and $b \in \{0,1\}$, $\Delta(P_{i,m}^{(b)}, \widehat{P}_{i,m}^{(b)}) \le 8\varepsilon$ and $\widehat{P}_{i,m}^{(b)}(\cdot) \not\equiv 0$, then $P_{i,m}$ has expected distance at least $1 - 9\varepsilon$ from satisfying* Identity Test.

Proof. The expected distance of $P_{i,m}$ from satisfying Identity Test equals

$$\begin{aligned}
\mathrm{E}_{V_\eta}[\Delta(P_{i,m}|_{V_\eta}, 0)] &= \Delta(P_{i,m}, 0) && \text{[since } \{V_\eta\} \text{ is a partition]} \\
&\ge \Delta(\widehat{P}_{i,m}, 0) - \Delta(P_{i,m}, \widehat{P}_{i,m}) \\
&\ge \left(1 - \tfrac{\kappa md}{|\mathbb{F}|}\right) - 8\varepsilon && \text{[by the Schwartz–Zippel lemma} \\
&&& \text{and hypothesis]} \\
&\ge 1 - 9\varepsilon. \qquad \square
\end{aligned}$$

Claim 8.3. *The following holds for all sufficiently small $\varepsilon > 0$. Suppose for some $i = 0, \ldots, l-1$, we have $\Delta(P_{i,0}^{(0)}, \widehat{P}_{i,0}^{(0)}) \le 8\varepsilon$, $\Delta(P_{i,0}^{(1)}, \widehat{P}_{i,0}^{(1)}) \le 8\varepsilon$, $\Delta(\tilde{A}_i, \widehat{A}_i) \le 8\varepsilon$, $\Delta(\tilde{A}_{i+1}, \widehat{A}_{i+1}) \le 8\varepsilon$, and $\widehat{P}_{i,0}(\cdot) \not\equiv \psi'(\tilde{T}_i(\cdot), \widehat{A}_i(\cdot), \widehat{A}_{i+1}(\tilde{\Gamma}_{i,0}(\cdot)), \widehat{A}_{i+1}(\tilde{\Gamma}_{i,1}(\cdot)))$. Then $\{P_{i,0}(\cdot), A_i(\cdot), A_{i+1}(\tilde{\Gamma}_{i,0}(\cdot))A_{i+1}(\tilde{\Gamma}_{i,1}(\cdot))\}$ has expected distance at least $(1 - 41\varepsilon)/5$ from satisfying* Edge-Consistency Test.

Proof. Note that the distance of $\{P_{i,0}(\cdot), A_i(\cdot), A_{i+1}(\tilde{\Gamma}_{i,0}(\cdot))A_{i+1}(\tilde{\Gamma}_{i,1}(\cdot))\}|_{U_\eta}$ from satisfying Edge-Consistency Test is at least $1/5$ times the distance of $P_{i,0}(\cdot)|_{U_\eta}$ to the function $\psi'(\tilde{T}_i(\cdot), A_i(\cdot), A_{i+1}(\tilde{\Gamma}_{i,0}(\cdot)), A_{i+1}(\tilde{\Gamma}_{i,1}(\cdot)))|_{U_\eta}$ (since for each point $x \in U_\eta$, where the latter two functions disagree, at least one of $P_{i,0}, A_i, A_{i+1} \circ \tilde{\Gamma}_{i,0}, A_{i+1} \circ \tilde{\Gamma}_{i,1}$ needs to be changed at x to make the test accept). As in the proof of Claim 8.2, we have

$$\begin{aligned}
&\mathrm{E}_{U_\eta}[\Delta(P_{i,0}(\cdot)|_{U_\eta}, \psi'(\tilde{T}_i(\cdot), A_i(\cdot), A_{i+1}(\tilde{\Gamma}_{i,0}(\cdot)), A_{i+1}(\tilde{\Gamma}_{i,1}(\cdot)))|_{U_\eta})] \\
&\qquad \ge \left(1 - \frac{\kappa md}{|\mathbb{F}|}\right) - 5 \cdot 8\varepsilon \ge 1 - 41\varepsilon,
\end{aligned}$$

where the $(1 - \kappa md/|\mathbb{F}|)$ term corresponds to the distance if we replace all five functions with their corrected polynomials (e.g., $\widehat{P}_{i,0}$, $\widehat{A}_i$, $\widehat{A}_{i+1} \circ \tilde{\Gamma}_{i,0}$, $\widehat{A}_{i+1} \circ \tilde{\Gamma}_{i,1}$) and the $-5 \cdot 8\varepsilon$ accounts for the distance between each of the five functions and their corrected polynomials. Thus, the overall expected distance to satisfying Edge-Consistency Test is at least $(1 - 41\varepsilon)/5$. □

Claim 8.4. *The following holds for all sufficiently small $\varepsilon > 0$. Suppose for some $i = 0, \ldots, l-1$, $j = 1, \ldots, m$, and $b \in \{0,1\}$, we have $\Delta(P_{i,j}^{(b)}, \widehat{P}_{i,j}^{(b)}) \le$*

8ε, $\Delta(P^{(b)}_{i,j-1}, \widehat{P}^{(b)}_{i,j-1}) \leq 8\varepsilon$, *and* $\widehat{P}^{(b)}_{i,j}(\dots, x_j, \dots) \not\equiv \sum_{k=0}^{|H|-1} \widehat{P}^{(b)}_{i,j-1}(\dots, h_k, \dots) x_j^k$. *Then* $(P^{(b)}_{i,j}, P^{(b)}_{i,j-1})$ *has expected distance at least* $(1-19\varepsilon)/2$ *from satisfying* Zero-Propagation Test.

Proof. Suppose that $\mathcal{L}$ is a jth axis-parallel line such that

$$\widehat{P}^{(b)}_{i,j}(\dots, x_j, \dots)|_{\mathcal{L}} \not\equiv \sum_{k=0}^{|H|-1} \widehat{P}^{(b)}_{i,j-1}(\dots, h_k, \dots) x_j^k|_{\mathcal{L}}.$$

Then in order for Zero-Propagation Test to accept, either $P^{(b)}_{i,j}|_{\mathcal{L}}$ must be modified to equal a degree κd polynomial other than $\widehat{P}^{(b)}_{i,j-1}(\dots, x_j, \dots)|_{\mathcal{L}}$, or $P^{(b)}_{i,j-1}|_{\mathcal{L}}$ must be modified to equal a degree κd polynomial other than $\widehat{P}^{(b)}_{i,j-1}(\dots, x_j, \dots)|_{\mathcal{L}}$. (Recall that Zero-Propagation Test checks that the said restrictions are in fact polynomials of degree κd.) This would require modifying $P^{(b)}_{i,j}|_{\mathcal{L}}$ (resp., $P^{(b)}_{i,j-1}|_{\mathcal{L}}$) in at least a $1 - \kappa d/|\mathbb{F}| - \Delta(P^{(b)}_{i,j}|_{\mathcal{L}}, \widehat{P}^{(b)}_{i,j}|_{\mathcal{L}})$ fraction (resp., a $1 - \kappa d/|\mathbb{F}| - \Delta(P^{(b)}_{i,j-1}|_{\mathcal{L}}, \widehat{P}^{(b)}_{i,j-1}|_{\mathcal{L}})$ fraction) of points. This implies that the pair $(P^{(b)}_{i,j}|_{\mathcal{L}}, P^{(b)}_{i,j-1}|_{\mathcal{L}})$ would have to be modified in at least a

$$\frac{1}{2} \cdot \left(1 - \frac{\kappa d}{|\mathbb{F}|} - \Delta(P^{(b)}_{i,j}|_{\mathcal{L}}, \widehat{P}^{(b)}_{i,j}|_{\mathcal{L}}) - \Delta(P^{(b)}_{i,j-1}|_{\mathcal{L}}, \widehat{P}^{(b)}_{i,j-1}|_{\mathcal{L}})\right)$$

fraction of points.

Thus the expected distance of $(P^{(b)}_{i,j}, P^{(b)}_{i,j-1})$ from satisfying Zero-Propagation Test is at least

$$\begin{aligned}
&\frac{1}{2} \cdot \mathrm{E}_{\mathcal{L}} \left[1 - \frac{\kappa d}{|\mathbb{F}|} - \Delta(P^{(b)}_{i,j}|_{\mathcal{L}}, \widehat{P}^{(b)}_{i,j}|_{\mathcal{L}}) - \Delta(P^{(b)}_{i,j-1}|_{\mathcal{L}}, \widehat{P}^{(b)}_{i,j-1}|_{\mathcal{L}})\right] \\
&\quad - \Pr_{\mathcal{L}} \left[\widehat{P}^{(b)}_{i,j}(\dots, x_j, \dots)|_{\mathcal{L}} \equiv \sum_{k=0}^{|H|-1} \widehat{P}^{(b)}_{i,j-1}(\dots, h_k, \dots) x_j^k|_{\mathcal{L}}\right] \\
&\qquad\qquad \geq \frac{1}{2}(1 - \varepsilon - 8\varepsilon - 8\varepsilon) - \frac{\kappa d}{|\mathbb{F}|} \\
&\qquad\qquad \geq \frac{1}{2}(1 - 19\varepsilon). \qquad \square
\end{aligned}$$

We are now left with analyzing the robustness of the proximity tests (Proximity Test and ALMSS Proximity Test). Note that the input for either of these proximity tests comes in two parts: (a) the restriction of A_0 to the line $\mathcal{L}$ and (b) the input W restricted to the line $\mathcal{L}$ (or to a point on $\mathcal{L}$). Thus, the robustness of these tests refers to both parts (i.e., parts of each of the two oracles), and it is beneficial to decouple the corresponding robustness properties. We note that the robustness of Proximity Test is proved by repeated applications of the sampling lemma (Lemma B.3), while the robustness of ALMSS Proximity Test follows by a simple Markov argument.

Let $B \subset \mathbb{F}^m$ denote the set of locations in I, where the assignment given by $\widehat{A}_0$ disagrees with W (i.e., $B = \{x \in I | \widehat{A}_0(x)$ disagrees with W at $x\}$). Recall that $|I| = k \geq n/5$.

Claim 8.5. *There exists a constant c and a constant $\varepsilon > 0$ such that for all $m, \lambda, \delta, \delta'$ satisfying $n \geq 8000|\mathbb{F}|^{m-1}/\delta^3$, $\lambda \leq 1/c \log n$, $\lambda \leq \delta^3/m^{cm}$, $\delta' > \delta$, the following holds. Suppose $\Delta(\tilde{A}_0, \widehat{A}_0) \leq 1/4$ and the input oracle W is δ'-far from*

$\widehat{A}_0|_I$ (i.e., $|B|/|L| \geq \delta'$); then with probability at least $1 - \delta/4$ (over the choice of the canonical line $\mathcal{L}$) either at least an ε-fraction of $A_0|_{\mathcal{L}}$ or at least a $(\delta' - \delta/4)$-fraction of $W|_{\mathcal{L}}$ needs to be changed to make PROXIMITY TEST *accept.*

This claim is the robust analogue of Lemma 7.5. Observe that the robustness of the verifier is expressed separately for the proof and input oracles. As expected, the robustness of the input oracle depends on the proximity parameter δ', while that of the proof oracle is independent of δ'.

Proof. Consider the following three events.

Event 1. $\Delta(\tilde{A}_0|_{\mathcal{L}}, \widehat{A}_0|_{\mathcal{L}}) \geq 1/3$.

By the sampling lemma (Lemma B.3) with $\mu = 1/4$ and $\zeta = 1/12$, this event occurs with probability at most $((1/|\mathbb{F}|) + \lambda) \cdot (1/4)/(1/12)^2 \leq (\delta/12)$ since $|\mathbb{F}| \geq (8000|\mathbb{F}|^m)/(\delta^3 n) > (12^3/2)/\delta$ and $\lambda < 2\delta/12^3$.

Event 2. $\frac{|I \cap \mathcal{L}|}{|\mathcal{L}|} > \left(1 + \frac{\delta}{8}\right) \cdot \frac{|I|}{|\mathbb{F}^m|}$.

Again by the sampling lemma (Lemma B.3) with $\mu = |I|/|\mathbb{F}^m| \geq \frac{n}{5|\mathbb{F}|^m}$ and $\zeta = \frac{\delta\mu}{8}$, this event occurs with probability at most

$$\left(\frac{1}{|\mathbb{F}|} + \lambda\right) \cdot \frac{8^2}{\delta^2 \mu} = \left(\frac{1}{|\mathbb{F}|} + \lambda\right) \cdot \frac{320|\mathbb{F}|^m}{\delta^2 n} \leq \frac{\delta}{12},$$

where the last inequality follows from the fact that $n \geq 24 \cdot 320 \cdot |\mathbb{F}|^{m-1}/\delta^3$ and $\lambda \leq \delta^3/(24 \cdot 320(c_F m^2)^m)$.

Event 3. $\frac{|B \cap \mathcal{L}|}{|\mathcal{L}|} < \frac{|B|}{|\mathbb{F}^m|} - \frac{\delta}{8} \cdot \frac{|I|}{|\mathbb{F}^m|}$.

Again by the sampling lemma (Lemma B.3) with $\mu = |B|/|\mathbb{F}^m| = \frac{\delta' n}{5|\mathbb{F}|^m}$ and $\zeta = \frac{\delta n}{40|\mathbb{F}|^m}$, this event occurs with probability at most

$$\left(\frac{1}{|\mathbb{F}|} + \lambda\right) \cdot \frac{\mu}{\zeta^2} \leq \left(\frac{1}{|\mathbb{F}|} + \lambda\right) \cdot \frac{320|\mathbb{F}|^m}{\delta^2 n} \leq \frac{\delta}{12}.$$

Hence, the probability that at least one of the three events occurs is at most $\delta/4$.

Now, suppose none of the three events occur. We then get that

$$\frac{|B \cap \mathcal{L}|}{|I \cap \mathcal{L}|} \geq \frac{|B| - \delta|I|/8}{(1 + \delta/8)|I|} = \frac{\delta' - \delta/8}{1 + \delta/8} \geq \delta' - \frac{\delta}{4}.$$

Now for PROXIMITY TEST to accept the pair $(\tilde{A}_0|_{\mathcal{L}}, W \cap \mathcal{L})$, either we must change $\tilde{A}_0|_{\mathcal{L}}$ to a polynomial other than $\widehat{A}_0|_{\mathcal{L}}$ or correct the input for all $x \in B \cap \mathcal{L}$. The former requires us to change at least a $(1 - (d/|\mathbb{F}|) - 1/3) \geq 1/2$-fraction of the points of $A_0|_{\mathcal{L}}$ while the latter requires us to change at least a $\delta' - \delta/4$-fraction of the input read (i.e., the input oracle W restricted to the line $\mathcal{L}$). This proves the claim. ∎

We now analyze the robustness of ALMSS PROXIMITY TEST.

CLAIM 8.6. *There exists a constant $\varepsilon_0 > 0$ such that for all $\delta \in (0, 1)$, the following holds. Suppose $\Delta(\tilde{A}_0, \widehat{A}_0) \leq 4\varepsilon_0$ and the input oracle W is δ-far from $\widehat{A}_0|_I$* (i.e., $|B|/|L| \geq \delta$)*; then with probability at least $\Omega(\delta)$ (over the choice of index i and direction y), either at least a 1/2-fraction of $A_0|_{\mathcal{L}}$ or $W[i]$ (i.e., the single symbol of the input oracle read by the verifier) needs to be changed to make* ALMSS PROXIMITY TEST *accept.*

This claim is the robust analogue of Lemma 7.7. As before, the robustness of the verifier is expressed separately for the proof and input oracles.

Proof. Since w is δ-far from any satisfying assignment, the assignment given by $\widehat{A}_0|_I$ must be δ-far from w. Thus, with probability greater than δ over the choice of

$i \in \{1, \ldots, k\}$ (and the corresponding point $x \in I$), we have $W[i] \neq \widehat{A}_0(x)$. If this occurs, the only way to make the verifier accept is to either change $W[i]$ or change $\tilde{A}_0|_{\mathcal{L}}$ to a degree md polynomial other than $\widehat{A}_0|_{\mathcal{L}}$. As in the proof of Lemma 7.7, for any fixed x, with probability at least $1 - 16\varepsilon_0$ (over the choice of the random direction y), $\tilde{A}_0|_{\mathcal{L}\setminus\{x\}}$ and $\widehat{A}_0|_{\mathcal{L}\setminus\{x\}}$ have distance at most $1/4$, and hence $\tilde{A}_0|_{\mathcal{L}}$ would have to be changed in at least $1 - ((md-1)/|\mathbb{F}|) - 1/4 \geq 1/2$ points to be a degree md polynomial other than $\widehat{A}_0|_{\mathcal{L}}$. Thus, with probability at least $\delta(1 - 16\varepsilon_0) = \Omega(\delta)$, either $W[i]$ would have to change or at least half of $\tilde{A}_0|_{\mathcal{L}}$ would have to change to make the verifier accept. ∎

8.2. Bundling. In section 8.1, we showed that each of the tests performed by the PCPP verifier is individually robust. However, we need to show that the conjunction of all these tests is also robust. This is not true for the PCPP verifier in its present form for the following reason: Suppose the input oracle W is δ-far from satisfying the circuit. We then know that one of the tests detects this fact with nonnegligible probability. Moreover, as seen in section 8.1, this test is robust. However, since this test is only one of the $O(lm)$ tests being performed by the verifier, the oracle bits read by this test comprise a small fraction of the total query complexity of the verifier. For instance, the number of bits read by a single Low-Degree Test is about $1/lm$ times the query complexity. This causes the robustness of the verifier to drop by a factor of at least lm. Note that the issue here is not the fact that the verifier performs different types of tests (i.e., Low-Degree Test, Identity Test, Zero-Propagation Test, etc.) but rather that it performs many instances of each test and that the soundness analysis guarantees only that one of these test instances rejects (robustly). This is not sufficient to make the verifier robust.

For this purpose, we bundle the various functions in the proof oracle so that the inputs required for the several test instances can be read together. This maintains the robustness of the individual tests, albeit over a larger alphabet. To understand this bundling, let us assume for the present that the only type of tests that the verifier performs is Low-Degree Test. There exists a natural bundling in this case. Instead of $l(m+2)$ different oracles $\{\tilde{A}_i\}$ and $\{P_{i,j}\}$, we have one oracle Π which bundles together the data of all these oracles. The oracle $\Pi : \mathbb{F}^m \to \mathbb{F}^{l\cdot(2m+3)}$ is assumed to satisfy $\Pi(x) = (\tilde{A}_0(x), \ldots, \tilde{A}_{l-1}(x), P_{0,0}(x), \ldots, P_{l-1,m}(x))$ for all $x \in \mathbb{F}^m$. It can now be easily checked that over this proof oracle, the conjunction of all Low-Degree Tests is robust (over alphabet $\mathbb{F}^{l\cdot(2m+3)}$) with the same soundness and robustness parameters as a single Low-Degree Test (over alphabet $\mathbb{F}$). However, this natural bundling does not lend itself to the other tests performed by the PCPP verifier—namely, Zero-Propagation Test, and Edge-Consistency Test—because the l executions of these tests each have different query patterns (i.e., we cannot execute all of these tests by querying the same set of points of Π). Next, we provide an alternate bundling and massage our verifier slightly to work with this bundling.

To find a suitable bundling, we examine the query patterns of Zero-Propagation Test and Edge-Consistency Test more closely. The (i,j)th Zero-Propagation Test queries $P_{i,j-1}$ and $P_{i,j}$ on a random jth axis-parallel line. Also the ith Edge-Consistency Test queries $P_{i,0}$, $\tilde{A}_i$ for all points $x \in U_\eta$, and $\tilde{A}_{i+1}$ for all points $x \in \tilde{\Gamma}_{i,0}(U_\eta) \cup \tilde{\Gamma}_{i,1}(U_\eta)$ for U_η being a random subset of an arbitrary partition of F^m. The key observation is that the neighborhood functions $\tilde{\Gamma}_{i,0}$ and $\tilde{\Gamma}_{i,1}$ take any ith axis-parallel line to itself. Thus, if we choose the partition U_η to consist of ith axis-parallel lines, then the ith Edge-Consistency Test is also making queries entirely along axis-parallel lines. However, for the bundling to work, we need all the tests to

be making queries along the *same* axis-parallel line. We accomplish this by shifting our functions according to appropriate cyclic permutations of the coordinates so that the query patterns of the tests "line up" (at least into a constant number of groups).

To implement this idea, we first need some notation. As mentioned earlier, we will be able to prove robustness of the verifier via bundling; however, over a larger alphabet. This large alphabet will be $\Sigma = \mathbb{F}^{l+2l\cdot(m+1)}$. Unlike before, the proof oracle for the robust PCPP verifier will consist of only one function $\Pi : \mathbb{F}^m \to \Sigma$. The robust PCPP verifier simulates the PCPP verifier as follows: To answer the queries of the PCPP verifier, the robust verifier bundles several queries together, queries the new proof oracle Π, and then unbundles the answer to obtain the answers to the queries of the original PCPP verifier. For convenience, we index the $l + 2l \cdot (m + 1)$ coordinates of $\Sigma = \mathbb{F}^{l+2l\cdot(m+1)}$ as follows: The first l coordinates are indexed by a single index i ranging from 0 to $l - 1$, while the remaining $2l \cdot (m + 1)$ are indexed by a triplet of indices (i, j, b), where i ranges over $0, \ldots, l-1$, j ranges over $0, \ldots, m$, and $b \in \{0, 1\}$. Let $S : \mathbb{F}^m \to \mathbb{F}^m$ denote the (linear) transformation that performs one cyclic-shift to the right; that is, $S(x_0, \ldots, x_{m-1}) = (x_{m-1}, x_0, \ldots, x_{m-2})$. The bundling of the proof oracles $\tilde{A}_i$ and $P_{i,j}$ by the modified proof oracle Π is as follows:

(8.1)
$$\forall x \in \mathbb{F}^m, \begin{cases} \Pi(x)_i = \tilde{A}_i\left(S^{\lfloor \frac{i}{h} \rfloor}(x)\right) & i = 0, \ldots, l-1, \\ \Pi(x)_{(i,j,b)} = P_{i,j}^{(b)}\left(S^{j+\lfloor \frac{i}{h} \rfloor}(x)\right) & i = 0, \ldots, l-1;\ j = 0, \ldots, m;\ b \in \{0,1\}, \end{cases}$$

where $h = \log |H| = \log n/m$. Note that the size of the new proof oracle Π is exactly equal to the sum of the size of the oracles $\tilde{A}_i$ and $P_{i,j}$.

We now state how the robust verifier performs the unbundling and the individual tests. We consider each step of the PCPP verifier and present its robust counterpart.

The first steps of PCPP Verifier (and ALMSS PCPP Verifier) are independent of the proof oracle and are performed as before. That is, the robust verifier, as before, reduces the CktSAT instance to an instance $\langle 1^n, 1^m, \mathbb{F}, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle$ of AS-CktSAT, sets $d = m \cdot |H|$, and generates a random string R of length $\log(|S_\lambda| \cdot |\mathbb{F}|^{m-1})$. The remaining steps are proof oracle dependent and we will discuss each of them in detail.

Proximity test. For the proximity test, the only portion of the proof oracle that we require is the portion containing $\tilde{A}_0$. For this, we observe that $\Pi(x)_0$ is $\tilde{A}_0 \circ S^{\lfloor \frac{0}{h} \rfloor}(x) = \tilde{A}_0(x)$. The two different proximity tests (Robust Proximity Test and Robust ALMSS Proximity Test) can easily be describes as follows:

Robust Proximity Test$^{W;\,\Pi}(R)$.

Use random string R to determine a random canonical line $\mathcal{L}$ in $\mathbb{F}^m$ using the λ-biased set S_λ. Query oracle Π on all points along the line $\mathcal{L}$. Unbundle $\Pi(\mathcal{L})$ to obtain the values of $\tilde{A}_0$ on all points along the line $\mathcal{L}$ and reject if the restriction $\tilde{A}_0$ to $\mathcal{L}$ is not a polynomial of degree at most d. Query the input oracle W on all locations corresponding to those in $I \cap \mathcal{L}$ and reject if W disagrees with $\tilde{A}_0$ on any of the locations in $I \cap \mathcal{L}$.

Robust ALMSS Proximity Test$^{W;\,\Pi}$.

Choose a random position $i \overset{R}{\leftarrow} \{1, \ldots, k\}$ in the input and a direction $y \leftarrow \mathbb{F}^m$. Let $x \in I$ be the point corresponding to i in H^m, and let $\mathcal{L}$ be the line through x in direction y. Query oracle Π on all points along the line $\mathcal{L}$. Unbundle $\Pi(\mathcal{L})$ to obtain the values of $\tilde{A}_0$ on all points along

the line $\mathcal{L}$ and reject if the restriction $\tilde{A}_0$ to $\mathcal{L}$ is not a polynomial of degree at most d. Query the input oracle W at location i and reject if $W[i] \neq \tilde{A}_0(x)$.

Low-degree test. We note that the distance of the polynomial $\tilde{A}_i : \mathbb{F}^m \to \mathbb{F}$ to being degree k (for any $k \in \mathbb{Z}^+$) is exactly the same as that of $\tilde{A}_i \circ S^{\lfloor \frac{i}{h} \rfloor} : \mathbb{F}^m \to \mathbb{F}$ since $S^{\lfloor \frac{i}{h} \rfloor}$ is an invertible linear transformation. Hence, it is sufficient if we check that $\tilde{A}_i \circ S^{\lfloor \frac{i}{h} \rfloor}$ is low degree. The case with the $P_{i,j}^{(b)}$'s is similar. Thus, the new Robust Low-Degree Test can be described as follows:

Robust Low-Degree Test$^{\Pi}(R)$.

Use random string R to determine a random canonical line $\mathcal{L}$ in $\mathbb{F}^m$ using the λ-biased set S_λ.

Query the oracle Π on all points along the line $\mathcal{L}$.

For $i = 0, \ldots, l-1$,

unbundle $\Pi(\mathcal{L})$ to obtain the values of $\tilde{A}_i \circ S^{\lfloor \frac{i}{h} \rfloor}$ on all points along the line $\mathcal{L}$ and reject if the restriction $\tilde{A}_i \circ S^{\lfloor \frac{i}{h} \rfloor}$ to $\mathcal{L}$ is not a polynomial of degree at most d.

For $i = 0, \ldots, l-1$, $j = 0, \ldots, m$ and $b \in \{0, 1\}$,

unbundle $\Pi(\mathcal{L})$ to obtain the values of $P_{i,j}^{(b)} \circ S^{j+\lfloor \frac{i}{h} \rfloor}$ on all points along the line $\mathcal{L}$ and reject if the restriction of $P_{i,j}^{(b)} \circ S^{j+\lfloor \frac{i}{h} \rfloor}$ to $\mathcal{L}$ is not a polynomial of degree at most κd.

Thus, effectively we are testing $\tilde{A}_i$ (respectively, $P_{i,j}$) using the line space $S^{\lfloor \frac{i}{h} \rfloor} \circ S_\lambda$ (respectively, $S^{j+\lfloor \frac{i}{h} \rfloor} \circ S_\lambda$).

Identity test. In the case of Identity Test, we observe that $P_{i,m}^{(b)}$ vanishes on $\mathbb{F}^m$ iff $P_{i,m}^{(b)} \circ S^{m+\lfloor \frac{i}{h} \rfloor}$ vanishes on $\mathbb{F}^m$. Recall that we were allowed to use arbitrary partitions of the space $\mathbb{F}^m$. The set of random 1st axis-parallel lines is one such partition and we use this partition.

- Robust Identity Test$^{\Pi}(R)$.
  - Use random string R to determine a random 1st axis-parallel line in $\mathbb{F}^m$ of the form $\mathcal{L} = (X, a_1, \ldots, a_{m-1})$. Query the oracle Π on all points along the line $\mathcal{L}$.

    For $i = 0, \ldots, l-1$ and $b \in \{0, 1\}$,

    * unbundle $\Pi(\mathcal{L})$ to obtain the values of $P_{i,m}^{(b)} \circ S^{m+\lfloor \frac{i}{h} \rfloor}$ on all points along the line $\mathcal{L}$ and reject if any of these are nonzero.

Edge-consistency test. For any $x \in \mathbb{F}^m$, we say that $P_{i,0}$ is *well formed* at x if (6.2) is satisfied for this x. Edge-Consistency Test verifies that $P_{i,0}$ is well formed for all $x \in U_\eta$ and $i = 0, \ldots, l-1$. This was done earlier by reading the values of $P_{i,0}, \tilde{A}_i, \tilde{A}_{i+1} \circ \tilde{\Gamma}_{i,0} = \tilde{A}_{i+1}$ and $\tilde{A}_{i+1} \circ \tilde{\Gamma}_{i,1}$ for all $x \in U_\eta$.

Let $\mathcal{L}$ be a random 1st axis-parallel line. The robust version of this test checks that $P_{i,0}$ is well formed for all points on $S^{\lfloor \frac{i}{h} \rfloor}(\mathcal{L})$. Consider any $x = (x_0, \ldots, x_{m-1}) \in \mathcal{L}$. To verify that $P_{i,0}$ is well formed at $S^{\lfloor \frac{i}{h} \rfloor}(x)$, the verifier needs the values $P_{i,0}(S^{\lfloor \frac{i}{h} \rfloor}(x))$, $\tilde{A}_i(S^{\lfloor \frac{i}{h} \rfloor}(x))$, $\tilde{A}_{i+1}(S^{\lfloor \frac{i}{h} \rfloor}(x))$, and $\tilde{A}_{i+1} \circ \tilde{\Gamma}_{i,1}(S^{\lfloor \frac{i}{h} \rfloor}(x))$. We will show that all these values can be obtained from unbundling the value of Π on $\mathcal{L}$ and $S^{-1}(\mathcal{L})$. Clearly, the values $P_{i,0}(S^{\lfloor \frac{i}{h} \rfloor}(x))$ and $\tilde{A}_i(S^{\lfloor \frac{i}{h} \rfloor}(x))$ can be obtained from unbundling the value of Π at x. The other two values that we require are $\tilde{A}_{i+1}(S^{\lfloor \frac{i}{h} \rfloor}(x))$ and $\tilde{A}_{i+1} \circ \tilde{\Gamma}_{i,1}(S^{\lfloor \frac{i}{h} \rfloor}(x))$. We first show that $\tilde{\Gamma}_{i,1}(S^{\lfloor \frac{i}{h} \rfloor}(x)) = S^{\lfloor \frac{i}{h} \rfloor}(x')$ for $x' = (x_0 + e_{(i \bmod h)}, x_1, \ldots, x_{m-1}) \in \mathcal{L}$ (recall that $\{e_0, \ldots, e_{f-1}\}$ are a basis for $\mathbb{F}$ over $\mathbb{F}_2$ and $\{e_0, \ldots, e_{h-1}\}$ span $H \subset \mathbb{F}$). For this purpose, we first recall the definition of $\tilde{\Gamma}_{i,1}$: $\tilde{\Gamma}_{i,1}(z_0, \ldots, z_{m-1}) =$

$(z_0, \ldots, z_{t-1}, z_t + e_u, z_{t+1}, \ldots, z_{m-1})$, where $t = \lfloor i/h \rfloor \bmod m$ and $u = i \bmod h$. Furthermore, since S^m is the identity map, we have that $S^{\lfloor \frac{i}{h} \rfloor \bmod m} = S^{\lfloor \frac{i}{h} \rfloor}$. With these observations, we have the following:

$$\begin{aligned}\tilde{\Gamma}_{i,1}\left(S^{\lfloor \frac{i}{h} \rfloor}(x)\right) &= \tilde{\Gamma}_{i,1}\left(S^{\lfloor i/h \rfloor \bmod m}(x)\right)\\ &= \tilde{\Gamma}_{i,1}\left(S^{\lfloor i/h \rfloor \bmod m}(x_0, \ldots, x_{m-1})\right)\\ &= S^{\lfloor i/h \rfloor \bmod m}\left(x_0 + e_{(i \bmod h)}, x_1, \ldots, x_{m-1}\right)\\ &= S^{\lfloor \frac{i}{h} \rfloor}(x').\end{aligned}$$

Now, $S^{\lfloor \frac{i+1}{h} \rfloor}$ is either $S^{\lfloor \frac{i}{h} \rfloor}$ or $S^{\lfloor \frac{i}{h} \rfloor + 1}$ depending on the value of i. Suppose $S^{\lfloor \frac{i+1}{h} \rfloor} = S^{\lfloor \frac{i}{h} \rfloor}$. We then have that $\tilde{A}_{i+1}(S^{\lfloor \frac{i}{h} \rfloor}(x)) = A_{i+1}(S^{\lfloor \frac{i+1}{h} \rfloor}(x))$ and $\tilde{A}_{i+1} \circ \tilde{\Gamma}_{i,1}(S^{\lfloor \frac{i}{h} \rfloor}(x)) = \tilde{A}_{i+1}(S^{\lfloor \frac{i}{h} \rfloor}(x')) = \tilde{A}_{i+1}(S^{\lfloor \frac{i+1}{h} \rfloor}(x'))$. Both these values can be obtained by unbundling the value of Π on $\mathcal{L}$ (since both x and x' lie on $\mathcal{L}$). In the other case, where $S^{\lfloor \frac{i+1}{h} \rfloor} = S^{\lfloor \frac{i}{h} \rfloor + 1}$, we have $A_{i+1}(S^{\lfloor \frac{i}{h} \rfloor}(x)) = A_{i+1}(S^{\lfloor \frac{i+1}{h} \rfloor}(S^{-1}x))$ and $A_{i+1} \circ \tilde{\Gamma}_{i,1}(S^{\lfloor \frac{i}{h} \rfloor}(x)) = A_{i+1}(S^{\lfloor \frac{i}{h} \rfloor}(x')) = A_{i+1}(S^{\lfloor \frac{i+1}{h} \rfloor}(S^{-1}x'))$. These values can be obtained by unbundling the value of Π on $S^{-1}(\mathcal{L})$. Thus, to check that $P_{i,0}$ is well formed for all points on $S^{\lfloor \frac{i}{h} \rfloor}(\mathcal{L})$, it suffices if the verifier queries Π on all points on $\mathcal{L}$ and $S^{-1}(\mathcal{L})$.

ROBUST EDGE-CONSISTENCY TEST$^{\Pi}(R)$.

Use the random string R to determine a random 1st axis-parallel line in $\mathbb{F}^m$ of the form $\mathcal{L} = (X, a_2, \ldots, a_m)$. Query the oracle Π along all points in the lines $\mathcal{L}$ and $S^{-1}(\mathcal{L})$.

For $i = 0, \ldots, l-1$,

for all $x \in S^{\lfloor \frac{i}{h} \rfloor}(\mathcal{L})$, reject if $P_{i,0}$ is not well formed at x. (Note that all the values required for this verification can be obtained by unbundling $\Pi(\mathcal{L})$ and $\Pi(S^{-1}(\mathcal{L}))$.)

Zero-propagation test. For each $i = 0, \ldots, l-1$ and $b \in \{0,1\}$, ZERO-PROPAGATION TEST checks that $P_{i,0}^{(b)}$ vanishes on H^m by verifying that (6.3) is satisfied for all $j = 1, \ldots, m-1$ (we also need to check that $P_{i,m}^{(b)} \equiv 0$; however, this is taken care of by IDENTITY TEST). Since $S(H^m) = H^m$, checking if $P_{i,0}^{(b)}$ vanishes on H^m is equivalent to checking if $P_{i,0}^{(b)} \circ S^{\lfloor \frac{i}{h} \rfloor}$ vanishes on H^m. Hence, we can perform the zero propagation on the polynomials $P_{i,0}^{(b)} \circ S^{\lfloor \frac{i}{h} \rfloor}; i = 0, \ldots, l-1, b \in \{0,1\}$, instead of the polynomials $P_{i,0}^{(b)}; i = 0, \ldots, l-1, b \in \{0,1\}$. In other words, we need to verify the following equation instead (6.3):

(8.2)

$$\begin{aligned}&P_{i,j}^{(b)} \circ S^{\lfloor \frac{i}{h} \rfloor}\left(\underbrace{x_1, \ldots, x_{j-1}}, x_j, \underbrace{x_{j+1}, \ldots, x_m}\right)\\ &= \sum_{k=0}^{|H|-1} P_{i,j-1}^{(b)} \circ S^{\lfloor \frac{i}{h} \rfloor}\left(\underbrace{x_1, \ldots, x_{j-1}}, h_k, \underbrace{x_{j+1}, \ldots, x_m}\right) x_j^k \qquad \forall (x_1, \ldots, x_m) \in \mathbb{F}^m.\end{aligned}$$

This equation can be further rewritten in terms of the cyclic-shift S as follows:

(8.3)
$$P_{i,j}^{(b)}\left(S^{\lfloor \frac{i}{h} \rfloor + j - 1}(x_1, x_2, \ldots, x_m)\right) = \sum_{k=0}^{|H|-1} P_{i,j-1}^{(b)}\left(S^{\lfloor \frac{i}{h} \rfloor + j - 1}(h_k, x_2, \ldots, x_m)\right) x_1^k \quad \forall (x_1, \ldots, x_m) \in \mathbb{F}^m.$$

This helps us to rewrite Zero-Propagation Test with bundling as follows:

Robust Zero-Propagation Test$^{\Pi(R)}$.

Use random string R to determine a random 1st axis-parallel line in $\mathbb{F}^m$ of the form $\mathcal{L} = (X, a_2, \ldots, a_m)$. Query the oracle Π along all points in the lines $\mathcal{L}$ and $S^{-1}(\mathcal{L})$.

For $i = 0, \ldots, l-1$, $j = 1, \ldots, m$, and $b \in \{0, 1\}$,

unbundle $\Pi(\mathcal{L})$ to obtain the value of $P_{i,j-1}^{(b)} \circ S^{\lfloor \frac{i}{h} \rfloor + j - 1}$ on all points along the line $\mathcal{L}$. Similarly, unbundle $\Pi(S^{-1}(\mathcal{L}))$ to obtain the value of $P_{i,j}^{(b)} \circ S^{\lfloor \frac{i}{h} \rfloor + j}$ on all points along the line $S^{-1}(\mathcal{L})$ (equivalently, this is the value of $P_{i,j}^{(b)} \circ S^{\lfloor \frac{i}{h} \rfloor + j - 1}$ on all points along the line $\mathcal{L}$). Reject either if the restriction of $P_{i,j-1}^{(b)} \circ S^{\lfloor \frac{i}{h} \rfloor + j - 1}$ or $P_{i,j}^{(b)} \circ S^{\lfloor \frac{i}{h} \rfloor + j - 1}$ to $\mathcal{L}$ is not a polynomial of degree at most κd or if any of the points on the line $\mathcal{L}$ violate (8.3).

The integrated robust verifiers. Having presented the robust version of each of the tests, the integrated robust verifiers are as follows: Robust PCPP Verifier is the robust analogue of PCPP Verifier, while ALMSS Robust PCPP Verifier is that of ALMSS PCPP Verifier. Following are full descriptions of these verifiers as well as their analyses.

8.2.1. Robust PCPP Verifier. Using the robust tests presented above, we present a robust analogue of the PCPP of section 7.1.

Robust PCPP Verifier$^{W;\,\Pi}_{m,\lambda,\delta}(C)$.

1. Using Proposition 6.11, reduce the instance C of CktSAT, using parameter m, to an instance $\langle 1^n, 1^m, \mathbb{F}, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle$ of AS-CktSAT, and set $d = m \cdot |H|$.
   We let $S_\lambda \subset \mathbb{F}^m$ be a λ-biased set of size at most $(\frac{\log |\mathbb{F}|^m}{\lambda})^2$ [AGHP92].
2. Choose a random string R of length $\log(|S_\lambda| \cdot |\mathbb{F}|^{m-1})$. (Note: We reuse R in all tests, but only Low-Degree Test utilizes the full length of R.)
3. Run Robust Low-Degree Test$^{\Pi}(R)$.
4. Run Robust Edge-Consistency Test$^{\Pi}(R)$.
5. Run Robust Zero-Propagation Test$^{\Pi}(R)$.
6. Run Robust Identity Test$^{\Pi}(R)$.
7. Run Robust Proximity Test$^{W;\Pi}(R)$.

Reject if any of the above tests reject.

The randomness of Robust PCPP Verifier is exactly the same as before, whereas the query complexity and decision complexity increase by a constant factor.[32]

[32] Though the new proof oracle returns elements of Σ and not bits, we express the query complexity as the number of bits read by the verifier rather than as the number of symbols (i.e., elements of $|\Sigma|$) to maintain consistency throughout calculation of the query complexity into the proof and input oracles.

PROPOSITION 8.7. *The randomness, query, and decision complexities of* ROBUST PCPP VERIFIER *are* $r = \left(1 - \frac{1}{m}\right)\log n + O(m \log m) + O(\log\log n) + O\left(\log\left(\frac{1}{\delta}\right)\right)$, $q = O(m^2 n^{1/m} \log^2 n)$ *and* $d = \widetilde{O}(q)$, *respectively.*

It is straightforward to check perfect completeness of this verifier.

Robustness analysis of the integrated verifier. For future use, it is beneficial (but, alas, more cumbersome) to state the robustness of the integrated verifier in a way that decouples the robustness with respect to the input oracle from the robustness with respect to the proof oracle. Let $W : [k] \to \{0,1\}$ be the input oracle and Π the proof oracle. For every sequence of coin tosses R (and a given setting of parameters), let $\Delta_{\text{inp}}^{W,\Pi}(R)$ (resp., $\Delta_{\text{pf}}^{W,\Pi}(R)$) denote the fraction of the bits read from W (resp., Π) that would need to be changed to make ROBUST PCPP VERIFIER accept on coin tosses R. The following lemma states the (expected) robustness property of our verifier.

LEMMA 8.8. *There are constants* $c \in \mathbb{Z}^+$ *and* $\rho > 0$ *such the following holds for every* $n, m \in \mathbb{Z}^+$, $\delta', \delta > 0$ *satisfying* $m \le \log n / \log\log n$, $n^{1/m} \ge m^{cm}/{\delta'}^3$, $\lambda \le \min\{1/c\log n, {\delta'}^3/m^{cm}\}$, $\delta > \delta'$. *If* W *is* δ*-far from satisfying the circuit, then for any proof oracle* $\Pi : \mathbb{F}^m \to \Sigma$, *either* $\mathrm{E}_R[\Delta_{\text{pf}}^{W,\Pi}(R)] \ge \rho$ *or* $\mathrm{E}_R[\Delta_{\text{inp}}^{W,\Pi}(R)] \ge \delta - \delta'/2$.

That is, the expected robustness with respect to the input is $\delta - \delta'/2$ (which should be compared against the proximity parameter δ), whereas the expected robustness with respect to the proof is a universal constant. Note that combining the two bounds into a single expected robustness parameter depends on the relative number of queries made to the input and proof oracles. To obtain Theorem 3.1, we will later modify ROBUST PCPP VERIFIER such that the relative number of queries is optimized to yield the best result.

Proof. Unbundle the proof oracle Π to obtain the functions $\tilde{A}_i$ and $P_{i,j}$ using (8.2). Consider the action of PCPP VERIFIER (i.e., the nonrobust verifier) on the proof oracles $\tilde{A}_i, P_{i,j}$ and input oracle W.

Let ε be a sufficiently small constant such that the Claims 8.1–8.5 hold. Suppose W is δ-far from satisfying the circuit. We then know that one of the following holds and that the corresponding test instance of PCPP VERIFIER rejects its input robustly (see Claims 8.1–8.5).

1. There exists a $i \in \{0, \ldots, l-1\}$ such that $\tilde{A}_i$ is 8ε-far from every degree md polynomial or there exists $i \in \{0, \ldots, l-1\}$, $j \in \{0, \ldots, m\}$, and $b \in \{0,1\}$ such that $P_{i,j}^{(b)}$ is 8ε-far from every degree κmd polynomial. In this case,the expected distance of $\tilde{A}_i$ (or resp., $P_{i,j}^{(b)}$) from satisfying LOW-DEGREE TEST with degree parameter d (resp., κd) is at least 2ε (Claim 8.1).
2. There exists $i \in \{0, \ldots, l-1\}$ and $b \in \{0,1\}$, such that $\Delta(P_{i,m}^{(b)}, \widehat{P}_{i,m}^{(b)}) \le 8\varepsilon$ and $\widehat{P}_{i,m} \not\equiv 0$. In this case, $P_{i,m}$ has expected distance at least $1 - 9\varepsilon$ from satisfying IDENTITY TEST (Claim 8.2).
3. There exists $i \in \{0, \ldots, l-1\}$ such that

$$\Delta(P_{i,0}^{(0)}, \widehat{P}_{i,0}^{(0)}) \le 8\varepsilon, \Delta(P_{i,0}^{(1)}, \widehat{P}_{i,0}^{(1)}) \le 8\varepsilon, \Delta(\tilde{A}_i, \widehat{A}_i) \le 8\varepsilon, \Delta(\tilde{A}_{i+1}, \widehat{A}_{i+1}) \le 8\varepsilon,$$
$$\text{and } \widehat{P}_{i,0}(x) \not\equiv \psi'(\tilde{T}_i(x), \widehat{A}_i(x), \widehat{A}_{i+1}(\tilde{\Gamma}_{i,0}(x)), \widehat{A}_{i+1}(\tilde{\Gamma}_{i,1}(x))).$$

In this case, $\{P_{i,0}(\cdot), A_i(\cdot), A_{i+1}(\tilde{\Gamma}_{i,0}(\cdot))A_{i+1}(\tilde{\Gamma}_{i,1}(\cdot))\}$ has expected distance at least $(1 - 41\varepsilon)/5$ from satisfying EDGE-CONSISTENCY TEST (Claim 8.3).

4. There exists $i \in \{0, \ldots, l-1\}$, $j \in \{1, \ldots, m\}$ and $b \in \{0,1\}$ such that

$$\Delta(P_{i,j}, \widehat{P}_{i,j}) \leq 8\varepsilon, \Delta(P_{i,j-1}, \widehat{P}_{i,j-1}) \leq 8\varepsilon,$$
$$\text{and } \widehat{P}_{i,j}(\ldots, x_j, \ldots) \not\equiv \sum_{k=0}^{|H|-1} \widehat{P}_{i,j-1}(\ldots, h_k, \ldots) x_j^k.$$

   In this case, $(P_{i,j}^{(b)}, P_{i,j-1}^{(b)})$ has expected distance at least $(1-19\varepsilon)/2$ from satisfying Zero-Propagation Test (Claim 8.4).
5. $\Delta(\tilde{A}_0, \widehat{A}_0) \leq 8\varepsilon$, but W and $\widehat{A}_0|_I$ disagree on at least δ' fraction of the points. In this case, with probability at least $1-\delta'/4$ (over the choice of the canonical line $\mathcal{L}$) either at least an ε-fraction of $A_0|_{\mathcal{L}}$ or at least a $(\delta - \delta'/4)$-fraction of $W|_{\mathcal{L}}$ needs to be changed to make Proximity Test accept (Claim 8.5). This implies that either A_0 has expected distance $(1-\delta'/4)\varepsilon \geq \varepsilon/2$ or W has expected distance $(1-\delta'/4)(\delta-\delta'/4) \geq (\delta - \delta'/2)$ from satisfying Proximity Test.

For instance, lets us assume $\tilde{A}_0$ is 8ε-far from being low degree so Low-Degree Test rejects it robustly; that is, for a random canonical line $\mathcal{L}$, the expected distance of $\tilde{A}_0|_{\mathcal{L}}$ from satisfying Low-Degree Test is at least 2ε. Recall from (8.2) that $\tilde{A}_0(x)$ is one of the coordinates in the bundled $\Pi(x)$. Hence, if $\tilde{A}_0|_{\mathcal{L}}$ is ρ-far from satisfying Low-Degree Test, so is $\Pi_{\mathcal{L}}$ from satisfying Robust Low-Degree Test. Thus, Π has expected distance at least 2ε from satisfying Robust Low-Degree Test. Now, the oracle positions read by Robust Low-Degree Test constitute a constant fraction of the oracle positions read by Robust PCPP Verifier, so Π has expected distance $\Omega(\varepsilon)$ from satisfying Robust PCPP Verifier. Thus, the robustness of the individual test instance is transferred to the combined Robust Low-Degree Test by bundling. The case with the other test types is similar. We thus have that $\mathrm{E}_R[\Delta_{\mathrm{pf}}^{W,\Pi}(R)] \geq \Omega(\varepsilon)$ or $\mathrm{E}_R[\Delta_{\mathrm{inp}}^{W,\Pi}(R)] \geq \delta - \delta'/2$. The lemma then follows by setting $\rho = \Omega(\varepsilon)$. □

8.2.2. ALMSS Robust PCPP Verifier. We now describe ALMSS Robust PCPP Verifier(which is a robust analogue of the PCPP of section 7.2) and analyze its complexity. ALMSS Robust PCPP Verifier verifier is identical to Robust PCPP Verifier except that Robust Proximity Test is replaced by Robust ALMSS Proximity Test.

ALMSS Robust PCPP Verifier$_\delta^{W;\,\Pi}(C)$.

1. Set parameters $m = \log n / \log\log n$ and $\lambda = 1/c\log n$.
   Using Proposition 6.11, reduce the instance C of CktSAT, using parameter m, to an instance $\langle 1^n, 1^m, \mathbb{F}, H, \{\tilde{T}_0, \ldots, \tilde{T}_{l-1}\}\rangle$ of AS-CktSAT, and set $d = m \cdot |H|$.
   We let $S_\lambda \subset \mathbb{F}^m$ be a λ-biased set of size at most $(\frac{\log|\mathbb{F}|^m}{\lambda})^2$ [AGHP92].
2. Choose a random string R of length $\log(|S_\lambda| \cdot |\mathbb{F}|^{m-1})$.
3. Run Robust Low-Degree Test$^\Pi(R)$.
4. Run Robust Edge-Consistency Test$^\Pi(R)$.
5. Run Robust Zero-Propagation Test$^\Pi(R)$.
6. Run Robust Identity Test$^\Pi(R)$.
7. Run Robust ALMSS Proximity Test$^{W;\Pi}$.

Reject if any of the above tests reject.

The randomness of the ALMSS Robust PCPP Verifier is exactly the same as before, whereas the query complexity and decision complexity increase by a constant

factor. Furthermore, it can easily be verified that ALMSS Robust PCPP Verifier has perfect completeness.

Proposition 8.9. *The randomness, and decision complexities of* ALMSS Robust PCPP Verifier *are* $O(\log n)$ *and* $\operatorname{poly}\log n$*, respectively.*

Robustness analysis of the integrated verifier. As in the case of Robust PCPP Verifier, it is beneficial to state the robustness of ALMSS Robust PCPP Verifier by decoupling the robustness with respect to the input oracle from the robustness with respect to the proof oracle. Here, however, we refer to the robustness and soundness parameters (rather than to expected robustness).

Lemma 8.10. *If* W *is* δ*-far from satisfying the circuit, then for any proof oracle* $\Pi : \mathbb{F}^m \to \Sigma$*, with probability at least* $\Omega(\delta)$*, either a constant fraction of the portion of the proof oracle* Π *read by the verifier or the single symbol of the input oracle* W *read by the verifier (i.e.,* $W[i]$*) needs to be changed in order to make* ALMSS Robust PCPP Verifier *accept.*

Proof. This proof proceeds in the same way as the proof of Lemma 8.8. For the sake of completeness, we present the entire proof.

Let ε be a sufficiently small constant such that Claims 8.1–8.4 hold and $\varepsilon \le \varepsilon_0/8$, where ε_0 is the constant that appears in Claim 8.6. Suppose W is δ'-far from satisfying the circuit. We then know that one of the following holds and that the corresponding test instance of ALMSS PCPP Verifier rejects its input robustly (see Claims 8.1–8.6).

1. There exists $i \in \{0, \ldots, l-1\}$ such that $\tilde{A}_i$ is 8ε-far from every degree md polynomial or there exists $i \in \{0, \ldots, l-1\}$, $j \in \{0, \ldots, m\}$, and $b \in \{0,1\}$ such that $P_{i,j}^{(b)}$ is 8ε-far from every degree κmd polynomial. In this case, the expected distance of $\tilde{A}_i$ (resp., $P_{i,j}^{(b)}$) from satisfying Low-Degree Test with degree parameter d (resp., κd) is at least 2ε (Claim 8.1).
   Translating to the bundled alphabet, we have that the expected distance of Π from satisfying Robust Low-Degree Test is at least 2ε. Since the number of oracle positions read by ALMSS Robust PCPP Verifier is at least a constant fraction of the number of oracle positions read by ALMSS Robust PCPP Verifier, the expected distance of Π from satisfying ALMSS Robust PCPP Verifier in this case is at least $\Omega(\varepsilon)$. Hence, with probability at least $\Omega(\varepsilon)$ (= constant), at least $\Omega(\varepsilon)$ (= constant)-fraction of the proof oracle Π needs to be modified to make ALMSS Robust PCPP Verifier accept.
2. There exists $i \in \{0, \ldots, l-1\}$ and $b \in \{0,1\}$ such that $\Delta(P_{i,m}^{(b)}, \widehat{P}_{i,m}^{(b)}) \le 8\varepsilon$ and $\widehat{P}_{i,m} \not\equiv 0$. In this case, $P_{i,m}$ has expected distance at least $1 - 9\varepsilon$ from satisfying Identity Test (Claim 8.2).
   Arguing as in the earlier case, we have that with probability at least $\Omega(1-9\varepsilon)$, at least $\Omega(1-9\varepsilon)$-fraction of the proof oracle Π needs to be modified in this case to make ALMSS Robust PCPP Verifier accept.
3. There exists $i \in \{0, \ldots, l-1\}$ such that

$$\Delta(P_{i,0}^{(0)}, \widehat{P}_{i,0}^{(0)}) \le 8\varepsilon, \Delta(P_{i,0}^{(1)}, \widehat{P}_{i,0}^{(1)}) \le 8\varepsilon, \Delta(\tilde{A}_i, \widehat{A}_i) \le 8\varepsilon, \Delta(\tilde{A}_{i+1}, \widehat{A}_{i+1}) \le 8\varepsilon,$$
$$\text{and } \widehat{P}_{i,0}(x) \not\equiv \psi'(\tilde{T}_i(x), \widehat{A}_i(x), \widehat{A}_{i+1}(\tilde{\Gamma}_{i,0}(x)), \widehat{A}_{i+1}(\tilde{\Gamma}_{i,1}(x))).$$

   In this case, $\{P_{i,0}(\cdot), A_i(\cdot), A_{i+1}(\tilde{\Gamma}_{i,0}(\cdot))A_{i+1}(\tilde{\Gamma}_{i,1}(\cdot))\}$ has expected distance at least $(1 - 41\varepsilon)/5$ from satisfying Edge-Consistency Test (Claim 8.3).

Again, we have that with probability at least $\Omega(1-41\varepsilon)$, at least $\Omega(1-41\varepsilon)$-fraction of the proof oracle Π needs to be modified in this case to make ALMSS Robust PCPP Verifier accept.

4. There exists $i \in \{0,\dots,l-1\}$, $j \in \{1,\dots,m\}$, and $b \in \{0,1\}$ such that

$$\Delta(P_{i,j}, \widehat{P}_{i,j}) \le 8\varepsilon, \Delta(P_{i,j-1}, \widehat{P}_{i,j-1}) \le 8\varepsilon,$$
$$\text{and } \widehat{P}_{i,j}(\dots, x_j, \dots) \not\equiv \sum_{k=0}^{|H|-1} \widehat{P}_{i,j-1}(\dots, h_k, \dots) x_j^k.$$

In this case, $(P_{i,j}^{(b)}, P_{i,j-1}^{(b)})$ has expected distance at least $(1-19\varepsilon)/2$ from satisfying Zero-Propagation Test (Claim 8.4).
We have that with probability at least $\Omega(1-19\varepsilon)$, at least $\Omega(1-19\varepsilon)$-fraction of the proof oracle Π needs to be modified in this case to make ALMSS Robust PCPP Verifier accept.

5. $\Delta(\tilde{A}_0, \widehat{A}_0) \le 8\varepsilon \le \varepsilon_0$, but W and $\widehat{A}_0|_I$ disagree on at least δ-fraction of the points. In this case, with probability at least $\Omega(\delta)$ (over the choice of index i and direction y), either at least a 1/2-fraction of $A_0|_{\mathcal{L}}$ or $W[i]$ (i.e., the entire portion of the input oracle read by the verifier) needs to be changed to make ALMSS Proximity Test accept.
This implies that with probability at least δ, either a constant fraction of the proof oracle Π or $W[i]$ (i.e., the entire portion of the input oracle read by the verifier) needs to be modified to make the verifier accept.

Since we do not know which of the five cases occur, we can guarantee only the weakest of the five claims. Hence, with probability at least $\Omega(\delta)$, either a constant fraction of the portion of the proof oracle Π read by the verifier or $W[i]$ (i.e., the entire portion of the input oracle W read by the verifier) needs to be changed in order to make ALMSS Robust PCPP Verifier accept. □

8.3. Robustness over the binary alphabet. The transformation from a robust verifier over the alphabet Σ to one over the binary alphabet is analogous to converting non-Boolean error-correcting codes to Boolean ones via "code concatenation." This transformation is exactly the same transformation as the one in the proof of Lemma 2.13. However, we cannot directly use Lemma 2.13 because we may apply the code concatenation process only to the proof oracle Π and not to the input oracle W. However, this is not a problem because the input oracle is already binary. Recall that applying the aforementioned transformation maintains the robustness of the proof oracle *up to a constant factor*, whereas the robustness of the input oracle remains unchanged (like the input oracle itself). Actually, in order to avoid decoding (by the modified decision circuit), we maintain the original proof oracle along with its encoded form. Thus, the complexity of this circuit will depend on the minimum between the complexity of encoding and decoding (rather than on the complexity of decoding). Details follow.

Let $\mathrm{ECC} : \{0,1\}^{\log|\Sigma|} \to \{0,1\}^b$ for $b = O(\log|\Sigma|)$ be a binary error-correcting code of constant relative minimum distance, which can be computed by an explicit circuit of size $O(\log|\Sigma|)$ [Spi96]. We augment the original proof oracle Π, viewed now as having $\log|\Sigma|$-bit long entries (i.e., elements of Σ) with an additional oracle Υ having b-bit long entries, where $\Upsilon(x)$ is assumed to be $\mathrm{ECC}(\Pi(x))$.

The actual transformation. We describe the transformation to the binary alphabet in the case of Robust PCPP Verifier. ALMSS Robust PCPP Verifier

can be transformed similarly. Our new verifier V, on oracle access to the input W and proof $\Pi \circ \Upsilon$, will simulate ROBUST PCPP VERIFIER. The queries to the input oracle are performed just as before. However, for each query $x \in \mathbb{F}^m$ in the proof oracle Π made by ROBUST PCPP VERIFIER, V will query the corresponding $\log|\Sigma|$ bits in $\Pi(x)$ and the b bits in $\Upsilon(x)$. Thus, the query complexity of V is at most $\log|\Sigma| + b$ times the number of queries issued by the earlier verifier. Since $b = O(\log|\Sigma|)$, the query complexity of the new verifier V is a constant times that of the previous one,[33] and the decision complexity will increase by at most the encoding time (which can even be linear). The randomness is exactly the same. The action of the new verifier V is as follows: Suppose ROBUST PCPP VERIFIER issues queries $x_1, \ldots, x_{q_1}$ to the proof oracle Π, and queries $i_1, \ldots, i_{q_2}$ to the input oracle; then V issues queries $x_1, \ldots, x_{q_1}$ to the proof oracle Π, a similar set of queries $x_1, \ldots, x_{q_1}$ to the proof oracle Υ, and $i_1, \ldots, i_{q_2}$ to the input oracle. V accepts $(\Pi(x_1), \ldots, \Pi(x_{q_1}), \Upsilon(x_1), \ldots, \Upsilon(x_{q_1}), W(i_1), \ldots, W(i_{q_2}))$ iff ROBUST PCPP VERIFIER accepts $(\Pi(x_1), \ldots, \Pi(x_{q_1}), W(i_1), \ldots, W(i_{q_2}))$ and $\Upsilon(x_i) = \mathrm{ECC}(\Pi(x_i))$ for all $i = 1, \ldots, q_1$. It is straightforward to check that V has perfect completeness if ROBUST PCPP VERIFIER has perfect completeness. For the robust soundness, we define $\Delta_{\mathrm{inp}}^{W,\Pi\circ\Upsilon}(R)$ and $\Delta_{\mathrm{pf}}^{W,\Pi\circ\Upsilon}(R)$ with respect to V analogously as in the statement immediately preceding Lemma 8.8, but referring to distance over $\{0,1\}$ (rather than Σ) for the proof oracle. The proof of the following claim regarding the robust soundness of V mimics the proof of Lemma 2.13.

LEMMA 8.11. *There are constants $c \in \mathbb{Z}^+$ and $\rho' > 0$ such that the following holds for every $n, m \in \mathbb{Z}^+$, $\delta, \delta' > 0$ satisfying $m \le \log n / \log\log n$, $n^{1/m} \ge m^{cm}/{\delta'}^3$, $\lambda \le \min\{1/c\log n, {\delta'}^3/m^{cm}\}$, $\delta > \delta'$. If W is δ-far from satisfying the circuit, then for any proof oracles $\Pi : \mathbb{F}^m \to \{0,1\}^{\log|\Sigma|}$, $\Upsilon : \mathbb{F}^m \to \{0,1\}^b$, either $\mathrm{E}_R[\Delta_{\mathrm{pf}}^{W,\Pi\circ\Upsilon}(R)] \ge \rho'$ or $\mathrm{E}_R[\Delta_{\mathrm{inp}}^{W,\Pi\circ\Upsilon}(R)] \ge \delta - \delta'/2$.*

A similar transformation for ALMSS ROBUST PCPP VERIFIER yields a verifier, the robustness of which is stated in Lemma 8.12 following. It is to be noted that the robustness of the proof oracle (i.e., ρ' in Lemma 8.11 and $\Omega(1)$ in Lemma 8.12) is a constant factor smaller than the corresponding parameter in the nonbinary verifier (i.e., the constant ρ in Lemma 8.8 and a different $\Omega(1)$ in Lemma 8.10). (Indeed, this constant factor appears also in Lemma 2.13.)

LEMMA 8.12. *If W is δ-far from satisfying the circuit, then for any proof oracles $\Pi : \mathbb{F}^m \to \{0,1\}^{\log|\Sigma|}$, $\Upsilon : \mathbb{F}^m \to \{0,1\}^b$, with probability at least $\Omega(\delta)$, either a constant (i.e., $\Omega(1)$) fraction of the portion of the proof oracle $\Pi\circ\Upsilon$ read by the verifier or $W[i]$ (i.e., the entire portion of the input oracle W read by the verifier) needs to be changed in order to make the transformed* ALMSS ROBUST PCPP VERIFIER *accept.*

We finally turn to deriving Theorem 3.1 (and Theorem 3.2).

Proof of Theorem 3.1. Theorem 3.1 is proved using ROBUST PCPP VERIFIER defined in this section, setting $\lambda = \min\{1/c\log n, \delta^3/m^{cm}\}$. The randomness, query, and decision complexity of ROBUST PCPP VERIFIER (i.e., before the transformation to the binary alphabet) are as mentioned in Proposition 8.7. As mentioned earlier in this section, the transformation from the alphabet Σ to the binary alphabet maintains the randomness complexity, while the query (and decision) complexity increases by at most a constant factor.

The manner in which the robustness of the verifier is expressed in Lemma 8.11

[33] Recall that the query complexity of the old verifier was measured in terms of "bits of information" rather than in terms of queries. That is, each query, answered by an element of Σ, contributes $\log_2|\Sigma|$ to the query complexity.

differs from that in Theorem 3.1 in two aspects. First, Lemma 8.11 expresses the robustness for the proof and input oracles separately, while Theorem 3.1 expresses them together. Second, Lemma 8.11 expresses robustness in terms of expected robustness, while Theorem 3.1 does it in terms of standard robustness. We obtain robustness as claimed in Theorem 3.1 in two steps, first by combining the proof and input oracles and then by moving from expected robustness to standard robustness.

First, we combine the robustness of the proof and input oracles, which were expressed separately in Lemma 8.11. This is done by giving adequate weights to the two oracle portions in the decision circuits (i.e., repeating queries; see Proposition 7.3). Let n, m, δ, and γ be as specified in Theorem 3.1. We give weight $(1-\gamma/3)$ to the input oracle and $\gamma/3$ to the proof oracle. Recall that these weights mean that each query to the input oracle is repeated several times such that the relative length of the input-part in the decision circuit is $1-\gamma/3$. These repeated queries may increase the query (and decision) complexity by a factor of at most $O(1/\gamma)$. Note that weighting does not affect the randomness complexity (or any other parameter, such as the proximity parameter δ).

Since $n^{1/m} \geq m^{cm}/\delta^3$, we have $n^{1/m} \geq 8000(c_F m^2)^{m-1}/\delta^3$, or equivalently $n \geq 8000|\mathbb{F}|^{m-1}/\delta^3$. Hence, Lemma 8.11 can be applied. Setting $\delta' = 2\delta\gamma/3$ in Lemma 8.11, we have that either $\mathrm{E}_R[\Delta_{\mathrm{pf}}^{W,\Pi\circ\Upsilon}(R)] \geq \rho'$ or $\mathrm{E}_R[\Delta_{\mathrm{inp}}^{W,\Pi\circ\Upsilon}(R)] \geq \delta - \delta'/2 = \delta(1-\gamma/3)$. Note that the first expression refers to the "expected robustness" of the proof-part, whereas the second expression refers to the input-part. The overall expected robustness is obtained by a weighted average of these two expressions, where the weights are with respect to the aforementioned weighting (which assigns weight $\gamma/3$ to the input-part). Hence, the expected robustness with respect to the said weighting is

$$\frac{\gamma}{3} \cdot \mathrm{E}_R[\Delta_{\mathrm{pf}}^{W,\Pi\circ\Upsilon}(R)] + \left(1-\frac{\gamma}{3}\right) \cdot \mathrm{E}_R[\Delta_{\mathrm{inp}}^{W,\Pi\circ\Upsilon}(R)] \;\geq\; \min\left\{\frac{\gamma}{3}\cdot\rho', \left(1-\frac{\gamma}{3}\right)^2 \cdot \delta\right\}.$$

This quantity is lower-bounded by $\rho \triangleq (1-\gamma/3)^2\delta$ since $\delta \leq \gamma/c$ for a suitably large c (and $\rho' > 0$ is a constant). We have thus obtained a robust PCPP for CktVal with randomness and decision complexities as claimed in Proposition 8.7, perfect completeness, and $\rho = (1-\gamma/3)^2\delta$ expected robustness for proximity parameter δ.

We now move from expected robustness to standard robustness, by using Lemma 2.11. Applying Lemma 2.11 with a slackness parameter of $\gamma' \triangleq \gamma\rho/3$ and $s = \gamma/3$ yields a robust-soundness error of $\gamma/3 \leq \gamma$ with robustness parameter of $\rho - \gamma' = (1-\gamma/3)^3 \cdot \delta \geq (1-\gamma)\delta$ for proximity parameter δ. Using $\gamma \leq 1/2$, note that the randomness increases by an additive term of $O(\log(1/\gamma')) + O(\log(1/\gamma)) = O(\log(1/\delta))$, and the decision complexity increases by a multiplicative factor of $O\left(1/(\gamma\cdot(\gamma\rho)^2)\right) = \mathrm{poly}(1/\delta)$. Hence, the randomness, query, and decision complexities of the verifier are as claimed in Theorem 3.1 □

Proof of Theorem 3.2. For this purpose we use ALMSS Robust PCPP Verifier described in this section. This verifier is then transformed to one over the binary alphabet as indicated earlier in this section. We combine the robustness of the proof and input oracles by giving equal weights to both oracles. This weighting may increase the query (and decision) complexity by at most a factor of 2 and has no affect on any other parameter. Proposition 8.9 and Lemma 8.12 then imply Theorem 3.2. □

8.4. Linearity of encoding. In this section we point out that, for linear circuits (to be defined below), the mapping from an assignment to the corresponding PCPP

is linear. Throughout this section, "linear" means $\mathbb{F}_2$-linear (yet, we will sometimes refer to $\mathbb{F}$-linearity, for an extension field $\mathbb{F}$ of $\mathbb{F}_2$). The main motivation of the current study is to derive linear codes satisfying local-testability and relaxed local-decodability (i.e., Theorems 1.4 and 1.5, respectively). Specifically, the constructions presented in section 4 yield linear codes provided that the corresponding PCPP is linear in the aforementioned sense.

We call a circuit *linear* if it is a conjunction of linear constraints. However, instead of representing this conjunction via AND gates, it is more convenient for us to work with circuits that have multiple output gates, i.e., one for each linear constraint. See the following definition.

DEFINITION 8.13. *A multi-output circuit is* linear *if all its* internal *gates are parity gates and an input is accepted by it iff all output gates evaluate to zero.*

PROPOSITION 8.14. *If C is a linear circuit, then there is a linear transformation T mapping satisfying assignments w of C to proof oracles $T(w)$ such that the PCPP verifier of Theorem* 3.1 *will, on input C, accept oracle $(w, T(w))$ with probability* 1*. Moreover, all the decision circuits produced by the verifier, on input C, can be made linear (while maintaining the claimed decision complexity). A similar result is true for the PCPP verifier of Theorem* 3.2*.*

In the rest of this section, we provide a proof of Proposition 8.14, starting with an assignment w that satisfies the linear circuit. We prove that the mapping from w to a proof oracle is linear by reviewing our construction of this mapping and ensuring that all steps in this construction are linear transformations.

Phase I: STRUCTURED-CKTSAT. In this phase (described in section 6.1) we write down the values to all gates of the circuit and route them along the wrapped de Bruijn graph. Actually, we make a few minor and straightforward modifications to Definition 6.3: we allow multiple output gates (rather than a single output gate) and require that each such gate evaluates to zero (rather than to 1).[34] Also, here we deal with gate types that are linear (e.g., XOR) rather than arbitrary (e.g., AND and OR).

Since all the circuit gates are linear functions of the input, the values on the wires leaving the zeroth layer of the well-structured circuit (i.e., the last two bits of the mapping $A_0 : \{0,1\}^N \to \{0,1\}^4$ in section 6.1) are linear in the input (i.e., in w). As to A_i, $i > 0$, (and the first two bits of A_0) notice that it is obtained by permuting the values of the previous layer A_{i-1} and setting some wires to zero (if they are not needed in the routing (e.g., gates 3 and 4 in Figure 3)). These operations are linear, and so all assignment functions are linear in the input.

Phase II: *Arithmetization.* In this phase (described in section 6.2) we extend the values given by A_i to an evaluation of a low-degree multivariate polynomial over a finite field $\mathbb{F}$ that is an extension field of $\mathbb{F}_2$ of degree $\mathfrak{v}$. Each value of A_i is four bits long (say, b_0, b_1, b_2, b_3) and identified with the element $b_0 e_0 + b_1 e_1 + b_2 e_2 + b_3 e_3$, where $e_0, \ldots, e_{\mathfrak{v}-1}$ is a basis for $\mathbb{F}$ viewed as a vector space over $\mathbb{F}_2$. We view A_i as a function $A_i : H^m \to \mathbb{F}$ and construct a low-degree extension $\tilde{A}_i : \mathbb{F}^m \to \mathbb{F}$ of A_i by interpolation on all inputs in H^m and use these values to interpolate and evaluate $\tilde{A}_i$ on all points in $\mathbb{F}^m$. Notice that interpolation is $\mathbb{F}$-linear, and hence also $\mathbb{F}_2$-linear. We conclude that the values of $\tilde{A}_i$ on all points in $\mathbb{F}^m$ are a linear transformation of the values of A_i. Since A_i is linear in the input assignment, so is $\tilde{A}_i$.

Clarification. Many parts of our encoding (starting with $\tilde{A}_i$) consist of evaluations of multivariate polynomials $P(x)$ over $\mathbb{F}^m$. The linearity we claim is not linearity in

[34] Recall that an input is accepted by the linear circuit iff all output gates evaluate to zero.

x (the free variables of the polynomial). Rather, we claim that the table of values $\{P(a) : a \in \mathbb{F}^m\}$ is linear in the initial assignment w, which may be viewed as the information encoded in this table. In contrast, throughout this section, x is merely an index to this table. For example, in Phase II we showed that the table $\{\tilde{A}_i(a) : a \in \mathbb{F}^m\}$ is obtained by a linear transformation applied to the table $\{A_i(a') : a' \in H^m\}$ (but we certainly do not claim that $\tilde{A}_i(a)$ is linear in a). That is, each $\tilde{A}_i(a)$ is a linear combination of the $A_i(a')$'s.

Phase III: *The constraint polynomials.* We now discuss the polynomials $P^{(0)}_{i,0}$ and $P^{(1)}_{i,1}$ defined in (6.2) and show that their values are a linear transformation of the values of $\tilde{A}_i$. The first polynomial (i.e., $P^{(0)}_{i,0}$) is obtained by applying the univariate polynomial ψ_0 defined in (6.1) to each value of $\tilde{A}_i$ (i.e., $P^{(0)}_{i,0}(x) = \psi_0(\tilde{A}_i(x))$). By definition, ψ_0 evaluates to zero iff its input, when represented as a vector in $\mathbb{F}_2^{\mathfrak{V}}$, belongs to the linear space spanned by $\{e_0, e_1, e_2, e_3\}$. This polynomial defines a linear transformation, as claimed by the following lemma.

LEMMA 8.15. *Let L be an $\mathbb{F}_2$-linear subspace of $\mathbb{F} = \mathbb{F}_2^{\mathfrak{V}}$ and $\psi_L(t) = \prod_{\alpha \in L}(t - \alpha)$. Then the mapping $\psi_L : \mathbb{F} \to \mathbb{F}$ is linear.*

Proof. We use the fact that for any integer i, the transformation $t \mapsto t^{2^i}$ is linear; that is, $(t + t')^{2^i} = t^{2^i} + t'^{2^i}$. Our main claim is that the polynomial $\psi_L(t)$ can be written as $\sum_i c_i t^{2^i}$ and hence is linear (being a sum of linear transformations). We prove this claim by induction on the dimension of $L \subseteq \mathbb{F}_2^{\mathfrak{V}}$. Indeed, for $\dim(L) = 0$ (i.e., $L = \{0^{\mathfrak{V}}\}$), it holds that $\psi_L(t) = t$, and our claim follows. In the induction step, write L as $L = L' \cup \{\alpha + L'\}$, where L' is a linear space of dimension $k - 1$ and $\alpha \in L \setminus L'$. Clearly, $\psi_L(t) = \psi_{L'}(t) \cdot \psi_{L'}(t + \alpha)$. Using the inductive hypothesis for L' (and the linearity of $t \mapsto t^{2^j}$), we get

$$\begin{aligned}
\psi_L(t) &= \left(\sum_i c_i \cdot t^{2^i}\right) \cdot \left(\sum_j c_j \cdot (t + \alpha)^{2^j}\right) \\
&= \left(\sum_i c_i \cdot t^{2^i}\right) \cdot \left(\sum_j c_j \cdot \left(t^{2^j} + \alpha^{2^j}\right)\right) \\
&= \sum_{i,j} c_i c_j t^{2^i} t^{2^j} + \sum_{i,j} c_i c_j t^{2^i} \alpha^{2^j} \\
&= \sum_i c_i^2 t^{2^{i+1}} + \sum_i c'_i t^{2^i},
\end{aligned}$$

where $c'_i = \sum_j c_i c_j \alpha^{2^j}$ and $\sum_{i \neq j} c_i c_j t^{2^i} t^{2^j} = 2 \sum_{i<j} c_i c_j t^{2^i} t^{2^j} = 0$ (because $\mathbb{F}$ has characteristic 2). This completes the proof of the inductive claim. ∎

We now turn to the second polynomial, $P^{(1)}_{i,0}$. Recall that $P^{(1)}_{i,0}(x) = \psi_1(s, a, a_0, a_1)$, where $s = \tilde{T}_i(x)$, $a = \tilde{A}_i(x)$, and $a_j = \tilde{A}_{i+1}(\tilde{\Gamma}_{i,j}(x))$. It can be verified that $\tilde{T}_i(x)$ (which represents the gate type) is independent of the input w to the circuit, and by our previous discussion a, a_0, a_1 are linear in the input w (to the circuit). Thus, it will suffice to show that ψ' is linear in its last three inputs. When discussing (6.2) we did not go into the specific construction of the polynomial ψ' because only its functionality mattered, and we showed that there exists a constant-degree polynomial that does the job. But for our current purposes (of showing linearity) we need to present a specific polynomial ψ' that is linear (as an operator over $\mathbb{F}_2^{\mathfrak{V}}$) and has the desired

properties needed by the verification process. To do this, we recall that $\mathcal{C}$ is the set of allowable gates in the well-structured circuit, define $\delta_{s_0}(z)$ to be the (minimal degree) univariate polynomial of degree $|\mathcal{C}|$ that is 1 if $z = s_0$ and 0 if $z \in \mathcal{C} \setminus \{s_0\}$, and write ψ' as

$$\psi'(s, a, a_0, a_1) = \sum_{s_0 \in \mathcal{C}} \delta_{s_0}(s) \cdot \psi'_{s_0}(a, a_0, a_1). \tag{8.4}$$

CLAIM 8.16. *For any $s_0 \in \mathcal{C}$ that can occur as a gate in a well-structured circuit constructed from a linear circuit C, the polynomial $\psi'_{s_0}(a, a_0, a_1)$ of (8.4) can be written as a linear transformation (of (a, a_0, a_1)).*

Proof. Recall that the value of $\psi'_{s_0}(a, a_0, a_1)$ is assumed to represent whether or not the four least significant bits of the three inputs (denoted a', a'_0, and a'_1) satisfy some specified condition. By inspecting Definition 6.3, it can be verified that (in our case) this condition is linear. That is, $\psi'_{s_0}(a, a_0, a_1) = 0$ iff the triplet (a', a'_0, a'_1), viewed as a 12-bit vector over $\mathbb{F}_2$, belongs to a specific linear space $L_{s_0} \subseteq \mathbb{F}_2^{12}$.

Recall that we may assume that $a = 0^{f-4}a'$ (and similarly for a_0 and a_1) because this condition is imposed by the constraint polynomial $P^{(0)}_{i,0}$. Thus, we seek a polynomial (over $\mathbb{F}^3$) such that if each of its three inputs belongs to $Span(e_0, \ldots, e_3)$, then it will output 0 iff the inputs reside in the linear space that is analogous to L_{s_0}; that is, the input (a, a_0, a_1) should evaluate to 0 iff $a' \circ a'_0 \circ a'_1 \in L_{s_0}$. To obtain this, we assume the existence of $\alpha \in \mathbb{F}$ such that multiplying an element by α corresponds to a left cyclic-shift by four positions (e.g., $\alpha \cdot \sigma_0 \cdots \sigma_{f-1} = \sigma_4 \cdots \sigma_{f-1}\sigma_0 \cdots \sigma_3$). Such an element exists for the standard representation of $\mathbb{F}$. Using this element we can write $\psi'_{s_0} : \mathbb{F}^3 \to \mathbb{F}$ as

$$\psi'_{s_0}(a, a_0, a_1) = \psi_{L_{s_0}}(\alpha^2 a + \alpha a_0 + a_1),$$

where $\psi_{L_{s_0}}$ is the univariate polynomial that is zero iff its input is in L_{s_0}. Note that, for inputs in $Span(e_0, \ldots, e_3)$, indeed $\psi'_{s_0}(a, a_0, a_1) = 0$ iff $a' \circ a'_0 \circ a'_1 \in L_{s_0}$. By Lemma 8.15, $\psi_{L_{s_0}}$ is linear. It follows that ψ'_{s_0} is linear because multiplication by a fixed element of $\mathbb{F}$ (i.e., α) is a linear operation. ∎

Recall $\delta_{s_0}(s)$ depends only on the circuit and not on its input (i.e., w). Thus, each summand of (8.4) is linear in w, and hence the sum is itself linear in w. We conclude that the table of evaluations of the polynomials given by (6.2) is obtained by linear transformations applied to the input to the circuit.

Phase IV: *The sum-check polynomials.* In this phase (described by (6.3)) we apply a sequence of interpolations to previously constructed polynomials $P^{(b)}_{i,j}$. Each such interpolation is an $\mathbb{F}$-linear transformation and hence also an $\mathbb{F}_2$-linear one. Thus, the sequence of polynomials $P^{(b)}_{i,j}$ is obtained by a linear transformation applied to the input.

Phase V: *Bundling and encoding.* In this phase (described in sections 8.2 and 8.3) we apply some cyclic-shifts to the (values of the) sequence of $l + 2l(m+1)$ polynomials obtained in the previous phases. Then we bundle the polynomials together, obtaining an alphabet of size $|\mathbb{F}|^{l+2l(m+1)}$. This bundling does not change the encoding (only the partitioning of the proof into symbols) and hence is also a linear transformation. Finally, we apply an error-correcting code to each symbol in order to reduce the alphabet size (from $|\mathbb{F}|^{l+2l(m+1)}$) to binary, and this is also a linear transformation as long as the error-correcting code is itself linear.

The result of this shifting, bundling, and encoding is the actual proof given to the (outer) verifier of Theorem 3.1 (the verifier of Theorem 3.2 is dealt with in a

similar fashion). Notice that this transformation from $l+2l(m+1)$ polynomials (each evaluated in $\mathbb{F}$) to one proof (over the binary alphabet) is linear because all three parts of it are linear.

Now we argue that all tests performed by the verifier are linear and the decision complexity claimed in Theorems 3.1 and 3.2 can be achieved by using small linear circuits. This can be seen by inspecting the various tests described in section 6.3, noticing that they all check either linear or $\mathbb{F}$-linear conditions, and applying the general result of Strassen [Str73], showing that any algebraic circuit that computes a linear function (as a formal polynomial) can be converted into a linear circuit with only a constant-factor increase in size. This completes the proof of Proposition 8.14.

Part III. Appendices.

Appendix A. Hadamard code–based PCPP. In this section we note that the Hadamard code–based inner verifier from Arora et al. [ALM+98] can be converted into a PCPP. Recall that the inner verifier of [ALM+98] accesses $O(1)$ input oracles, where the ith oracle is assumed to provide the Hadamard *encoding* of some string w_i, and verifies that their concatenation satisfies some given circuit C.

Here we simplify this verifier to work with a single string w, and the verifier accesses a *single input oracle* that represents this string itself (not some encoding of it) and verifies that w is close to an assignment acceptable by the circuit C given as explicit input.

THEOREM A.1. *There exists a constant $\delta_0 > 0$ such that there exists a PCPP for* CKTVAL *(for circuits of size n) with randomness complexity $O(n^2)$, query complexity $O(1)$, perfect completeness, soundness error $1-\delta$, and proximity parameter 5δ for any $\delta \le \delta_0$. That is, inputs that are δ-far from satisfying the circuit are rejected with probability at least $\min(\delta, \delta_0)/5$.*

Notice that we do not claim robustness of this PCPP. This is because we don't intend to use this verifier (or any verifier derived from it) as the outer verifier during composition. However, this verifier is robust (in a trivial sense). Indeed, any PCPP with $O(1)$ query complexity is trivially ρ-robust for some constant $\rho > 0$ (since the relative distance between two query patterns is lower-bounded by the inverse of number of bits queried).

Proof. Let V denote the claimed verifier. We first list the oracles used by V, then we describe the tests that V performs, and finally we verify that V's complexities are as claimed and analyze its performance (most notably its soundness and proximity).

Oracles. Let C be a circuit with n gates on m input bits. The verifier accesses an input oracle $W : [m] \to \{0,1\}$ (representing a string $w \in \{0,1\}^m$) and a proof oracle $\Pi = (A, B)$, with $A : \{0,1\}^n \to \{0,1\}$ and $B : \{0,1\}^{n \times n} \to \{0,1\}$.

To motivate the verifier's tests, we describe what is expected from the oracles in the "completeness" case, i.e., when $C(w) = 1$. The input oracle, by definition, gives the string w, i.e., $W[i] = w_i$. Now let $z \in \{0,1\}^n$ be the string of values of all the gates of the circuit C (including the input, the internal gates, and the output gate(s)). Without loss of generality, assume $z = w \circ y$, where y represents the values assumed for internal gates. The oracle A is expected to give the values of all linear functions at z (over $\mathbb{F}_2$), and the oracle B is supposed to give the value of all quadratic functions at z. More precisely, $A = A[x]_{x \in \{0,1\}^n}$ is expected to be $A[x] = \sum_{i=1}^{n} x_i z_i = x^T z$ (where x and z are thought of as column vectors). Similarly, $B = B[M]_{M \in \{0,1\}^{n \times n}}$ is expected to be $B[M] = \sum_{i,j} M_{ij} z_i z_j = z^T M z$ (where M is an $n \times n$ matrix). In order to verify that w satisfies C, the verifier will verify that A and B have indeed been constructed according to some string z as above, that z represents an accepting

computation of the circuit, and finally that A is the encoding of some string $w' \circ y$, where w' is close to the string w given by the input oracle W.

Tests. Given the circuit C, the verifier first constructs polynomials $P_1(z), \ldots, P_n(z)$ as follows. Viewing the variables $\{z_i\}$ as representing the values at the individual gates of the circuit C (with $z_1, \ldots, z_m$ being the input gates), the polynomial $P_i(z)$ is the quadratic polynomial (over $\mathbb{F}_2$) expressing the constraint imposed by the ith gate of the circuit on an accepting computation. For example,

$$P_i(z) = \begin{cases} z_i - z_j z_k & \text{if the } i\text{th gate is an AND gate with inputs from gates } j \text{ and } k. \\ z_i - z_j - z_k + z_j z_k & \text{if the } i\text{th gate is an OR gate with inputs from gates } j \text{ and } k. \\ z_i - (1 - z_j) & \text{if the } i\text{th gate is a NOT gate with input from gate } j. \\ z_i - (z_j + z_k) & \text{if the } i\text{th gate is a PARITY gate with inputs from gates } j \text{ and } k. \\ 1 - z_j & \text{if the } i\text{th gate is an output gate with input from gate } j. \\ 0 & \text{if the } i\text{th gate is an input gate (i.e., } i \leq m). \end{cases}$$

Note that $z = w \circ y$ reflects the computation of C on an acceptable input w iff $P_i(z) = 0$ for every $i \in [n]$. The verifier conducts the following tests.

Codeword tests. These tests refer to (A, B) being a valid encoding of some string $z \in \{0,1\}^n$. That is, these tests check that both A and B are linear functions, and that B is consistent with A. In the latter check, the verifier employs a self-correction procedure (cf. [BLR93]) to the oracle B. (There is no need to employ self-correction to A because it is queried at random locations.)

Linearity of A. Pick x_1, x_2 uniformly at random from $\{0,1\}^n$ and verify that $A[x_1 + x_2] = A[x_1] + A[x_2]$.

Linearity of B. Pick M_1, M_2 uniformly at random from $\{0,1\}^{n \times n}$ and verify that $B[M_1 + M_2] = B[M_1] + B[M_2]$.

Consistency of A and B. Pick x_1, x_2 uniformly at random from $\{0,1\}^n$ and M uniformly from $\{0,1\}^{n \times n}$ and verify that $B[M + x_1 x_2^T] - B[M] = A[x_1]A[x_2]$.

Circuit test. This test checks that the string z encoded in (A, B) represents an accepting computation of C; that is, that $P_i(z) = 0$ for every $i \in [n]$. The test checks that a random linear combination of the P_i's evaluates to 0, while employing self-correction to A and B.

Pick $\alpha_1, \ldots, \alpha_n \in \{0,1\}$ uniformly and independently and let $\sum_{k=1}^{n} \alpha_k P_k(z) = c_0 + \sum_i \ell_i z_i + \sum_{i,j} Q_{i,j} z_i z_j$. Pick $x \in \{0,1\}^n$ and $M \in \{0,1\}^{n \times n}$ uniformly at random. Verify that $c_0 + (A[x + \ell] - A[x]) + (B[M + Q] - B[M]) = 0$.

Proximity test. This test checks that the m-bit long prefix of the string z, encoded in A, matches (or is close to) the input oracle W, while employing self-correction to A.

Pick $j \in [m]$ and $x \in \{0,1\}^n$ uniformly. Let $e_j \in \{0,1\}^n$ denote the vector that is 1 in the jth coordinate and 0 everywhere else. Verify that $W[j] = A[x + e_j] - A[x]$.

The verifier *accepts* if all the tests above accept; otherwise it *rejects*.

Resources. The verifier uses $O(n^2)$ random bits and makes $O(1)$ binary queries.

Completeness. It is straightforward to see that if w, the string given by W, satisfies C, then letting z be the set of values of the gates of C and letting $A[x] = x^T z$ and $B[M] = z^T M z$ will satisfy all tests above. Thus the verifier has perfect completeness.

Soundness (with proximity). It follows directly from the analysis of [ALM+98] that there exists a $\delta_0 > 0$ such that for every $\delta \leq \delta_0$, if *Codeword tests* and *Circuit test* above accept with probability at least $1 - \delta$, then the oracle A is 2δ-close to the Hadamard encoding of some string $z = w' \circ y$ such that $C(w')$ accepts. Now we augment this soundness with a proximity condition. Suppose the verifier also accepts *Proximity test* with probability at least $1-\delta$. Then we have that $w_j \neq A[x+e_j]-A[x]$ with probability at most δ. Furthermore, the events $A[x + e_j] \neq (x + e_j)^T z$ and $A[x] \neq x^T z$ happen with probability at most 2δ each. Thus, with probability at least $1 - 5\delta$ (over the possible choices of j and x), both $w_j = A[x + e_j] - A[x]$ and $A[x + e_j] - A[x] = (x + e_j)^T z - x^T z$ hold. Since $(x + e_j)^T z - x^T z = e_j^T z = z_j = w'_j$, it follows that, with probability at least $1 - 5\delta$ (over the choices of j), $w_j = w'_j$. In other words, the string w represented by the oracle W is at distance at most 5δ away from some string w' that is accepted by the circuit C. □

Appendix B. Randomness-efficient low-degree tests and the sampling lemma. Following [BSVW03], our construction makes heavy use of small-biased spaces [NN93] to save on randomness when choosing random lines. For a field $\mathbb{F}$ and parameters $m \in \mathbb{Z}^+$ and $\lambda > 0$, we require a set $S \subseteq \mathbb{F}^m$ that is λ-*biased* (with respect to the additive group of $\mathbb{F}^m$). Rather than define small-biased spaces here, we simply state the properties we need. (See, e.g., [BSVW03] for definitions and background on small-biased spaces.)

LEMMA B.1. *For every $\mathbb{F}$ of characteristic 2, $m \in \mathbb{Z}^+$, and $\lambda > 0$, there is an explicit construction of a λ-biased set $S \subseteq \mathbb{F}^m$ of size at most* $(\log |\mathbb{F}^m|)/\lambda^2$ [AGHP92].

We now discuss the properties of such sets that we will use.

Expanding Cayley graphs. λ-biased sets are very useful pseudorandom sets in algebraic applications, and this is due in part to the expansion properties of the Cayley graphs they generate. See the following lemma.

LEMMA B.2. *If $S \subseteq \mathbb{F}^m$ is λ-biased and we let G_S be the graph with vertex set $\mathbb{F}^m$ and edge set $\{(x, x + s) : x \in \mathbb{F}^m, s \in S\}$, then all the nontrivial eigenvalues of G_S have absolute value at most $\lambda|S|$.*

Randomness-efficient line samplers. In [BSVW03], Lemma B.2 was used to prove the following sampling lemma. This lemma says that if one wants to estimate the density of a set $B \subseteq \mathbb{F}^m$ using lines in $\mathbb{F}^m$ as the sample sets, one does not need to pick a random line in $\mathbb{F}^m$ which costs $2 \log |\mathbb{F}^m|$ random bits. A pseudorandom line whose slope comes from a λ-biased set will do nearly as well, and the randomness is only $(1 + o(1)) \cdot \log |\mathbb{F}^m|$. In what follows, $l_{x,y}$ is the line passing through point x in direction y, formally; $l_{x,y} = \{x + ty : t \in \mathbb{F}\}$

LEMMA B.3 ([BSVW03, Sampling Lemma 4.3]). *Suppose $S \subseteq \mathbb{F}^m$ is λ-biased. Then, for any $B \subseteq \mathbb{F}^m$ of density $\mu = |B|/|\mathbb{F}^m|$, and any $\zeta > 0$,*

$$\Pr_{x \in \mathbb{F}^m, y \in S}\left[\left|\frac{|l_{x,y} \cap B|}{|l_{x,y}|} - \mu\right| > \zeta\right] \leq \left(\frac{1}{|\mathbb{F}|} + \lambda\right) \cdot \frac{\mu}{\zeta^2}.$$

Randomness-efficient low-degree tests. Ben-Sasson et al. [BSVW03] use the randomness-efficient sampling lemma, Lemma B.3, to obtain randomness-efficient low-degree tests by performing a "line versus point" test only for pseudorandom lines with a direction y coming from a small λ-biased set. That is, for a set $S \subseteq \mathbb{F}^m$, we consider

lines of the form $l_{x,y}(t) = x + ty$ for $x \in \mathbb{F}^m$ and $y \in S$, and let $\mathbb{L}$ be the set of all such lines, where each line is parameterized in a canonical way.

Then for functions $f : \mathbb{F}^m \to \mathbb{F}$ and $g : \mathbb{L} \to P_d$, where P_d is the set of univariate polynomials of degree at most d over $\mathbb{F}$, we let $\mathrm{LDT}_{S,d}^{f,g}$ be the test that uniformly selects $l \overset{R}{\leftarrow} \mathbb{L} \triangleq \{l_{x,y} : x \in \mathbb{F}^m, y \in S\}$ and $t \in \mathbb{F}$ and accepts iff $g(l)(t) = f(l(t))$. That is, the value of the degree d univariate polynomial $g(l)$ at point t equals the value of f at $l(t)$. We quote the main theorem of [BSVW03] and will use it in our constructions.

THEOREM B.4 ([BSVW03, Theorem 4.1]). *There exists a universal constant $\alpha > 0$ such that the following holds. Let $d \le |\mathbb{F}|/3, m \le \alpha|\mathbb{F}|/\log|\mathbb{F}|, S \subseteq \mathbb{F}^m$ be a λ-biased set for $\lambda \le \alpha/(m \log|\mathbb{F}|)$, and $\delta \le \alpha$. Then, for every $f : \mathbb{F}^m \to \mathbb{F}$ and $g : \mathbb{L} \to P_d$ such that f is at least 4δ-far from any polynomial of degree at most md, we have the following:*

$$\Pr[\mathrm{LDT}_{S,d}^{f,g} = \mathrm{rej}] > \delta.$$

Acknowledgments. We are grateful to Avi Wigderson for collaborating with us at early stages of this research and to Irit Dinur for inspiring discussions at late stages of this research. We thank Sergey Yekhanin and Jaikumar Radhakrishnan for bringing to our attention the "algebraic AND" mentioned in footnote 28. We thank the two anonymous referees for their careful reading and many comments which improved the presentation.

REFERENCES

[AGHP92] N. ALON, O. GOLDREICH, J. HÅSTAD, AND R. PERALTA, *Simple constructions of almost k-wise independent random variables*, Random Structures Algorithms, 3 (1992), pp. 289–304.

[ALM+98] S. ARORA, C. LUND, R. MOTWANI, M. SUDAN, AND M. SZEGEDY, *Proof verification and the hardness of approximation problems*, J. ACM, 45 (1998), pp. 501–555. (Preliminary version in Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science, 1992, pp. 14–23.)

[AS98] S. ARORA AND S. SAFRA, *Probabilistic checking of proofs: A new characterization of NP*, J. ACM, 45 (1998), pp. 70–122. (Preliminary version in Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science, 1992, pp. 2–13.)

[BFLS91] L. BABAI, L. FORTNOW, L. A. LEVIN, AND M. SZEGEDY, *Checking computations in polylogarithmic time*, in Proceedings of the 23rd ACM Symposium on Theory of Computing, New Orleans, LA, 1991, pp. 21–31.

[Bar01] B. BARAK, *How to go beyond the black-box simulation barrier*, in Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, Las Vegas, NV, 2001, pp. 106–115.

[BIKR02] A. BEIMEL, Y. ISHAI, E. KUSHILEVITZ, AND J. F. RAYMOND, *Breaking the $O(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval*, in Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science, Vancouver, BC, Canada, 2002, pp. 261–270.

[BGS95] M. BELLARE, O. GOLDREICH, AND M. SUDAN, *Free bits, PCPs, and nonapproximability—towards tight results*, SIAM J. Comput., 27 (1998), pp. 804–915. (Preliminary version in Proceedings of the 36th IEEE Symposium on Foundations of Computer Science, 1995, pp. 422–431.)

[BGLR93] M. BELLARE, S. GOLDWASSER, C. LUND, AND A. RUSSELL, *Efficient probabilistically checkable proofs and applications to approximation*, in Proceedings of the 25th ACM Symposium on Theory of Computing, San Diego, CA, 1993, pp. 294–304.

[BGH+04a] E. BEN-SASSON, O. GOLDREICH, P. HARSHA, M. SUDAN, AND S. VADHAN, *Robust PCPs of proximity, shorter PCPs and applications to coding*, in Proceedings of the 36th ACM Symposium on Theory of Computing, Chicago, IL, 2004, pp. 1–10.

[BGH+04b] E. BEN-SASSON, O. GOLDREICH, P. HARSHA, M. SUDAN, AND S. VADHAN, *Robust PCPs of Proximity, Shorter PCPs and Applications to Coding*, Tech. report

TR04-021, Electronic Colloquium on Computational Complexity, 2004. Available online at http://eccc.hpi-web.de/eccc-reports/2004/TR04-021/index.html

[BGH+05] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan, *Short PCPs verifiable in polylogarithmic time*, in Proceedings of the 20th IEEE Conference on Computational Complexity, San Jose, CA, 2005, pp. 120–134.

[BHR05] E. Ben-Sasson, P. Harsha, and S. Raskhodnikova, *Some 3CNF properties are hard to test*, SIAM J. Comput., 35 (2005), pp. 1–21. (Preliminary version in Proceedings of the 35th ACM Symposium on Theory of Computing, 2003, pp. 345–354.)

[BS05] E. Ben-Sasson and M. Sudan, *Simple PCPs with poly-log rate and query complexity*, in Proceedings of the 37th ACM Symposium on Theory of Computing, Baltimore, MD, 2005, pp. 266–275.

[BSVW03] E. Ben-Sasson, M. Sudan, S. Vadhan, and A. Wigderson, *Randomness-efficient low degree tests and short PCPs via epsilon-biased sets*, in Proceedings of the 35th ACM Symposium on Theory of Computing, San Diego, CA, 2003, pp. 612–621.

[BLR93] M. Blum, M. Luby, and R. Rubinfeld, *Self-testing/correcting with applications to numerical problems*, J. Comput. System Sci., 47 (1993), pp. 549–595. (Preliminary version in Proceedings of the 22nd ACM Symposium on Theory of Computing, 1990, pp. 73–83.)

[BOT02] A. Bogdanov, K. Obata, and L. Trevisan, *A lower bound for testing 3-colorability in bounded-degree graphs*, in Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science, Vancouver, Canada, 2002, pp. 93–102.

[BT04] A. Bogdanov and L. Trevisan, *Lower bounds for testing bipartiteness in dense graphs*, in Proceedings of the 19th IEEE Conference on Computational Complexity, Amherst, MA, 2004, pp. 75–81.

[BdW04] H. Buhrman and R. de Wolf, *On Relaxed Locally Decodable Codes*, manuscript, 2004.

[CGH04] R. Canetti, O. Goldreich, and S. Halevi, *The random oracle methodology, revisited*, J. ACM, 51 (2004), pp. 557–594. (Preliminary version in Proceedings of the 30th ACM Symposium on Theory of Computing, 1998, pp. 209–218.)

[Coo88] S. A. Cook, *Short propositional formulas represent nondeterministic computations*, Inform. Process. Lett., 26 (1988), pp. 269–270.

[DJK+02] A. Deshpande, R. Jain, T. Kavitha, S. V. Lokam, and J. Radhakrishnan, *Lower bounds for adaptive locally decodable codes*, Random Structures Algorithms, 27 (2005), pp. 358–378. (Preliminary version in Proceedings of the 17th IEEE Conference on Computational Complexity, 2002, pp. 152–161.)

[Din06] I. Dinur, *The PCP theorem by gap amplification*, in Proceedings of the 38th ACM Symposium on Theory of Computing, Seattle, WA, 2006, pp. 241–250.

[DR04] I. Dinur and O. Reingold, *Assignment-testers: Towards a combinatorial proof of the PCP-theorem*, in Proceedings of the 45th IEEE Symposium on Foundations of Computer Science, Rome, Italy, 2004, pp. 155–164.

[EKR04] F. Ergün, R. Kumar, and R. Rubinfeld, *Fast approximate probabilistically checkable proofs*, Inform. Comput., 189 (2004), pp. 135–159. (Preliminary version in Proceedings of the 31st ACM Symposium on Theory of Computing, 1999, pp. 41–50.)

[FGL+96] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, *Interactive proofs and the hardness of approximating cliques*, J. ACM, 43 (1996), pp. 268–292. (Preliminary version in Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science, 1991, pp. 2–12.)

[FRS94] L. Fortnow, J. Rompel, and M. Sipser, *On the power of multi-prover interactive protocols*, Theoret. Comput. Sci., 134 (1994), pp. 545–557. (Preliminary version in Proceedings of the 3rd IEEE Symposium on Structure in Complexity Theory, 1988, pp. 156–161)

[Gol97] O. Goldreich, *A Sample of Samplers—A Computational Perspective on Sampling*, Tech. report TR97-020, Electronic Colloquium on Computational Complexity, 1997. Available online at http://www.eccc.uni-trier.de/eccc-reports/1997/TR97-020.

[GGR98] O. Goldreich, S. Goldwasser, and D. Ron, *Property testing and its connection to learning and approximation*, J. ACM, 45 (1998), pp. 653–750. (Preliminary version in Proceedings of the 37th IEEE Symposium on Foundations of Computer Science, 1996, pp. 339–348.)

[GR02] O. Goldreich and D. Ron, *Property testing in bounded degree graphs*, Algorithmica, 32 (2002), pp. 302–343. (Preliminary version in Proceedings of the 29th ACM

Symposium on Theory of Computing, 1997, pp. 416–415.)

[GS00] O. Goldreich and S. Safra, *A combinatorial consistency lemma with application to proving the PCP theorem*, SIAM J. Comput., 29 (2000), pp. 1132–1154. (Preliminary version in RANDOM, Lecture Notes in Comput. Sci. 1269, Springer, Berlin, 1997, pp. 67–84.)

[GS02] O. Goldreich and M. Sudan, *Locally testable codes and PCPs of almost-linear length*, in Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science, Vancouver, Canada, 2002, pp. 13–22. Available online at http://www.wisdom.weizmann.ac.il/~oded/p_ltc.html.

[GW97] O. Goldreich and A. Wigderson, *Tiny families of functions with random properties: A quality-size trade-off for hashing*, Random Structures Algorithms, 11 (1997), pp. 315–343. (Preliminary version in Proceedings of the 26th ACM Symposium on Theory of Computing, 1994, pp. 574–584.)

[GLST98] V. Guruswami, D. Lewin, M. Sudan, and L. Trevisan, *A tight characterization of NP with* 3*-query PCPs*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science, Palo Alto, CA, 1998, pp. 18–27.

[HS00] P. Harsha and M. Sudan, *Small PCPs with low query complexity*, Comput. Complex., 9 (2000), pp. 157–201. (Preliminary version in 18th STACS, Lecture Notes in Comput. Sci. 2010, Springer, Berlin, 2001, pp. 327–338.)

[Hås01] J. Håstad, *Some optimal inapproximability results*, J. ACM, 48 (2001), pp. 798–859. (Preliminary version in Proceedings of the 29th ACM Symposium on Theory of Computing, 1997, pp. 1–10.)

[HS66] F. C. Hennie and R. E. Stearns, *Two-tape simulation of multitape Turing machines*, J. ACM, 13 (1966), pp. 533–546.

[KT00] J. Katz and L. Trevisan, *On the efficiency of local decoding procedures for error-correcting codes*, in Proceedings of the 32nd ACM Symposium on Theory of Computing, Portland, OR, 2000, pp. 80–86.

[KdW04] I. Kerenidis and R. de Wolf, *Exponential lower bound for* 2*-query locally decodable codes via a quantum argument*, J. Comput. System Sci., 69 (2004), pp. 395–420. (Preliminary version in Proceedings of the 35th ACM Symposium on Theory of Computing, 2003, pp. 106–115.)

[Kil92] J. Kilian, *A note on efficient zero-knowledge proofs and arguments (extended abstract)*, in Proceedings of the 24th ACM Symposium on Theory of Computing, Victoria, BC, Canada, 1992, pp. 723–732.

[LS97] D. Lapidot and A. Shamir, *Fully parallelized multi prover protocols for NEXP-time*, J. Comput. System Sci., 54 (1997), pp. 215–220. (Preliminary version in Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science, 1991, pp. 13–18.)

[Lei92] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures*, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1992.

[LFKN92] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, *Algebraic methods for interactive proof systems*, J. ACM, 39 (1992), pp. 859–868. (Preliminary version in Proceedings of the 31st IEEE Symposium on Foundations of Computer Science, 1990, pp. 2–10.)

[Mic00] S. Micali, *Computationally sound proofs*, SIAM J. Comput., 30 (2000), pp. 1253–1298. (Preliminary version in Proceedings of the 35th IEEE Symposium on Foundations of Computer Science, 1994, pp. 436–453.)

[MR06] D. Moshkovitz and R. Raz, *Sub-constant error low degree test of almost linear size*, in Proceedings of the 38th ACM Symposium on Theory of Computing, Seattle, WA, 2006, pp. 21–30.

[NN93] J. Naor and M. Naor, *Small-bias probability spaces: Efficient constructions and applications*, SIAM J. Comput., 22 (1993), pp. 838–856. (Preliminary version in Proceedings of the 22nd ACM Symposium on Theory of Computing, 1999, pp. 213–233.)

[PF79] N. Pippenger and M. J. Fischer, *Relations among complexity measures*, J. ACM, 26 (1979), pp. 361–381.

[PS94] A. Polishchuk and D. A. Spielman, *Nearly-linear size holographic proofs*, in Proceedings of the 26th ACM Symposium on Theory of Computing, Montréal, Québec, Canada, 1994, pp. 194–203.

[Raz98] R. Raz, *A parallel repetition theorem*, SIAM J. Comput., 27 (1998), pp. 763–803. (Preliminary version in Proceedings of the 27th ACM Symposium on Theory of Computing, 1995, pp. 447–556.)

[RS96] R. Rubinfeld and M. Sudan, *Robust characterizations of polynomials with applications to program testing*, SIAM J. Comput., 25 (1996), pp. 252–271. (Preliminary versions in Proceedings of the 23rd ACM Symposium on Theory of Computing, 1991, pp. 33–42 and Proceedings of the 3rd ACM–SIAM Symposium on Discrete Algorithms, 1992, pp. 23–43.)

[ST00] A. Samorodnitsky and L. Trevisan, *A PCP characterization of NP with optimal amortized query complexity*, in Proceedings of the 32nd ACM Symposium on Theory of Computing, Portland, OR, 2000, pp. 191–199.

[Sch77] A. Schönhage, *Schnelle multiplikation von polynomen über Körpern der charakteristik* 2 *[Fast multiplication of polynomials over characteristic* 2*]*, Acta Inform., 4 (1977), pp. 395–398.

[SS71] A. Schönhage and V. Strassen, *Schnelle multiplikation großer zahlen [Fast multiplication of large numbers]*, Computing, 7 (1971), pp. 281–292.

[Spi96] D. A. Spielman, *Linear-time encodable and decodable error-correcting codes*, IEEE Trans. Inform. Theory, 42 (1996), pp. 1723–1732. (Preliminary version in Proceedings of the 27th ACM Symposium on Theory of Computing, 1995, pp. 388–397.)

[Spi95] D. A. Spielman, *Computationally Efficient Error-Correcting Codes and Holographic Proofs*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, June 1995.

[Str73] V. Srassen, *Vermeidung von Divisionen [Avoiding divisions]*, J. Reine Angew. Math., 264 (1973), pp. 184–202.

[Sze99] M. Szegedy, *Many-valued logics and holographic proofs*, in Proceedings of the 26th International Colloquium of Automata, Languages, and Programming (ICALP '99), J. Wiedermann, P. van Emde Boas, and M. Nielsen, eds., Lecture Notes in Comput. Sci. 1644, Springer-Verlag, Berlin, pp. 676–686.

SIAM J. COMPUT.
Vol. 36, No. 4, pp. 975–1024

ASSIGNMENT TESTERS: TOWARDS A COMBINATORIAL PROOF OF THE PCP THEOREM*

IRIT DINUR† AND OMER REINGOLD‡

Abstract. In this work we look back into the proof of the PCP (probabilistically checkable proofs) theorem, with the goal of finding new proofs that are "more combinatorial" and arguably simpler. For that we introduce the notion of an assignment tester, which is a strengthening of the standard PCP verifier, in the following sense. Given a statement and an alleged proof for it, while the PCP verifier checks correctness of the *statement*, the assignment tester checks correctness of the statement *and the proof*. This notion enables composition that is truly modular; i.e., one can compose two assignment testers without any assumptions on how they are constructed. A related notion called *PCPs of proximity* was independently introduced in [E. Ben-Sasson et al., *Proceedings of the* 36*th Annual ACM Symposium on Theory of Computing*, Chicago, IL, 2004, ACM, New York, 2004, pp. 1–10]. We provide a toolkit of (nontrivial) generic transformations on assignment testers. These transformations may be interesting in their own right, and allow us to present the following two main results: 1. A new proof of the PCP theorem. This proof relies on a rather weak assignment tester given as a "black box." From this, we construct combinatorially the full PCP. An important component of this proof is a new combinatorial aggregation technique (i.e., a new transformation that allows the verifier to read fewer, though possibly longer, "pieces" of the proof). An implementation of the black-box tester can be obtained from the algebraic proof techniques that already appear in [L. Babai et al., *Proceedings of the* 23*rd ACM Symposium on Theory of Computing*, New Orleans, LA, 1991, ACM, New York, 1991, pp. 21–31; U. Feige et al., *J. ACM*, 43 (1996), pp. 268–292]. 2. Our second construction is a "standalone" combinatorial construction showing $NP \subseteq PCP[polylog, 1]$. This implies, for example, that approximating max-SAT is quasi-NP-hard. This construction relies on a transformation that makes an assignment tester "oblivious," so that the proof locations read are independent of the statement that is being proven. This eliminates, in a rather surprising manner, the need for aggregation in a crucial point in the proof.

Key words. PCP theorem, assignment tester, combinatorial proof

AMS subject classifications. 68Q15, 68Q17, 68R05

DOI. 10.1137/S0097539705446962

1. Introduction.

The PCP theorem is a characterization of the class NP which was discovered in the early 90s [2, 1] following an exhilarating sequence of results, including [20, 3, 7, 15, 22, 27, 5, 4, 13], to list just a few. It has had tremendous impact; most notably it lies at the heart of virtually all inapproximability results, starting with the seminal work of [13].

Recall that a language L is in NP if there is a polynomial-time algorithm (verifier) that can verify whether an input is in the language, with the assistance of a proof, called the NP witness. The PCP theorem says that every NP witness can be rewritten

*Received by the editors January 7, 2005; accepted for publication (in revised form) April 1, 2006; published electronically December 15, 2006. A preliminary version of this work appeared in Proceedings of the 45th Symposium on Foundations of Computer Science (FOCS '04), Rome, 2004, pp. 155–164.

http://www.siam.org/journals/sicomp/36-4/44696.html

†Computer Science Department, Hebrew University, Jerusalem, Israel (dinuri@cs.huji.ac.il). Part of this research was performed while the author was at NEC Research, Princeton, NJ.

‡Incumbent of the Walter and Elise Haas Career Development Chair, Department of Computer Science and Applied Mathematics, Weizmann Institute, Rehovot, 76100 Israel (omer.reingold @weizmann.ac.il). Part of this research was performed while the author was at AT&T Labs—Research, Florham Park, NJ, and while visiting the Institute for Advanced Study, Princeton, NJ. This author's research was supported in part by US-Israel Binational Science Foundation grant 2002246.

in a "PCP" format that allows ultraefficient (probabilistic) checking. Hence the name, **P**robabilistically **C**heckable **P**roofs.

More concretely, the PCP verifier is an algorithm that is given direct access to a proof, and also a logarithmic number of random coins. The verifier reads the input, tosses the random coins, and then decides which (constant number of) bits to read from the proof. Based on the content of these bits, the verifier decides whether to accept or reject. The PCP theorem asserts the existence of a polynomial-time verifier for any $L \in NP$ such that

- (completeness) for every $x \in L$ there is a proof that causes the verifier to always accept;
- (soundness) for every $x \notin L$, every alleged proof causes the verifier to reject with probability 99% over its coin tosses.

Let us fix the NP-language L in our discussion to be (circuit) satisfiability (SAT). For every fixed outcome of the random coin tosses of the verifier, the verifier's action can be described by specifying which (constant number of) bits are read from the proof, along with the acceptance predicate over these bits. Enumerating over all possible random coin tosses, one gets a list of $\mathsf{R} = 2^{O(\log n)} = n^{O(1)}$ constant-size predicates (described, say, via circuits) over Boolean variables representing the bits in the proof. Thus, the verifier can be thought of as a deterministic polynomial-time reduction whose input is a Boolean circuit φ of size n (denoted $|\varphi| = n$), and whose output is a list of $\mathsf{R} = n^{O(1)}$ circuits $\psi_1, \ldots, \psi_{\mathsf{R}}$ over a new set of variables such that the following holds:

- (completeness) if φ is satisfiable, then there is an assignment that simultaneously satisfies all of $\psi_1, \ldots, \psi_{\mathsf{R}}$;
- (soundness) if φ is unsatisfiable, then every assignment simultaneously satisfies at most 1% of $\psi_1, \ldots, \psi_{\mathsf{R}}$.

This result is already nontrivial for $|\psi_i| = o(n)$, but in fact holds for $|\psi_i| = O(1)$. Indeed, the discovery of this theorem was extremely surprising. The proof is not an easy one and combines many beautiful and influential ideas.

1.1. Overall goals. Acknowledging the importance of the PCP theorem, we look back into its proof, with the goal of finding proofs that are substantially different and desirably also simpler. We note that while the statement of the PCP theorem is purely "combinatorial," the original proof of [2, 1] is heavily based on algebra: low degree polynomials play a crucial role. Therefore, of particular interest is coming up with proofs of "combinatorial" nature; see also [19]. In that we were also influenced by the view that such combinatorial proofs, albeit more messy, are sometimes more intuitive (or at least may shed new intuition). We also note that such new proofs and constructions have the potential to imply new results, unknown with previous techniques. A recent example is the combinatorial construction of expander graphs and the subsequent construction of the so-called lossless expanders [25, 9].

Composition and recursion. We first tackle a major ingredient in the original proof, namely the use of composition. Composition is indeed natural in this setting. A given verifier reduction may possibly be improved by applying another (inner) verifier reduction to each one of $\psi_1, \ldots, \psi_{\mathsf{R}}$, replacing each ψ_i with a system Υ_i of even smaller circuits.

Unfortunately, this simplistic composition has a "consistency flaw." It is likely that the resulting system of circuits $\bigcup_i \Upsilon_i$ will be completely satisfiable, even if the original circuit φ is unsatisfiable. Indeed each one of the ψ_i's could be individually satisfiable, as it is only impossible to satisfy more than 1% of them *by the same assignment*. Since $\Upsilon_1, \ldots, \Upsilon_{\mathsf{R}}$ are outcomes of independent runs of the second verifier

reduction, they are defined over syntactically disjoint sets of variables. This makes it easy to combine inconsistent assignments (each satisfying one ψ_i) into an assignment that satisfies all of $\cup\Upsilon_i$.

In the proof of [2, 1], this difficulty was overcome by having the second verifier utilize the concrete structural details of the circuits $\psi_1, \ldots, \psi_{\mathsf{R}}$ output by the first verifier, and relying on ingenious consistency mechanisms of algebraic nature.

In search of a simpler and more modular approach to composition, we introduce a natural strengthening of the PCP verifier, which we call an *assignment tester*. Intuitively, an assignment tester verifies not only satisfiability, but satisfiability *by a specified assignment* (given as oracle). This will provide an alternative and simple way of ensuring consistency in the context of PCP composition.

Assignment testers. An assignment tester is a polynomial-time reduction, whose input is a circuit φ over a set of variables X, and whose output is a list of polynomially many significantly smaller circuits $\Psi = \{\psi_1, \ldots, \psi_{\mathsf{R}}\}$ over *both* X and auxiliary variables Y. The guarantee of the reduction (for a complete definition, see Definition 3.1) is that for every possible assignment $a : X \to \{0,1\}$,

- (completeness) if a satisfies φ, then there is an extension of a, namely an assignment b for Y, such that all of $\psi_1, \ldots, \psi_{\mathsf{R}}$ are satisfied by $a \cup b$;
- (soundness) if a is "far" from any satisfying assignment for φ, then every extension of a to Y can satisfy at most 1% of $\psi_1, \ldots, \psi_{\mathsf{R}}$.

Thus, even if φ is satisfiable, but not by anything close to a, then 99% of $\psi_1, \ldots, \psi_{\mathsf{R}}$ must reject any extension of a. An intuitive way to understand the difference between a standard PCP verifier and an assignment tester is the following. Given a statement (a predicate φ claimed to be satisfiable) and an alleged proof for it (an assignment for X), the verifier checks that the statement is correct. In contrast, the assignment tester checks that *the proof* is correct. A related notion was independently introduced by Ben-Sasson et al. [8]; see further discussion below.

With this notion, we proceed to prove a composition theorem that is truly modular. The main idea is the following. Previously, relations between ψ_i and ψ_j (for $i \neq j$) were lost upon reduction to Υ_i and Υ_j. Now, all of the R reductions (each reducing ψ_i to the system Υ_i) are correlated through having to refer to the same assignment for X. To carry out this intuition, we give a generic transformation of any assignment tester into a "robust" one (we discuss this notion below). Once the first (outer) assignment tester is robust, the naive composition is trivially sound.

1.2. Our contributions. To be able to discuss our results, let us first briefly state the parameters of assignment testers. Let R be the number of circuits output by the assignment tester, and let s upper bound their size. There are three additional parameters: the query complexity q (how many variables are read by each output circuit), the error-probability ε (what fraction of the output circuits may erroneously accept on a "no" input; $\varepsilon = 0.01$ in the above discussion), and the distance parameter δ (what distance of the input assignment from a satisfying assignment should make the tester reject). For simplicity, we will ignore these last three parameters in most of the following discussion, with the implicit understanding that any mention of an assignment tester means constant q, ε, and δ.

New proofs of versions of the PCP theorem. As we discuss below, this paper provides a variety of transformations on assignment testers, complementing the generic composition theorem already mentioned above. Armed with this "toolkit" of transformations, we consider various ways of composing assignment testers with the goal of reproving the PCP theorem. We now elaborate on two results that we obtain

in this manner.

An assignment tester with $\mathsf{s} = n$ (recall that s is the size of the circuits produced by the assignment tester reduction) is completely trivial (letting the output equal the input). Nevertheless, we prove that, given an assignment tester with $\mathsf{s} = n^{0.99}$, we can get s all the way down to a constant.

THEOREM 1.1 (informal statement). *Given an assignment tester with* $\mathsf{R} = n^{O(1)}$ *and* $\mathsf{s} = n^{0.99}$*, we construct an assignment tester with* $\mathsf{R} = n^{O(1)}$ *and* $\mathsf{s} = O(1)$.

The idea of the construction is to use the given assignment tester as a building block and to compose it with itself. The output circuits will have size $n^{0.99}, n^{(0.99)^2}$, $n^{(0.99)^3}$, and so on, $n^{(0.99)^t}$ after t compositions. So taking $t = \log\log n$ results in a polynomially long list of constant size circuits. However, since the composition step roughly sums up the error probabilities of the two components, going beyond a constant number of composition steps requires an additional ingredient. This is where we incorporate several of the assignment tester transformations mentioned above, to achieve error reduction. Particularly, we employ a new combinatorial aggregation technique (namely, introducing new "aggregate" variables that represent ℓ-tuples of old variables).

Our building block assignment tester (i.e., the one with $\mathsf{s} = n^{0.99}$) can be constructed using *algebraic* techniques (for example, such an assignment tester can be constructed via techniques already present in [4, 13][1]). Coming up with a combinatorial construction for such an assignment tester would yield a completely combinatorial proof of the PCP theorem. Subsequent to our work, Dinur [11] *directly* (i.e., not going through this building block) gave a completely combinatorial proof of the PCP theorem.

Next, we present a combinatorial construction of an assignment tester, which is quasi-polynomial. It gives a combinatorial proof for $NP \subseteq PCP[\text{polylog}, 1]$ and implies, for example, that approximating Max-3-SAT is quasi-NP-hard.

THEOREM 1.2 (informal statement). *There exists an explicit combinatorial construction of an assignment tester with* $\mathsf{R} = n^{\text{polylog}n}$ *and* $\mathsf{s} = O(1)$.

This construction does not rely on any algebraic techniques; rather, it is based on recursive applications of our combinatorial transformations. In particular the construction relies on a transformation that makes an assignment tester "oblivious," so that the proof locations read are independent of the statement that is being proven. This eliminates, in a rather surprising manner, the need for aggregation in a crucial point in the proof.

The starting point is the previous construction, again relying on $\log\log n$ steps of composition and recursion. However, to compensate for the lack of a powerful building block, the main idea is to construct it ourselves, recursively. Thus the main step involves constructing an assignment tester with $\mathsf{s} = \mathsf{n}^{\alpha}$ (for some constant $\alpha < 1$) relying on recursion.

The resulting construction of an assignment tester is analogous to the following recursive construction of error-correcting codes based on tensor products. First, put your n-bit input in a $\sqrt{n} \times \sqrt{n}$ matrix. Now, recursively apply an error-correcting code to each row. Then, making additional recursive calls, apply an error-correcting code to each column (of the new matrix). Finally, as the relative distance has slightly deteriorated by this process, one can use simple transformations to amplify it back. Our related construction, in the context of assignment testers, is naturally more delicate and requires new ideas. However, the overall structure is very similar.

[1] Taking a low degree extension with a constant dimension, as in [23].

We note that this construction makes use of a *constant-size* assignment tester, to facilitate the composition. That is, the construction relies on an assignment tester that is required to work only on inputs of size $\leq n_0$ for some large enough constant n_0. (The only requirement from this assignment tester is that it produces small enough circuits. Say, circuits smaller than $s_0 = (n_0)^{0.99}$.) As in a similar situation in [25], such an object can be obtained via an exhaustive search over a constant range. However, the only proofs for its existence that we know of rely on previous constructions of PCP verifiers. Nevertheless, we can take the constant-size assignment tester needed by the construction to be an instantiation (for constant-size inputs) of extremely inefficient constructions, e.g., a Long-code-based assignment tester [6].

Combinatorial transformations on assignment testers. As the above description of our constructions indicates, our main technique is perhaps the most basic technique in computer science, namely recursion. This is implemented through the basic composition of a relatively weak assignment tester with a (strong) assignment tester of smaller size that is inductively constructed. In particular, the composition theorem mentioned above gives a way for reducing the circuit size (s) of an assignment tester (from s_1 or s_2 to $\mathsf{s}_1 \circ \mathsf{s}_2$).

We study several generic transformations on assignment testers that serve to improve various other parameters. We describe combinatorial methods to reduce the error probability, the query complexity, and the distance parameter. While these transformations are required for our constructions to work, we believe they are interesting in their own right. In particular, they provide tradeoffs that allow us to focus our attention on the important parameters of an assignment tester (e.g., it allows us to focus on constructing assignment testers with constant error and constant distance parameter).

Robustness. We mentioned above that for our generic composition theorem to work, the first (outer) assignment tester needs to be converted into a "robust" one. This means that in the "no" case, not only does a $1 - \varepsilon$ fraction of the circuits $\psi_1, \ldots, \psi_{\mathsf{R}}$ reject, but moreover they "strongly" reject in that their input is at least δ-far from any satisfying input. More formally, recall that each circuit reads some q variables. These are both X-variables (the original input variables of the input circuit), which are bits, and also auxiliary Y-variables that may come from a larger alphabet. We interpret here the Y-variables as bit strings (i.e., chunks of bits that are always read as a whole), and therefore the input of each ψ_i can be viewed as a longer bit string which is the concatenation of all the variables it reads. It is now well defined to say that an assignment to the input variables of ψ_i is δ-far (in Hamming distance) from any satisfying input.

We show a simple generic transformation taking every assignment tester into a robust one, where the robustness parameter is inversely related to the number of variables read, q. Loosely, the transformation goes as follows: given the output $\psi_1, \ldots, \psi_{\mathsf{R}}$ of the tester, construct a revised output $\psi'_1, \ldots, \psi'_{\mathsf{R}}$ over new variables that are supposed to be the error-corrected version of the old variables. (Here too the Y-variables are interpreted as bit strings, and each X-variable is simply encoded by repetition.) Each ψ'_i now decodes its input variables and applies ψ_i on the decoded variables. Our transformation uses an off-the-shelf error-correcting code, and its efficiency depends on standard parameters of the code (essentially on its distance, its rate, and the complexity of decoding codewords).

With this generic transformation, we can completely ignore the notion of robustness in all other transformations and refer to it only in the composition theorem. We

note in passing that [8] has the same notion of robustness. However, in that context, even the modest cost of our generic transformation is impermissible. Therefore, [8] works with assignment testers that are already robust (and indeed, they name these objects "*robust* PCP of proximity"). This difference between our definitions is further discussed in section 7.

Distance reduction. The only new parameter of assignment testers not already used by PCP verifiers is their distance parameter δ (what distance of the input assignment from a satisfying assignment should make the tester reject). We provide a generic method for strengthening assignment testers so that they identify smaller deviations from satisfying assignments. This transformation comes at a fair cost in other parameters. The main idea is the following: On input circuit φ, over Boolean variables X, encode X with "amplified" distance (so that two assignments for X that are δ' far from each other will be encoded by assignments that are $\delta > \delta'$ apart), and let φ' be the corresponding circuit (that first decodes its input and then applies φ). Now, apply the original assignment tester on φ' (instead of φ).

Error reduction with aggregation. It is easy to reduce the error probability of an assignment tester by repetition: replace $\Psi = \{\psi_1, \ldots, \psi_{\mathsf{R}}\}$ by ANDs of all possible ℓ-tuples of circuits in Ψ. This reduces the error-probability from ε to ε^ℓ but causes an ℓ-fold increase in the number of queried variables. We cannot afford such an increase, as it hurts the effective "robustness" of the assignment tester, a crucial property for composition. We avoid this increase through aggregation (or alternatively, parallelization). That is, the introduction of new variables to represent ℓ-tuples of previous ones. Our new method of aggregation is purely combinatorial.

The original proof of the PCP theorem also contains an aggregation method, based on sophisticated algebraic techniques. One disadvantage of that method is that it causes a blow-up in the size of the domain of the variables, by a factor that is roughly $\log |X|$ (more precisely, it is the degree of the low degree representation of the assignment for X), which cannot be afforded in our context. In contrast, our combinatorial aggregation increases the variables by a factor independent of $|X|$. On the other hand, our combinatorial aggregation introduces $|X|^\ell$ new variables, even if we are interested in the values of only some t possible ℓ-tuples. This is much worse than the $poly(\ell, |X|, t)$ new variables and tests introduced by the algebraic aggregation. Nevertheless, this blow-up is tolerable in our context as we care only about $\ell = O(1)$ (since we only plan on reducing error from one constant to another).

The main idea of our aggregation is the following. When reading ℓ-tuples of variables, to simulate the AND of ℓ circuits we are faced with the difficulty that the assignment to the ℓ-tuples may not be consistent with any assignment to the original variables. This problem is exactly what makes the proof of Raz's parallel repetition theorem [24] so challenging. Nevertheless, in our context, we can tolerate a more modest drop in the error probability than in [24], and can afford adding a constant (independent of ℓ) number of queries. So our main idea is the following: simply add a few queries directly aimed at testing the consistency of the ℓ-tuples of variables. This simple idea results in a significantly simpler analysis. In particular, a sufficiently good consistency test was already provided by Goldreich and Safra [19]. We give direct and simple analysis of essentially the same test.

This is somewhat similar to the situation analyzed in the parallel repetition theorem of [24]. There too, all possible ℓ-tuples are considered, and an exponential decrease in the error-probability occurs. Our situation is different in several respects. Most importantly, for the outcome system to remain an assignment tester, some sort

of "decoding" of the ℓ-tuple variables is necessary. Fortunately, we can tolerate a more modest drop in the error probability and can afford more than two (but still a constant independent of ℓ) queries, as in the case of [24]. This makes our analysis significantly simpler, as follows.

Assuming that the ℓ-tuples are assigned consistently (i.e., a variable x is assigned consistently in all ℓ-tuples containing it), the AND circuits described above can be simulated with just a few accesses to the new ℓ-tuple variables. Separately, we ensure the consistency of the ℓ-tuples through a consistency test. The consistency test that we use (see Figure 3 in section 4) simply compares two random ℓ-tuples that intersect on some $\sqrt[3]{\ell}$ elements and rejects if there is any disagreement. This test was suggested by Goldreich and Safra [19], who were interested in derandomized versions of the test and analyzed the nonderandomized version only indirectly. We provide a direct analysis by appealing to the expansion of the underlying graph. We prove that if $F : X^\ell \to \Sigma^\ell$ passes the test with high probability, then there is some $f : X \to \Sigma$, defined simply by plurality, such that F is close to $(f)^\ell$.

Related work. We have already mentioned that Ben-Sasson et al. [8] independently introduced an object called "PCP of proximity," which is essentially the same as our assignment tester, presented in a somewhat different language. The work of [8] shows connections of these objects to locally testable codes, and therefore the results of this paper seem relevant in this context as well. We further discuss this in section 7.

The notion of assignment testers is very related to the area of property testing. In fact, both assignment testers and PCPs of proximity can be viewed as special cases of more general definitions given by Ergün, Kumar, and Rubinfeld [12], in the context of proof-assisted testing. To the best of our knowledge, the connection to the construction of PCPs has not been explored in the past. The connection of assignment testers to property testing is elaborated upon in section 7.

As we discussed above, the motivation for defining assignment testers lies in the desire to obtain simple and modular composition of PCPs. This goal was already explored in the past. In particular, Szegedy [30] derived a syntactic composition theorem for abstractions of PCPs based on many valued logics.

Subsequent to our work, Dinur [11] described a combinatorial proof of the PCP theorem which also uses composition of assignment testers (and, in particular, composition with an assignment tester of constant size, just as in our second construction).

Organization. We formally define assignment testers in section 3, and then proceed to prove the composition theorem. In section 4 we provide generic transformations on assignment testers. Our two main constructions (Theorems 1.1 and 1.2) are proven in sections 5 and 6, respectively. The proof of the consistency test (required for the aggregation) can be found in Appendix A.

2. Preliminaries. In this section we define some standard combinatorial objects that our constructions rely upon. These are error-correcting codes and hitting sets. Both have a trivial random construction and can also be constructed explicitly.

2.1. Error-correcting codes. As in the original proof of the PCP theorem [2, 1], error-correcting codes are a very useful tool for our proof. However, unlike the original proof, we do not rely on algebraic properties of the codes, nor do we require the codes to be locally testable. The relevant parameters of the code in our case are rather generic: these are its rate, its distance, and the circuit complexity of verifying and decoding legitimate codewords. We call the last task "codeword decoding." In the next lemma we give the parameters of the codes that will imply the most elegant

version of our results. As we discuss below, such codes are easy to come by. We also note that much weaker codes still imply our main results.

LEMMA 2.1. *There exists a polynomial-time computable family of codes* $e = \{e_{\mathsf{w}} : \{0,1\}^{\mathsf{w}} \to \{0,1\}^{O(\mathsf{w})}\}_{\mathsf{w}\in\mathbb{N}}$ *such that* $e_{\mathsf{w}}(\cdot)$ *is a code with minimum distance* w *that satisfies the following:*

Linear circuit size for codeword decoding. *For every* w *there exists a circuit* C_{w} *of size* $O(\mathsf{w})$ *that takes as input a string* $z \in \{0,1\}^{O(\mathsf{w})}$ *and outputs* y *such that* $z = e_{\mathsf{w}}(y)$ *if such a string* y *exists and* $\perp$ *otherwise. Furthermore, the circuit* C_{w} *can be uniformly constructed in time poly*(w).

Lemma 2.1 asks for codes with constant rate and constant relative distance, which is quite standard. In addition, it requires linear circuit size for "codeword decoding," that is, for the task of verifying that a word is a legitimate codeword and then decoding it. This second part of decoding a legitimate codeword is trivial when $e_{\mathsf{w}}(y)$ contains y as a substring (such codes are sometimes called *systematic*).[2] Linear error-correcting codes can be assumed without loss of generality to be systematic. (By changing the basis of the generating matrix of the code, one can obtain a generating matrix for the *same code* that contains the identity matrix as a submatrix. Note that changing the basis doesn't change the code, so the parity check matrix is the same.) In addition, whenever a linear code is defined by a sparse parity check matrix (that contains only a linear number of nonzero entries), verifying that a word is a legitimate codeword can be performed by a circuit of linear size. In particular, Lemma 2.1 holds for the linear codes that are obtained by selecting a sparse parity check matrix, uniformly at random. In addition, it holds for the *explicit* codes best known under the name LDPC (low density parity check) codes. Note that the expander LDPC codes of [16, 31, 28, 29] have explicit combinatorial constructions based on the combinatorial construction of expander graphs in [25].

NOTATION 2.2. *We denote by* e_{w}^{-1} *the "maximum likelihood" decoding transformation that corresponds to the code* e_{w}. *That is,* $e_{\mathsf{w}}^{-1}(z') = y$ *if* $z = e_{\mathsf{w}}(y)$ *is the codeword of minimal Hamming distance to* z' (*where ties can be broken arbitrarily*). *In particular,* $e_{\mathsf{w}}^{-1}(e_{\mathsf{w}}(y)) = y$ (*assuming* $\mathsf{w} > 0$). *We do not assume anything on the computability of this mapping* (*in particular, we do not assume that it is polynomial-time computable*). *It is important to note that this maximum likelihood notion of decoding has little to do with "codeword decoding" as defined in Lemma* 2.1.

2.2. Hitting sets. We specify parameters of two families of sets with standard hitting properties. We then spell out in Corollary 2.5 the precise (standard) use of them for "error reduction." Both of these hitters can be constructed in an elementary way based on expander graphs. The hitters we give here rely on optimal expanders known as Ramanujan graphs [21]. We note that our main results can be proven using significantly weaker hitters (such as those implied by the expander graphs of [25]).

LEMMA 2.3. *Let* N *be an integer and* $0 \le \mu \le \alpha \le 1/2$ *two real values. Then there exists a family* $\mathcal{F} = \{F_1, \ldots, F_N\}$ *such that the following hold:*

[2] A natural approach for obtaining efficient codeword decoding is the following. First, slightly revise the error-correcting code such that it will be systematic (simply append y to $e_{\mathsf{w}}(y)$). Now codeword decoding is as easy as encoding: First, we can extract y from the alleged codeword, then re-encode y, and finally compare the result with the original alleged codeword. Unfortunately, error-correcting codes with linear size circuits for encoding are not known (and may very well not exist). Instead, one may use error-correcting codes with quasi-linear size circuits for encoding (these are not hard to come by). This natural approach can therefore give codes that are only slightly less efficient than those promised by Lemma 2.1 and are still good enough for obtaining the main results of this paper.

1. *Each F_i is a k-tuple of integers in $[N] = \{1, \ldots, N\}$, with $k = O(\alpha/\mu)$.*
2. *Every subset $T \subset [N]$ of size at least μN intersects at least αN of the F_i's (i.e., $|\{i : F_i \cap T \neq \phi\}| \geq \alpha N$).*
3. *Given N, i, α, and μ, each F_i can be constructed uniformly in polynomial time (in the input and output lengths).*

The set $\mathcal{F}$ in Lemma 2.3 can be constructed from a k-regular expander graph G_N on the set of vertices $[N]$. Each F_i is simply the neighbor list of vertex i. Requirement 2 asks for sets of size μN to expand by a factor α/μ, which for the expanders of [21] can be obtained with degree $k = O(\alpha/\mu)$. (For the expanders of [25] it requires $k = poly(\alpha/\mu)$.)

Next, we consider a more traditional setting of the hitting problem. For that we use the so-called combined hitter from Corollary C.5 in [17], as follows.

LEMMA 2.4. *Let N be an integer and β and μ be two positive real values. Then there exists a set $\mathcal{F} = \{F_1, \ldots, F_M\}$ of size $M = N \cdot poly(1/\beta)$ such that the following hold:*

1. *Each F_i is a k-tuple of integers in $[N]$, with $k = O(\log(1/\beta)/\mu)$.*
2. *Every subset $T \subset [N]$ of size at least μN intersects at least $(1-\beta)M$ of the F_i's (i.e., $|\{i : F_i \cap T \neq \phi\}| \geq (1-\beta)N$).*
3. *Given N, i, β, and μ, each F_i can be constructed uniformly in polynomial time (in the input and output lengths).*

The usefulness of Lemma 2.4 for this paper is in the standard application of hitters to error reduction. Particularly, we will use the following immediate corollary.

COROLLARY 2.5. *Let $\psi_1, \ldots, \psi_N$ be a sequence of N circuits over a set of variables Y. Let β and μ be two positive real values. Then there exists a sequence of $M = N \cdot poly(1/\beta)$ new circuits $\psi'_1, \ldots, \psi'_M$ such that we have the following:*

1. *Each new circuit ψ'_i is the AND of k old circuits ψ_i with $k = O(\log(1/\beta)/\mu)$. In particular, every assignment to the variables Y that satisfies all of the old circuits also satisfies all of the new circuits.*
2. *Every assignment to the variables Y that causes μN of the old circuits to reject also causes $(1-\beta)M$ of the new circuit to reject.*
3. *On input $\psi_1, \ldots, \psi_N$, β, and μ, the new sequence can be constructed uniformly in polynomial time (in the input and output lengths).*

3. Assignment testers and their composition. In this section we formally introduce the notion of an *assignment tester*, which is an enhancement of the PCP verifier. As discussed in the introduction, the motivation for assignment testers lies in the rather simple and natural way two assignment testers compose. This property is very appealing, as composition is a major ingredient in the proof of the PCP theorem.

Like the PCP verifier, an assignment tester reduces an input circuit φ over variables X into a list of output circuits $\psi_1, \ldots, \psi_{\mathsf{R}}$ over the variables (X and) Y. The main difference is that the output circuits of the PCP verifier might not depend on X at all, while the output circuits of the assignment tester certainly do. Moreover, the completeness and soundness conditions of an assignment tester are with respect to a specific assignment for X, rather than with respect to the general satisfiability of φ. Loosely, an assignment tester doesn't just check that the input is *satisfiable*, but rather that the input is *satisfied by a specified assignment*.

This simplifies composition by eliminating consistency issues altogether. Recall that the main idea of composition is to improve a given verifier reduction by applying another (inner) verifier reduction to each one of $\psi_1, \ldots, \psi_{\mathsf{R}}$, replacing each ψ_i with a system Υ_i of even smaller circuits. By feeding the same assignment to each of the

parallel runs of the inner reduction, all of the systems Υ_i directly refer to satisfiability by the same single assignment.

An important parameter, inherent to an assignment tester, is its *distance* parameter. In the soundness condition, it is unreasonable to require that in case the assignment for X is not satisfying, a sizeable fraction of $\psi_1, \ldots, \psi_{\mathsf{R}}$ reject. If we wish each ψ_i to read only a constant number of bits, most ψ_i's won't be sensitive to a single bit flip in X, turning a satisfying assignment into an unsatisfying one. Thus, we require only that if the assignment is δ-far (i.e., is at relative Hamming distance δ) from *every* satisfying assignment, then an $1-\varepsilon$ fraction of $\psi_1, \ldots, \psi_{\mathsf{R}}$ must reject. The parameter $\delta > 0$ is the distance parameter of the assignment tester, and it should be at least inversely proportional to the number of variables (unless we are willing to compromise on the detection probability $1-\varepsilon$ being subconstant).

In subsection 3.1 we give the formal definition of an assignment tester. In subsection 3.3 we define "robust" assignment testers and prove an immediate composition theorem for this object. We show a generic way to transform every assignment tester into a robust one ("robustization") in section 3.4. Finally, in subsection 3.5 we combine the above and deduce a composition theorem of assignment testers. We consider additional transformations on assignment testers in section 4. In section 7, we briefly discuss the relation between PCP testers and property testers.

3.1. Defining assignment testers. We denote Boolean circuits by φ, ψ, etc., and refer to the predicate computed by the circuit by the same name. We say that an assignment is δ-far from satisfying a circuit φ, if its relative Hamming distance from *every* satisfying assignment for φ is at least δ.

DEFINITION 3.1 (assignment tester). *An assignment tester with parameters* $(\mathsf{R}, \mathsf{s}, \mathsf{q}, \delta, \varepsilon)$ *is a reduction whose input is a Boolean circuit* φ *of size* n *over Boolean variables* X*. The reduction outputs a system of* $\mathsf{R}(\mathsf{n})$ *Boolean circuits* $\Psi = \{\psi_1, \ldots, \psi_{\mathsf{R}}\}$*, each of size at most* $\mathsf{s}(\mathsf{n})$ *over* X*, and auxiliary variables* Y *such that the following conditions hold:*

- *The running time of the algorithm is polynomial in* n *and* $\mathsf{R}(\mathsf{n})$*.*
- *Each* ψ_i *depends on* $\mathsf{q}(\mathsf{n})$ *variables from* $X \cup Y$*. The variables in* Y *take values in an alphabet* Σ *and are accessible to* ψ_i *as a tuple of* $\mathsf{w}(\mathsf{n}) = \lceil \log |\Sigma| \rceil$ *bits.*[3]
- *For every assignment* $a : X \to \{0,1\}$*,*
  1. [*completeness*] *if* a *satisfies* φ*, then there exists an assignment* $b : Y \to \Sigma$ *such that* $a \cup b$ *satisfies all of* $\psi_1, \ldots, \psi_{\mathsf{R}}$*;*
  2. [*soundness*] *if* a *is* δ*-far from every satisfying assignment for* φ*, then for every assignment* $b : Y \to \Sigma$*, at least* $1-\varepsilon$ *of* $\psi_1, \ldots, \psi_{\mathsf{R}}$ *reject* $a \cup b$*.*

Figure 1 gives a summary of the parameters.

This definition should be compared to the standard notion of a PCP verifier. The PCP verifier can also be defined as a reduction[4] precisely as above, but there are two differences. One is superficial in that the PCP verifier produces circuits that depend only on the (new) Y variables. The main difference is in the completeness and soundness conditions, which in the case of the PCP verifier reduction are defined

[3] Thus accessing all $\mathsf{w}(\mathsf{n})$ bits of a single variable in Y counts as a single "query."

[4] The PCP verifier is usually described as a probabilistic polynomial-time algorithm that verifies a (PCP) proof by tossing r random coins and then probing the proof in some q locations. By considering the action of the verifier in parallel over all possible outcomes of the random coins, the verifier corresponds to a list of 2^r circuits (each over q input variables). Thus, the verifier can also be viewed as a (deterministic) reduction that outputs a list of circuits. We call this a PCP verifier reduction.

- $\mathsf{R}(n)$ - Number of output circuits. Reminiscent of the amount of Randomness of the verifier.
- $\mathsf{s}(\mathsf{n})$ - Size of output circuits.
- $\mathsf{q}(\mathsf{n})$ - Maximal number of variables read (or queried) in one circuit.
- $\delta(\mathsf{n})$ - Distance to a satisfying assignment.
- $\varepsilon(\mathsf{n})$ - Error probability: the fraction of circuits that erroneously accept a far-from-satisfying assignment. (Sometimes we consider the detection probability $\gamma = 1 - \varepsilon$, i.e., the remaining fraction of circuits that reject.)
- $\mathsf{w}(\mathsf{n})$ - "Width" of Y variables, i.e., log of alphabet size. This parameter plays a minor role in our discussion, and so we usually omit it. We note that w is smaller than s.

FIG. 1. *The parameters* $(\mathsf{R}, \mathsf{s}, \mathsf{q}, \delta, \varepsilon)$ *of an assignment tester.*

as follows:

1. [*completeness*] *if* φ *is satisfiable, then there exists an assignment* $b : Y \to \Sigma$ *that satisfies all of* $\psi_1, \ldots, \psi_{\mathsf{R}}$*;*
2. [*soundness*] *if* φ *is unsatisfiable, then for every assignment* $b : Y \to \Sigma$*, at least* $1 - \varepsilon$ *of* $\psi_1, \ldots, \psi_{\mathsf{R}}$ *reject.*

Every assignment tester is also a PCP verifier reduction. To see, for example, that the soundness condition carries over, observe that if the input φ is unsatisfiable, then any $a : X \to \{0,1\}$ is ($\delta = 1$)-far from all (nonexistent) satisfying assignments. Thus it cannot be extended with b so as to satisfy more than ε of $\psi_1, \ldots, \psi_{\mathsf{R}}$.

The converse is not necessarily true since in an arbitrary PCP verifier reduction we have no control over the dependence of $\psi_1, \ldots, \psi_{\mathsf{R}}$ on X. In particular, the output circuits may not even depend on the variables in X. Interestingly, the original proof of the PCP theorem implicitly constructs assignment testers rather than just PCP verifiers.

THEOREM 3.2 (the PCP theorem [2, 1]). *There is a polynomial-time PCP verifier algorithm* (*alternatively, assignment tester*) *with* $\mathsf{R}(\mathsf{n}) = \mathsf{n}^{O(1)}$ *and* $\mathsf{q}(\mathsf{n}) \cdot \mathsf{w}(\mathsf{n}) \le \mathsf{s}(\mathsf{n}) = O(1)$ *and constant* $0 < \varepsilon, \delta < 1$.

3.2. On the width of variables. An assignment tester produces circuits ψ_i that are defined over two different kinds of variables. The X variables are Boolean, whereas the auxiliary Y variables take value from a possibly larger alphabet Σ. We would like to think of the assignment to a Y variable as a w-long bit string. (Recall that $\mathsf{w}(\mathsf{n}) = \lceil \log |\Sigma| \rceil$ is the width of the Y-variables.) This allows us to view each ψ_i as a Boolean circuit (just as φ is), which is particularly important for the composition theorem (where we apply the inner assignment tester to the circuits ψ_i produced by the outer assignment tester). In particular, if ψ_i reads the assignment to q_x variables from X and to q_y variables from Y, then we view its input as a $(\mathsf{q}_x + \mathsf{q}_y \cdot \mathsf{w})$-long bit string that is the concatenation of the assignment to all of the variables it reads. Note that, since the size of each ψ_i is larger than the length of its input, we have that $\mathsf{s} > \mathsf{q}_x + \mathsf{q}_y \cdot \mathsf{w}$. We can now define the restriction of a global assignment to the input variables of a particular ψ_i. This definition will allow repetitions of variables.

DEFINITION 3.3. *Each output circuit in* $\{\psi_1, \ldots, \psi_{\mathsf{R}}\}$ *is defined to be a pair* $\langle C, \tau \rangle$*, where* C *is a circuit over local inputs* $v_1, v_2, \ldots, v_q$ *and* $\tau = (x_{i_1}, \ldots, x_{i_{q_x}}, y_{i_{q_x+1}}, \ldots, y_{i_q})$ *is a tuple of variables from* $X \cup Y$ *specifying which variables are mapped to the inputs*

of the circuit. We emphasize that τ is allowed to have repetition of variables.[5]

Given an assignment σ for $X \cup Y$, its restriction to ψ_i is denoted $\sigma|_{\psi_i}$ and is defined as the appropriate string of $q_x + (q - q_x) \cdot w$ bits (that possibly reflects the repetitions in τ).

It turns out that, almost everywhere, the width parameter w plays only a very minor role. The two related parameters that will be much more crucial to our discussion are the size of the circuits s and the query complexity q. We will therefore almost always omit the reference w and be satisfied with the bound on w implied by s (as discussed above).

3.3. Robust assignment testers and their composition. A *robust* assignment tester is an assignment tester such that in the soundness case, not only do $1-\varepsilon$ of the output circuits reject, but in fact they see an assignment that is *ρ-far* from a satisfying one (i.e., at least a ρ-fraction of the bits read by each of these circuits need to be changed in order for the circuits to be satisfied). This variant is natural in the context of composition, as will be seen below.

Notation. For a circuit φ, denote by $SAT(\varphi)$ the set of all satisfying assignments for φ. Let $SAT_\delta(\varphi)$ be the set of assignments that are δ-close to some assignment in $SAT(\varphi)$ (namely, assignments that are at relative Hamming distance at most δ from some assignment in $SAT(\varphi)$).

DEFINITION 3.4. *An assignment tester is called ρ-robust if in the soundness case in Definition 3.1 above, for every assignment $b : Y \to \Sigma$, the assignment $(a \cup b)|_{\psi_i}$ is ρ-far from $SAT(\psi_i)$ for at least $1-\varepsilon$ fraction of $\psi_1, \ldots, \psi_{\mathsf{R}}$.*

It is very easy to compose robust assignment testers.

LEMMA 3.5. *Let $\mathcal{A}_1, \mathcal{A}_2$ be two assignment testers with parameters $(\mathsf{R}_1, \mathsf{s}_1, \mathsf{q}_1, \delta_1, \varepsilon_1)$ and $(\mathsf{R}_2, \mathsf{s}_2, \mathsf{q}_2, \delta_2, \varepsilon_2)$, respectively. If $\mathcal{A}_1$ is ρ-robust with $\rho = \delta_2$, then one can construct an assignment tester $\mathcal{A}_3$ with parameters $(\mathsf{R}_3, \mathsf{s}_3, \mathsf{q}_3, \delta_3, \varepsilon_3)$ such that*

$$\mathsf{R}_3(\mathsf{n}) = \mathsf{R}_1(\mathsf{n}) \cdot \mathsf{R}_2(\mathsf{s}_1(\mathsf{n})), \quad \mathsf{s}_3(\mathsf{n}) = \mathsf{s}_2(\mathsf{s}_1(\mathsf{n})), \quad \mathsf{q}_3(\mathsf{n}) = \mathsf{q}_2(\mathsf{s}_1(\mathsf{n})),$$

and

$$\varepsilon_3(\mathsf{n}) = \varepsilon_1(\mathsf{n}) + \varepsilon_2(\mathsf{s}_1(\mathsf{n})) - \varepsilon_1(\mathsf{n})\varepsilon_2(\mathsf{s}_1(\mathsf{n})), \quad \delta_3(\mathsf{n}) = \delta_1(\mathsf{n}).$$

Moreover, if $\mathcal{A}_2$ is ρ_2-robust, then so is $\mathcal{A}_3$.

Proof. Given an input φ, the tester $\mathcal{A}_3$ will simply run $\mathcal{A}_1$ on it, outputting $\psi_1, \ldots, \psi_{\mathsf{R}}$, and then run $\mathcal{A}_2$ on each ψ_i. The completeness and the parameters of $\mathcal{A}_3$ follow from the definition. The soundness of $\mathcal{A}_3$ draws on the robustness of $\mathcal{A}_1$ in the following way. Let $\{\psi_{i,j}\}$ be the list of circuits output by $\mathcal{A}_2$ on input ψ_i. The soundness of $\mathcal{A}_2$ asserts that a $1-\varepsilon_2$ of the $\{\psi_{i,j}\}$ reject if the assignment for ψ_i's variables *is far from* a satisfying one. The robustness of $\mathcal{A}_1$ guarantees that this is indeed the case for $1-\varepsilon_1$ of the ψ_i's, provided that the assignment for φ's variables is far from a satisfying one. ∎

3.4. Robustization. We next show a generic way to transform an arbitrary assignment tester into a robust one. The idea is to replace each variable in Y with a collection of bits that are supposed to be an encoding via some error-correcting code e of the value of the variable. In addition, we repeat the X variables for balance and modify the output circuits accordingly.

[5] Such repetition will be quite useful below: It implicitly affects the *distance* between the restrictions of assignments to ψ_i, turning it into a weighted Hamming distance (more weight to the repeated variable).

LEMMA 3.6. *There exists some $c_1 > 0$ such that, given an assignment tester $\mathcal{A}$ with parameters $(\mathsf{R}, \mathsf{s}, \mathsf{q}, \delta, \varepsilon)$, we can construct a ρ-robust assignment tester $\mathcal{A}'$ with parameters $\mathsf{R}' = \mathsf{R},\ s' = c_1 \cdot \mathsf{s},\ \rho = \Omega(\frac{1}{\mathsf{q}}),\ \varepsilon' = \varepsilon,\ \delta' = \delta$.*

This transformation allows us to replace the condition in Lemma 3.5 about $\mathcal{A}_1$ being ρ-robust with a condition about its query complexity; see Theorem 3.7 below. Throughout the rest of the paper Lemma 3.6 is used only in the proof of Theorem 3.7. In fact there is no further mention of robustness, as the lemma allows us to restrict our attention to the query complexity parameter.

We also mention that this transformation is useless for the setting of [8], as they cannot afford the (superlinear) increase in the number of variables that is incurred here.

Proof. $\mathcal{A}'$ will run $\mathcal{A}$ on input φ and obtain output circuits $\psi_1, \ldots, \psi_{\mathsf{R}}$ over variables X and Y. Each ψ_i will be replaced by a "robust" circuit ψ_i', whose inputs are encodings (via some error-correcting code e) of the inputs to ψ_i.

Let Σ be the alphabet of the Y variables, and let $\mathsf{w} = \lceil \log |\Sigma| \rceil$ be the number of bits needed to represent a value in Σ. Let $e_{\mathsf{w}} : \Sigma \to \{0,1\}^{\ell}$ be an error-correcting code as in Lemma 2.1 (with $\ell = c \cdot \mathsf{w}$ and c a small absolute constant). For each $y \in Y$ introduce new Boolean variables $\bar{v}(y) = v_1(y), \ldots, v_\ell(y)$ supposedly representing y's encoding $e_{\mathsf{w}}(y)$. Denote these new sets of bits by

$$Y' = \bigcup_{y \in Y} \bar{v}(y).$$

Suppose the tuple of variables accessed by ψ_i is $(x_1, \ldots, x_{\mathsf{q}_x}, y_1, \ldots, y_{\mathsf{q}_y})$, with $\mathsf{q} = \mathsf{q}_x + \mathsf{q}_y$. Define a new circuit ψ_i' whose input consists of $\ell \cdot \mathsf{q}$ variables: the first $\ell \mathsf{q}_x$ variables are ℓ copies of each variable of $x_1, \ldots, x_{\mathsf{q}_x}$. The next $\ell \mathsf{q}_y$ variables are $\bar{v}(y_1), \ldots, \bar{v}(y_{\mathsf{q}_y})$. The circuit ψ_i' accepts an input if and only if it is a correct encoding of an input that would have satisfied ψ_i. More explicitly, ψ' accepts input $z_1, \ldots, z_{\ell \mathsf{q}} \in \{0,1\}^{\ell \mathsf{q}}$ if and only if (a) $z^i = (z_{i\mathsf{q}+1}, \ldots, z_{(i+1)\mathsf{q}})$ is the all-0 or all-1 string for $1 \le i \le \mathsf{q}_x$, (b) z^i is a legal codeword of e for $i > \mathsf{q}_x$, and (c) the values encoded by the z^i's satisfy ψ_i.

This completes the description of $\mathcal{A}'$, and we now prove its properties. Completeness is clear: A satisfying assignment $a : X \to \{0,1\}$ for φ can easily be extended by $b : Y' \to \{0,1\}$ so that $a \cup b$ satisfies all of $\{\psi_i'\}$.

What is the size of ψ_i'? Since for the code e_{w} there exists a linear size circuit for codeword decoding (see Lemma 2.1), then there exists some c_1 such that the size of each ψ_i' is bounded by $\mathsf{s}(\mathsf{n}) + O(\ell \mathsf{q}) \le c_1 \cdot \mathsf{s}(\mathsf{n})$.

Next, the soundness of $\mathcal{A}'$. Assume an assignment $a : X \to \{0,1\}$ that is δ-far from satisfying φ. The soundness of $\mathcal{A}$ guarantees that every $b : Y \to \Sigma$ extending a will be rejected by at least $1 - \varepsilon$ of the ψ_i's. What does this mean for the robust version ψ_i'? Consider an arbitrary assignment $b' : Y' \to \{0,1\}$. Such an assignment defines a "decoded" assignment $b'' : Y \to \Sigma$ by relying on the "maximum likelihood" decoding mapping e_{w}^{-1} of the code e_{w} (see Notation 2.2) as follows:

$$b''(y) : Y \to \Sigma, \qquad b''(y) \stackrel{def}{=} e_{\mathsf{w}}^{-1}(b'(v_1(y)), \ldots, b'(v_\ell(y))).$$

The soundness of $\mathcal{A}$ implies that at least $1 - \varepsilon$ of $\psi_1, \ldots, \psi_{\mathsf{R}}$ will reject $a \cup b''$, regardless of the assignment b'' to Y. For each rejecting ψ_i, at least one of the q variables it queries must be reassigned in order for it to accept. Consider now the corresponding ψ_i' and its assignment $\sigma = (a \cup b')|_{\psi_i'}$. Recall that by definition

(see also Definition 3.3), σ is an $\ell\mathsf{q}$-bit string. Clearly, σ will not satisfy ψ'_i. More importantly, there must be at least one of the q (ℓ-bit) blocks that need to be changed *into a different legal ℓ-bit block* in order to turn σ into a satisfying input: If this ℓ-bit block consists of Y'-variables, then it must be changed in more than half the code distance (i.e., $\frac{\mathsf{w}}{2}$) locations. If it consists of (repetitions of) an X-variable, then it must be changed in all ℓ bits. In any case, this means that $\mathcal{A}'$ is ρ-robust with $\rho = \frac{\min(\mathsf{w}/2,\ell)}{\mathsf{q}\cdot c\mathsf{w}} = \frac{1}{2c\mathsf{q}}$. □

It is important to note that the error-correcting code e we use to obtain the robustness property is not part of the definition of an assignment tester but rather part of the robustization transformation. Furthermore, a feature of this transformation is that e, defined in Lemma 2.1, is quite a generic error-correcting code. We do not rely on algebraic properties of e nor require it to be locally testable.

3.5. Generic composition of assignment testers. Combining Lemmas 3.6 and 3.5, we get the following convenient composition theorem.

THEOREM 3.7 (composition). *There exist some constants c_1, c_2 such that $\mathcal{A}_1, \mathcal{A}_2$ are two assignment testers with parameters $(\mathsf{R}_1, \mathsf{s}_1, \mathsf{q}_1, \delta_1, \varepsilon_1)$ and $(\mathsf{R}_2, \mathsf{s}_2, \mathsf{q}_2, \delta_2, \varepsilon_2)$, respectively. If $\delta_2 \le \frac{1}{c_2\cdot\mathsf{q}_1}$, then one can construct an assignment tester $\mathcal{A}_3$ with parameters $(\mathsf{R}_3, \mathsf{s}_3, \mathsf{q}_3, \delta_3, \varepsilon_3)$ such that, for $\mathsf{n}' \stackrel{def}{=} c_1 \cdot \mathsf{s}_1(\mathsf{n})$,*

$$\mathsf{R}_3(\mathsf{n}) = \mathsf{R}_1(\mathsf{n}) \cdot \mathsf{R}_2(\mathsf{n}'), \quad \mathsf{s}_3(\mathsf{n}) = \mathsf{s}_2(\mathsf{n}'), \quad \mathsf{q}_3(\mathsf{n}) = \mathsf{q}_2(\mathsf{n}'),$$

and

$$\varepsilon_3(\mathsf{n}) = \varepsilon_1(\mathsf{n}) + \varepsilon_2(\mathsf{n}') - \varepsilon_1(\mathsf{n})\varepsilon_2(\mathsf{n}'), \quad \delta_3(\mathsf{n}) = \delta_1(\mathsf{n}).$$

Comparing this composition theorem with the robust composition of Lemma 3.5, we see that the condition about $\mathcal{A}_1$ being robust has been removed, and the condition $\delta_2 \le \rho$ has been replaced by the condition $\delta_2 \le \frac{1}{c_2\cdot\mathsf{q}_2}$. The parameters are almost the same, except here $n' = c_1 \cdot \mathsf{s}_1(\mathsf{n})$ rather than $n' = \mathsf{s}_1(\mathsf{n})$ (and this slightly affects $\mathsf{R}_3, \mathsf{s}_3, \mathsf{q}_3$, and ε_3).

We mention that the parameters of $\mathcal{A}_3$ in Theorem 3.7 are very similar to those that would follow from a naive composition of two PCP verifiers, when in the soundness argument one ignores consistency issues altogether (imagining a prover that is "honest" with respect to consistency). In that sense, these parameters are essentially the best one could hope for in this type of composition.

Proof. We first turn $\mathcal{A}_1$ into a robust assignment tester $\mathcal{A}'_1$, using the transformation of Lemma 3.6. Thus, $\mathsf{n}' = c_1 \cdot \mathsf{s}_1(\mathsf{n})$ is the size of the output circuits of $\mathcal{A}'_1$. Next, we compose $\mathcal{A}'_1$ with $\mathcal{A}_2$ according to Lemma 3.5, obtaining $\mathcal{A}_3$. See also Figure 2 for an illustration of the two transformations combined. □

4. Transformations on assignment testers. The composition of assignment testers is mainly used as a tool for reducing the size, s, of the circuits that an assignment tester outputs. In this section we give general transformations for reducing (and thus improving) three additional parameters: (i) the tested *distance*, δ, from a satisfying assignment; (ii) the *error probability*, ε, in case of a far-from-satisfying assignment; and (iii) the number, q, of variables read by each output circuit.

Our motivation is threefold. First, these transformations will come in handy in our constructions of assignment testers. Second, given these transformations, it is fair to concentrate on the construction of testers for some constant values of δ and ε.

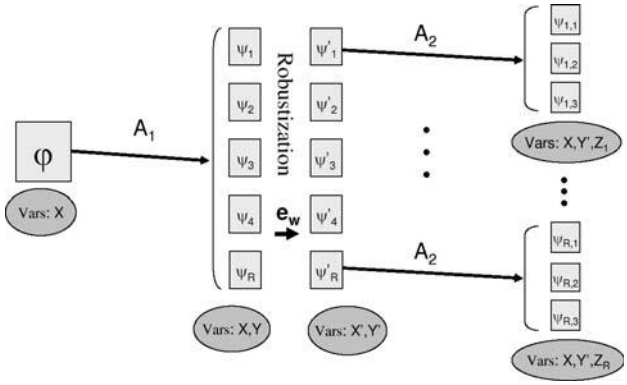


FIG. 2. *Composing* $\mathcal{A}_1$ *and* $\mathcal{A}_2$.

These parameters can be then reduced to the desired values using the general transformations. Finally, as we believe that the concept of assignment tester is interesting in its own right, it is natural to study the behavior of its various parameters.

To illustrate the need, in our context, for transformations that improve these $(\varepsilon, \mathsf{q}, \delta)$ parameters, note that composition may indeed reduce s, but it incurs costs in other parameters. In particular, it causes the error-probability, ε, to increase, as it is the sum of the error-probabilities of the two composed component. This is easily fixable since it is easy to reduce ε by increasing q. However, if we increase q, then we will require a smaller value of δ during composition in the next phase. The transformations of this section will help us essentially enhance the basic composition theorem such that it reduces s without harming other parameters.

4.1. Reducing the distance: $\delta \to \delta'$. In this subsection we describe how the distance parameter δ of a generic assignment tester can be improved (with fair cost in terms of the other parameters). The goal here is to improve the "sensitivity" of the assignment tester, so that the behavior of its output-circuits on mildly bad assignments (i.e., whose distance from satisfying is at least δ') imitates their behavior on very bad assignments (i.e., whose distance from satisfying is at least $\delta > \delta'$).

Our transformation is of "black-box" nature: We are given an assignment tester $\mathcal{A}$, whose inner workings we do not wish to manipulate, yet we want to reduce its distance parameter. A natural approach would be to manipulate the input variables X, creating a new set of variables X' (which would be part of the auxiliary variables) that encode X with amplified distance. This encoding would guarantee that two assignments for X that are δ'-apart are encoded by two assignments for X' that are δ-apart. Naturally, manipulating the input variables is not enough, and we also need to manipulate φ such that it "recognizes" the X' variables. Specifically, instead of applying $\mathcal{A}$ to φ, we apply it to φ' that is defined over X' (rather than over X), where φ' is defined to decode the assignment to X' and to apply φ to the decoded assignment (viewed as an assignment to X). But we are not done, as the circuits that $\mathcal{A}$ produces on φ' *do not even depend on the* X *variables* (as φ' does not depend on these variables either). We therefore augment these circuits by a verification that the assignment to X' correctly encodes the assignment to X. We will need to carefully define the encoding of the X variables so that verifying consistency between the X and X' variables will be sufficiently efficient.

LEMMA 4.1 (distance reduction). *There exists a positive constant $\bar{\delta}$ such that for every $0 < \delta' \leq \delta \leq \bar{\delta}$, given an assignment tester $\mathcal{A}$ with parameters $(\mathsf{R}, \mathsf{s}, \mathsf{q}, \delta, \varepsilon)$, we can construct an assignment tester $\mathcal{A}'$ with parameters $(\mathsf{R}', \mathsf{s}', \mathsf{q}', \delta', \varepsilon)$, where for $Q = O(\frac{1}{\delta'} \log \frac{1}{\varepsilon})$ and $M = O(\frac{\delta}{\delta'}\mathsf{n})$,*

$$\mathsf{R}'(\mathsf{n}) = poly(1/\varepsilon) \cdot \mathsf{R}(M), \quad \mathsf{s}' = \mathsf{s}(M) + Q, \quad \mathsf{q}'(\mathsf{n}) = \mathsf{q}(M) + Q.$$

Remark 4.2. Note that both the size of the new circuits s' and the number of queries q' contain an additive term of $O(\frac{1}{\delta'} \log \frac{1}{\varepsilon})$. This seems acceptable, as it is not hard to show that both s' and q' must be $\Omega(\frac{1}{\delta'} \log \frac{1}{\varepsilon})$ by the definition of assignment testers and lower bounds on the query complexity of hitters [17]. Consider, for example, an input circuit φ that is only satisfied by the all-zero assignment. For each of the circuits produced by the assignment tester consider the set of variables in X that it reads. For every subset T of variable of density δ', at least a $\gamma = 1 - \varepsilon$ fraction of the sets must hit T.

Proof. We follow the basic sketch outlined above. Let the input for our tester be a circuit φ over Boolean variables X. We fix an encoding E whose properties will be formally defined below. We let X' be new variables whose assignment supposedly represents the encoding via E of some assignment for X. We define the circuit φ' over X' to be a circuit that accepts only assignments for X' that encode (via E) an assignment for X that would have caused φ to accept. In other words, $SAT(\varphi') = E(SAT(\varphi))$. We choose E so that if an assignment $a : X \to \{0,1\}$ is δ'-far from the set $SAT(\varphi)$, then its encoding $b : X' \to \{0,1\}$ is δ-far (recall $\delta > \delta'$) from the set $SAT(\varphi')$.

Now, we run $\mathcal{A}$ on φ' and obtain a list of output circuits $\psi'_1, \ldots, \psi'_{\mathsf{R}(|\varphi'|)}$ over the variables X' and new variables Y. By the soundness of $\mathcal{A}$, starting with an assignment $b \notin SAT_\delta(\varphi')$ for X', no matter how one assigns the remaining Y variables, at most ε fraction of the ψ'_is accept.

It remains to add tests comparing the assignment b for X' to the assignment a for X. In order to be able to do this via circuits that make only few queries, X' must encode X in a *"locally checkable"* manner. Thus, the heart of our proof is the encoding E, whose properties are formalized next.

LEMMA 4.3. *Let $\bar{\delta}$ be a universal constant. For every $\delta_1 < \delta_2 < \bar{\delta}$, there exists a constant $c = O(\frac{\delta_2}{\delta_1}) \geq 1$ and an encoding $E : \{0,1\}^n \to \{0,1\}^{cn}$ that is polynomial-time computable, such that*

1. *if $a_1, a_2 \in \{0,1\}^n$, $\mathsf{dist}(a_1, a_2) > \delta_1$, then $\mathsf{dist}(E(a_1), E(a_2)) > \delta_2$;*
2. *there is a linear-time circuit that computes $E^{-1}(b)$ if b is in the image of E, and otherwise rejects;*
3. *there is a polynomial-time constructible collection of n circuits each of size $O(c) = O(\frac{\delta_2}{\delta_1})$ such that, given $a \in \{0,1\}^n$ and $b \in \{0,1\}^{cn}$, the following hold:*
   - *If $b = E(a)$, all of the circuits accept.*
   - *At least a $\mathsf{dist}(b, E(a))$ fraction of the circuits reject.*

Before proving Lemma 4.3, let us complete the description of $\mathcal{A}'$ and prove its properties. Let E be as in the lemma, choosing $\delta_1 = \delta'$ and $\delta_2 = 2\delta$. As a first step, $\mathcal{A}'$ will compute φ' from φ and generate $\psi'_1, \ldots, \psi'_{\mathsf{R}(|\varphi'|)}$, which are the outcome of running $\mathcal{A}$ on φ'. Recall that φ' is a circuit over variables X' that satisfies $SAT(\varphi') = E(SAT(\varphi))$. Due to the second item in Lemma 4.3, the size of φ', denoted M, is larger than the size of φ by a multiplicative factor $O(\frac{\delta}{\delta'})$. The number of circuits output by $\mathcal{A}$ is $\mathsf{R}(M)$, their size is $\mathsf{s}(M)$, and they read $\mathsf{q}(M)$ variables.

In addition, let $\{\mathsf{compare}_r\}_r$ be the collection of at most n circuits guaranteed by the third item of Lemma 4.3. We amplify the rejection probability of $\{\mathsf{compare}_r\}_r$ by derandomized serial error-reduction. We define a set of tests by applying Corollary 2.5 on the sequence $\{\mathsf{compare}_r\}$ with parameters $\mu = \delta$ and $\beta = \varepsilon$. Denote the new tests by $\{\mathsf{compare}'_1, \ldots, \mathsf{compare}'_{M_1}\}$. By Corollary 2.5, $M_1 = poly(1/\varepsilon)n$, and each $\mathsf{compare}'_i$ is the AND of $d = O(\frac{1}{\delta}\log\frac{1}{\varepsilon})$ $\mathsf{compare}$ tests. Therefore the size, denoted by Q, of each $\mathsf{compare}'_i$ satisfies $Q = O(c \cdot d) = O(\frac{1}{\delta'}\log\frac{1}{\varepsilon})$ (which also upper bounds the number of X and X' variables these circuits read).

The final output circuits of $\mathcal{A}'$ will be $\{(\psi_i \wedge \mathsf{compare}'_i)\}_i$, where by repetition we may assume an equal number of at most $poly(1/\varepsilon) \cdot \mathsf{R}(M)$ circuits of each type.

It is easy to check that $\mathcal{A}'$ has the claimed parameters, and it remains to prove the completeness and soundness of $\mathcal{A}'$.

Completeness is immediate: Given some $a \in SAT(\varphi)$, extend it to X'-variables by letting $b : X' \to \{0,1\}$ be defined by $b \stackrel{def}{=} E(a)$. The rest follows from the completeness of $\mathcal{A}$.

For soundness, one needs to show that, given any assignment for X that is δ'-far from a satisfying assignment, no assignment for the remaining variables $(X' \cup Y)$ can cause more than ε of the output circuits to accept. So let $a \notin SAT_{\delta'}(\varphi)$. Let $b : X' \to \{0,1\}$ and $c : Y \to \Sigma$ be arbitrary. There are two cases:

- If $\mathsf{dist}(E(a), b) \geq \delta$, then by Lemma 4.3 at least δ fraction of the circuits $\{\mathsf{compare}_r\}_r$ reject, so by construction and according to Corollary 2.5, at most ε of the circuits in $\{\mathsf{compare}'_i\}$ accept.
- Otherwise, $\mathsf{dist}(E(a), b) < \delta$. Then, using $\mathsf{dist}(a, SAT(\varphi)) > \delta'$, the first item in Lemma 4.3 implies $\mathsf{dist}(E(a), SAT(\varphi')) > 2\delta$. So by the triangle inequality, $\mathsf{dist}(b, SAT(\varphi')) > \delta$. The soundness of $\mathcal{A}$ implies that no matter what the assignment for Y, at most ε of the circuits ψ'_i accept.

This completes the proof of soundness, since in both cases, $a \notin SAT_{\delta'}(\varphi)$ allows at most an ε fraction of the final circuits to accept. Thus, assuming Lemma 4.3, we have proved Lemma 4.1. □

Proof of Lemma 4.3. Clearly, item 1 can be obtained using any error-correcting code. However, this would fail to give item 3, as given strings a and b, it is not clear how to check that $\mathsf{dist}(E(a), b)$ is small with few queries.[6] Instead, we will use an encoding that is not an error-correcting code in the standard sense but does have the desired amplification property.

Let $k = O(\frac{\delta_2}{\delta_1})$. We first describe an encoding over nonbinary alphabet $\Sigma = \{0,1\}^k$, denoted $E_0 : \{0,1\}^n \to \Sigma^N$. To encode a string $a \in \{0,1\}^n$, simply write its restriction on all possible k-bit substrings (so there are $N = n^k$ possible such restrictions). It is easy to see that this encoding achieves distance amplification. Indeed, two strings $a_1, a_2 \in \{0,1\}^n$ with $\mathsf{dist}(a_1, a_2) > \delta_1$, will differ on a $1-(1-\delta_1)^k \approx k\delta_1 > \delta_2$ fraction of the symbols.

Moreover, it is unnecessary to take all $N = n^k$ restrictions. With judicious choice of k-tuples, distance amplification will hold even with $N = n$ restrictions. Let X_k be an efficiently constructible hitting set of k-tuples of $[n]$ such that every subset of X of size $\geq \delta_1 n$ intersects at least a $c_1\delta_2$ fraction of the k-tuples in X_k (the constant $c_1 > 1$ will be chosen below). Lemma 2.3 guarantees such a set X_k with $|X_k| = n$.

[6] The question of testing whether a given word is the encoding of another word slightly resembles questions regarding the local testability or local decodability of the code. However, in this context we avoid the issue altogether.

This gives an encoding $E_1 : \{0,1\}^n \to \Sigma^{|X_k|}$. The choice of X_k guarantees that if $\mathsf{dist}(a, a') > \delta_1$, then $\mathsf{dist}(E_1(a), E_1(a')) > c_1\delta_2$, because at least $c_1\delta_2$ fraction of the tuples in X_k "hit" the set $\{\, i \mid a_i \neq a'_i\}$. Note that we are assuming (when applying Lemma 2.3) $c_1\delta_2 < 1/2$, which can be ensured by setting $\bar{\delta}$ to be a small enough constant.

The encoding E_1 is almost what we want, except that it is not binary. Thus, we concatenate E_1 (in the coding-theoretical sense) with the error-correcting code $e_k : \{0,1\}^k \to \{0,1\}^{c_2 \cdot k}$, given by Lemma 2.1. We now fix $c_1 > 0$ to be a constant such that e_k has relative distance $\geq 1/c_1$. Recall that e_k also has constant rate (i.e., c_2 is a constant) and linear size "codeword decoding" as defined in Lemma 2.1. Let $E \stackrel{def}{=} E_1 \circ e_k$ encode a string $a \in \{0,1\}^n$ by first computing $E_1(a)$ and then encoding each symbol of $E_1(a)$ using e_k. The distance of e_k being at least $1/c_1$ guarantees that

$$\mathsf{dist}(a, a') > \delta_1 \Rightarrow \mathsf{dist}(E_1(a), E_1(a')) > c_1\delta_2 \Rightarrow \mathsf{dist}(E(a), E(a')) > \delta_2,$$

which establishes item 1 of the lemma. For item 2, we use the fact that codeword decoding e_k needs circuits of size $O(k)$, and the fact that verifying that a string is an output of E_1 and then inverting E_1 on it is easy to do by a linear size circuit.

For item 3, we consider the following randomized test (which translates in the natural way to the required n circuits). The input is $a \in \{0,1\}^n$ and $b \in \{0,1\}^{c_2 kn}$, and we wish to verify that $b = E(a)$.

- Select a random k-tuple in X_k, and denote it by $(i_1, \ldots, i_k)$. Read $a_{i_1}, \ldots, a_{i_k}$.
- Let $j_1, \ldots j_{c_2 k}$ be the indices such that $b|_{j_1,\ldots,j_{c_1 k}}$ supposedly equals $e_k(a|_{i_1,\ldots,i_k})$. Read $b_{j_1}, \ldots, b_{j_{c_1 k}}$.
- Accept iff $e_k(a_{i_1}, \ldots, a_{i_k}) = b_{j_1}, \ldots, b_{j_{c_1 k}}$ (this is performed by first applying the circuit for codeword decoding of e_k on $b_{j_1}, \ldots, b_{j_{c_1 k}}$ and then comparing the result with $a_{i_1}, \ldots, a_{i_k}$).

The test requires $\log |X_k| = \log n$ random bits and reads $k + c_1 k = O(\frac{\delta_1}{\delta_2})$ bits. If $b = E(a)$, the test clearly accepts. Otherwise, denote $\beta = \mathsf{dist}(b, E(a))$. By construction of E_1, for at least a β fraction of the tuples $(i_1, \ldots, i_k) \in X_k$: $e_k(a_{i_1}, \ldots, a_{i_k}) \neq (b_{j_1}, \ldots, b_{j_{c_1 k}})$, so the test rejects with probability at least β. The test for any particular fixing of the $\log n$ random bits can be implemented by a circuit of size $O(k) = O(\frac{\delta_1}{\delta_2})$. □

Remark 4.4. In the construction provided in Lemma 4.1 of the new distance-reduced assignment tester $\mathcal{A}'$ for inputs of size n, we require only that $\mathcal{A}$ be well defined on inputs of size at most $M = O(\frac{\delta}{\delta'} n)$. This will be important for our inductive constructions, where $\mathcal{A}$ has been defined only for inputs of up to a certain size.

Remark 4.5. Lemma 4.1 will be used in this work only for constant δ. However, we will ignore, for the simplicity of presentation, the requirement that δ be smaller than some fixed constant $\bar{\delta}$. It is not hard to ensure that the lemma is applied only with sufficiently small δ.

4.2. Serial error-reduction: $\varepsilon \to \varepsilon'$. Let ε be the error-probability of an assignment tester $\mathcal{A}$, i.e., the fraction of output circuits that erroneously accept a far-from-satisfying assignment. The naive way of reducing ε is by serial repetition, namely, taking ANDs of k uniformly selected output circuits of $\mathcal{A}$ (as the output circuits of the new assignment tester $\mathcal{A}'$). The set of variables for $\mathcal{A}'$ is exactly the same as for $\mathcal{A}$. (Therefore the width of variables does not change by this transformation.) However, the number of variables read by each circuit increases by a factor k. This implies the following reduction.

LEMMA 4.6 (serial error-reduction). *For any integer ℓ, given an assignment tester $\mathcal{A}$ with parameters $(\mathsf{R},\mathsf{s},\mathsf{q},\delta,\varepsilon)$, we can construct a new assignment tester $\mathcal{A}'$ whose error probability is ε^ℓ. The other parameters of $\mathcal{A}'$ are $\mathsf{R}'(\mathsf{n}) = (\mathsf{R}(\mathsf{n}))^\ell$, $\mathsf{s}' = O(\ell\mathsf{s}(\mathsf{n}))$, $\mathsf{q}' = O(\ell\mathsf{q}), \delta' = \delta$.*

The number of circuits R' that $\mathcal{A}'$ outputs can be decreased by standard derandomization techniques. Particularly, based on Corollary 2.5, we obtain the following reduction.

LEMMA 4.7 (derandomized serial error-reduction). *Given an assignment tester $\mathcal{A}$ with parameters $(\mathsf{R},\mathsf{s},\mathsf{q},\delta,\varepsilon)$, we can construct a new assignment tester $\mathcal{A}'$ whose error probability is ε'. The other parameters of $\mathcal{A}'$ are $\mathsf{R}'(\mathsf{n}) = O(poly(1/\varepsilon')\mathsf{R}(\mathsf{n}))$, $\mathsf{s}' = O(\ell\mathsf{s}(\mathsf{n}))$, $\mathsf{q}' = O(\ell\mathsf{q}), \delta' = \delta$, where $\ell = \log(1/\varepsilon')/(1-\varepsilon)$.*

The main disadvantage of the serial error reduction (in both versions) is that it reduces ε at the expense of increasing the number of variables q that the circuits read. When error reduction will be used in our constructions, increasing q will not be acceptable. Therefore, we next give a new method for decreasing q back, by aggregation (i.e., by "consistently" reading some ℓ variables at once "in parallel").

4.3. Error-reduction via parallelization/aggregation. A central ingredient in the proof of the PCP theorem is the notion of aggregation (alternatively referred to as parallelization). Namely, starting with an assignment tester that reads q variables, we want to construct a new assignment tester that reads fewer $\mathsf{q}' < \mathsf{q}$ variables and is otherwise comparable to the old assignment tester (though typically the width of the new variables will be larger to compensate for the smaller number that are being read). One motivation for aggregation in our context is error-reduction, as discussed above. (Indeed, Theorem 4.8 below reduces both q and ε.) The original proof of the PCP theorem [2, 1] gives a very powerful aggregation method based on low-degree curves. The proofs in this paper will also require an aggregation theorem. However, our setting is significantly simpler, as we only need to reduce the number of variables read by the assignment tester from one *constant* q to a smaller constant q'. A "combinatorial" aggregation method for this setting can possibly be based on parallel repetition theorems. In particular, we could rely on the work of Feige and Kilian [14], as we do not require the full exponential decrease provided by [24]. We remark that since we require the result to be an assignment tester (rather than a verifier), some more details may be involved when applying [14, 24] in this context.

In this section, we present a simple alternative to both solutions. An advantage of our aggregation compared with the one based on curves is that it can produce variables of *constant width* (rather than logarithmic width). This feature is vital for our proofs. Compared to parallel repetition theorems, our setting is easier since we can afford more than two queries (but shall make a constant number of queries).

Note that the aggregation method of Theorem 4.8 below increases the distance parameter δ. This is the reason that the constructions of this paper require a method for reducing δ, as indeed given by the transformation of Lemma 4.1.

THEOREM 4.8 (aggregation). *Let $\mathcal{A}$ be an assignment tester with parameters $(\mathsf{R},\mathsf{s},\mathsf{q},\delta,\varepsilon)$. For every $\varepsilon' > 0$, one can construct a new assignment tester with parameters $(poly(1/\varepsilon')\cdot\mathsf{R}^\ell, \ell\cdot\mathsf{s}\cdot Q, Q, 2\delta, \varepsilon')$, for $Q = O(\frac{1}{\delta}\log(\frac{1}{\varepsilon'}))$, and $\ell = poly(\frac{\mathsf{q}}{1-\varepsilon})$.*

Before turning to the actual proof, let us sketch the idea. First, it is easy to reduce q to a constant ($\mathsf{q} = 3$ in our case), at the price of increasing the error. This is done by adding auxiliary variables which encode the entire input of a circuit, and then replacing that circuit by q circuits that compare the new "big" variable to one of the q original variables. (In fact, it will be convenient for us to compare to one

X-variable and also to one Y-variable, implying query complexity three rather than two.) This (standard) transformation will be described in detail later.

Next, the error probability can easily be reduced again by serial repetition, i.e., by taking ANDs of multiple circuits. However, the resulting circuits access more variables than before (i.e., by these two steps, q just got larger). To avoid this, it is natural to use *parallel repetition*. Namely, we introduce new variables $\overline{X} \equiv X^\ell$ that are supposed to be ℓ-tuples of the original variables X. Now, the effect of serial repetition can be emulated with only *few* accesses to the new (wider) variables. The main obstacle this approach must overcome is that of consistency. Suppose that $x \in X$ occurs in two tuples, $\bar{\mathbf{x}}$ and $\bar{\mathbf{x}}'$. Given an assignment $F : \overline{X} \to \Sigma^\ell$, it is quite possible that the value that x receives in $F(\bar{\mathbf{x}})$ differs from that in $F(\bar{\mathbf{x}}')$. If we were assured that there were no (or few) inconsistencies of this sort, then the analysis of parallel repetition would be as easy as that of serial repetition. We therefore address this issue by a consistency test (described in Figure 3) such that passing it with high probability guarantees that most tuples are "mostly" consistent with the plurality function of F, defined shortly (after specifying some notation and conventions).

For the sake of generality, and in order to simplify our analyses, we will define both the plurality function and the consistency test with respect to an arbitrary distribution $\mathcal{D}$ on X. Also for simplicity, we assume that we will always encounter assignments $F : \overline{X} \to \Sigma^\ell$ that are *rotation consistent*. Namely, if $\bar{\mathbf{x}}'$ is obtained from $\bar{\mathbf{x}}$ using some cyclic shift of its ℓ components, then $F(\bar{\mathbf{x}}')$ can be obtained from $F(\bar{\mathbf{x}})$ in the same way. This assumption slightly simplifies the consistency test and is easy to achieve in our setting using a trivial folding argument (simply let F be specified by the assignment for the various *subsets* of X of at most ℓ elements rather than by assignments to *ordered ℓ-tuples of elements*). Finally, for any ℓ tuple $\bar{\mathbf{x}}$, let $\bar{\mathbf{x}}_i$ denote its ith component.

DEFINITION 4.9 (plurality). *Let X and Σ be finite sets, let $\ell \geq 1$ be an integer, and set $\overline{X} \equiv X^\ell$. Let $\mathcal{D}$ be an arbitrary probability distribution over X. Let $F : \overline{X} \to \Sigma^\ell$. Define the* plurality *function of F with respect to $\mathcal{D}$, denoted $f_{F,\mathcal{D}} : X \to \Sigma$ as follows. For every $x \in X$, let $f_{F,\mathcal{D}}(x)$ be the value that is assigned to x most frequently by F (with respect to the distribution $\mathcal{D}^\ell$ on $\overline{X}$). Formally,*

$$\forall x \in X, \quad f_{F,\mathcal{D}}(x) \stackrel{def}{=} \max_{a \in \Sigma} \arg \left\{ \Pr_{\bar{\mathbf{x}} \in \mathcal{D}^\ell, i \in [\ell]} [F(\bar{\mathbf{x}})_i = a \mid \bar{\mathbf{x}}_i = x] \right\}.$$

In case $\mathcal{D}$ is the uniform distribution over X, we denote the plurality function as f_F (i.e., we omit $\mathcal{D}$ from the notation).

In Figure 3 we describe the consistency test. Essentially the same test was first studied by Goldreich and Safra [19], who proved that (for uniform $\mathcal{D}$) the test establishes consistency with some f. Their proof used a different "two-stage" plurality function f and was established via reduction to another (derandomized) test. In Appendix A we give a direct proof for the same test, summarized by the following theorem.

THEOREM 4.10 (consistency). *Let $F : \overline{X} \to \Sigma^\ell$ and $\mathcal{D}$ be an arbitrary probability distribution over X. Let $f = f_{F,\mathcal{D}} : X \to \Sigma$ be the plurality function of F. Let T be the test described in Figure 3. Then the following hold:*

- *If for all $\bar{\mathbf{x}} = (x_1, \ldots, x_\ell) \in \overline{X}$ it holds that $F(x_1, \ldots, x_\ell) = (f(x_1), \ldots, f(x_\ell))$, then T always accepts.*
- *Call $\bar{\mathbf{x}} = (x_1, \ldots, x_\ell) \in \overline{X}$ bad if $F(\bar{\mathbf{x}})$ disagrees with $(f(x_1), \ldots, f(x_\ell))$ on*

1. Select a random $\bar{\mathbf{x}} = (x_1, \ldots, x_\ell) \in \mathcal{D}^\ell$.
2. Select a random $\bar{\mathbf{x}}' = (x'_1, \ldots, x'_\ell) \in \overline{X}$ as follows: for each $j \in [\ell]$, $x'_j = x_j$ with probability $\alpha \stackrel{def}{=} \ell^{-1/3}$; otherwise x'_j is selected independently according to $\mathcal{D}$.
3. Accept only if $F(\bar{\mathbf{x}})$ agrees with $F(\bar{\mathbf{x}}')$ on *all* common variables; otherwise reject.

FIG. 3. *Consistency test for $F : \overline{X} \to \Sigma^\ell$, any $|X|, |\Sigma| < \infty$, and any probability distribution $\mathcal{D}$ over X. In case F is not guaranteed to be rotation consistent, then the test is revised slightly: The query $\bar{\mathbf{x}}'$ is also rotated cyclicly by a random shift in $[\ell]$.*

more than $\sqrt[3]{\ell}$ entries. There exists a constant c such that for every $\gamma > 0$,

$$\Pr_{\bar{\mathbf{x}} \in \mathcal{D}^\ell}[\bar{\mathbf{x}} \text{ bad}] > \gamma \Rightarrow T \text{ rejects with probability at least } \gamma/c.$$

Proof of Theorem 4.8. Our approach will be as follows. We first transform the circuits generated by $\mathcal{A}$ to circuits that each depend on three variables. This is a simple (almost standard) transformation that replaces every circuit reading q variables by several circuits each reading three variables (over a larger domain). This causes the error-probability to increase. Next, we perform ℓ-parallel repetition: Instead of reading 3ℓ variables, as in the serial repetition, we introduce new variables that are ℓ-tuples of the previous ones and read only three of these. We ensure soundness of this step by adding a separate consistency test. The number of circuits produced by the assignment tester will be bounded by $\mathsf{R}^{O(\ell \log \ell)}$. By redefining ℓ to be a slightly larger polynomial in $\frac{\mathsf{q}}{1-\varepsilon}$, the dependence on ℓ becomes as claimed.

Let $\mathcal{A}$ be an assignment tester with parameters $(\mathsf{R}, \mathsf{s}, \mathsf{q}, \delta, \varepsilon)$. Let $\gamma = 1 - \varepsilon$ be the detection probability of $\mathcal{A}$. We first fix $\gamma' = \delta/24c$, where c is the absolute constant from Theorem 4.10, and prove that one can construct a new assignment tester with parameters $(\mathsf{R}^{O(\ell \log \ell)}, O(\ell \cdot \mathsf{s}), O(1), 2\delta, \gamma')$, for $\varepsilon' = 1 - \gamma'$ and $\ell = O((\frac{\mathsf{q}}{\gamma})^{3/2})$. The theorem will then follow for an arbitrarily small ε' as a simple corollary of Lemma 4.7 (i.e., by serial error reduction). The proof will follow the two steps mentioned above.

Step 1: Getting to three queries. Let φ be a circuit over variables X. Run $\mathcal{A}$ on input φ, generating output circuits $\psi_1, \ldots, \psi_{\mathsf{R}}$ over X, Y. Define new variables $Z = \{z_1, \ldots, z_{\mathsf{R}}\}$, one z_i per ψ_i. The variable z_i will assume values that are supposedly the complete input to ψ_i. Notation: we denote by $\Sigma_X = \{0,1\}, \Sigma_Y, \Sigma_Z$ the set of values assumed by the variables in X, Y, Z, respectively. Thus, $\Sigma_Z \subseteq (\Sigma_X \cup \Sigma_Y)^{\mathsf{q}}$. We also denote by $X(z_i)$ (resp., $Y(z_i)$) the set of X- (resp., Y-) variables accessed by the corresponding circuit, ψ_i. Assume w.l.o.g. (without loss of generality) that these sets are nonempty for all i. Clearly, $|X(z) \cup Y(z)| \leq \mathsf{q}$ for all z.

By introducing the Z variables, we can get a system of circuits each reading three (rather than q) variables, but with a lower detection probability. This is a variant on a standard "transformation to two-provers" [15], and can be done, for example, as follows. Replace each ψ_i with circuits $\psi_i^{(1)}, \ldots, \psi_i^{(\mathsf{q})}$ that each read z_i, and also one $x \in X(z_i)$ and one $y \in Y(z_i)$, and then check that the value of z_i would have satisfied ψ_i, and that it is consistent with the values of x and y. (Each one of the $x \in X(z_i)$ and $y \in Y(z_i)$ is read by at least one of these circuits $\psi_i^{(\mathsf{q})}$.) We remark that each new circuit $\psi_i^{(j)}$ could have read two variables (as is more standard) rather than three: Instead of reading both an X-variable *and* a Y-variable (in addition to

the Z-variable), it could have read either an X- *or* a Y-variable. However, looking ahead, making three queries (each into a different set of variables) will be convenient for repetition. It will allow us, more naturally, to use three separate tables for ℓ-tuples of Z-, Y-, and X-variables. This way we will avoid having to deal with tuples mixing both X- and Y-variables.

Clearly if $\sigma|_{X\cup Y}$ satisfies all ψ_i, it can be extended to Z such that all of the $\psi_i^{(j)}$ accept. Also, we find the following result.

PROPOSITION 4.11. *Let $\sigma : X \cup Y \cup Z \to \Sigma_X \cup \Sigma_Y \cup \Sigma_Z$. If $\sigma|_X$ is δ-far from satisfying φ, then*

$$\Pr_{i,j}\left[\psi_i^{(j)} \text{ rejects}\right] \geq \gamma/\mathsf{q}.$$

Proof. By the soundness of $\mathcal{A}$, at least a γ fraction of the ψ_i's reject the assignment $\sigma|_{X\cup Y}$. No matter how one assigns the Z-variables, on these ψ_i's either $\sigma(z_i) \notin SAT(\psi_i)$ or there is an inconsistency between the value of z_i and at least one of the q variables accessed by ψ_i. We detect this inconsistency with probability at least $\frac{1}{\mathsf{q}}$. □

Step 2: Parallelization. We now define new variables that are tuples of the variables X, Y, Z above. Let $\ell = O((\frac{\mathsf{q}}{\gamma})^{3/2})$. For every possible ℓ-tuple of X-variables, we have a new variable $\bar{\mathbf{x}} = (x_1, \ldots, x_\ell) \in \overline{X}$, so that $\overline{X} \equiv X^\ell$. Similarly we define $\overline{Y} \equiv Y^\ell$ and $\overline{Z} \equiv Z^\ell$.[7]

We now define three types of building-block circuits. The first will perform ℓ-parallel repetition of the circuits $\{\psi_i^{(j)}\}$ in order to reduce their error. The second will facilitate the analysis of the parallel repetition by performing a consistency test of the assignment to the ℓ-tuples (as in Figure 3). The first two types test only the new variables that supposedly represent ℓ-tuples of original variables. We therefore still need to compare the assignment for the ℓ-tuples to the assignment for the original X-variables.

Parallel-repetition circuits. For every choice of an ℓ-tuple of circuits $\psi_{i_1}^{(j_1)}, \ldots, \psi_{i_\ell}^{(j_\ell)}$ in $\{\psi_i^{(j)}\}_{i\in[R],j\in[\mathsf{q}]}$ we will have one circuit simulating their ℓ-wise AND. For every $1 \leq t \leq \ell$ let x_t (resp., y_t, z_t) be the single x- (resp., y-, z-) variable accessed by $\psi_{i_t}^{(j_t)}$. The circuit will access $\bar{\mathbf{z}} = (z_1, \ldots, z_\ell) \in \overline{Z}$, $\bar{\mathbf{x}} = (x_1, \ldots, x_\ell)$, and $\bar{\mathbf{y}} = (y_1, \ldots, y_\ell)$ and accept if on every coordinate $1 \leq t \leq \ell$, z_t satisfies ψ_{i_t} and is consistent with x_t and y_t. (Recall that the assignment for each z_t is interpreted as an assignment for all of the variables of ψ_{i_t}, and in particular it specifies values for x_t and y_t.) Denote these circuits by $C_1^1, \ldots, C_{R_1}^1$ with $R_1 = (\mathsf{q}\mathsf{R})^\ell \leq R^{O(\ell \log \ell)}$ (in this bound the $\log \ell$ could have been dropped, assuming $\mathsf{q} < R$, but this makes no difference later so we leave it as is). Let $\mathsf{C}_1 = \{C_1^1, \ldots, C_{R_1}^1\}$.

Consistency circuits. Let $\mathcal{D}_x$ be the distribution on X defined by having $\Pr_{x\in\mathcal{D}_x}[x]$ equal the probability that a uniformly random $\psi_i^{(j)}$ circuit reads x. Define $\mathcal{D}_y, \mathcal{D}_z$ similarly. For all possible random choices of $\bar{\mathbf{x}}, \bar{\mathbf{x}}'$ as described in the consistency test in Figure 3, with $\mathcal{D} = \mathcal{D}_x$, we have a consistency circuit for the corresponding test. Likewise for all choices of $\bar{\mathbf{y}}, \bar{\mathbf{y}}'$ and of $\bar{\mathbf{z}}, \bar{\mathbf{z}}'$. Let $\mathsf{C}_2 = \{C_1^2, \ldots, C_{R_2}^2\}$ be the sequence of these circuits. The number of random choices is bounded by the number of pairs

[7] As discussed above, we ensure that the assignment to these ℓ-tuples is *rotation consistent*, by a simple folding argument. Instead of requiring an assignment to ℓ-tuples we ask for an assignment to subsets of cardinality at most ℓ. This naturally defines a rotation consistent assignment to the ℓ-tuples.

of circuit $\psi_i^{(j)}$ and variable x (or y) which is $\leq \mathsf{qR}$, raised to the power ℓ, and then at most squared. So, $R_2 \leq 3\,(\mathsf{qR})^{2\ell} \leq \mathsf{R}^{O(\ell \log \ell)}$. (Note that we have exactly the same number of circuits testing the X-variables, the Y-variables, and the Z-variables.)

Comparison circuits. These ensure consistency between $\overline{X}$ and X. For each $x \in X$ and $\bar{\mathbf{x}} \in \overline{X}$ that contains it, we have a comparison circuit, comparing the value of x, with the value given to it in the tuple $\bar{\mathbf{x}}$. Denote the resulting circuits $\mathsf{C}_3 = \{C_1^3, \ldots, C_{R_3}^3\}$, then $R_3 = \ell \mathsf{R}^\ell = \mathsf{R}^{O(\ell)}$ such circuits.

Final circuits. For simplicity, we assume $R_1 = R_2 = R_3 = R^{O(\ell \log \ell)}$ (otherwise, equality can be obtained by duplication). The final circuits output by our assignment tester will be the AND of the ith C_1 circuit with the ith C_2 circuit and with the ith C_3 circuit. That is, the ith output circuit is $C_i^1 \wedge C_i^2 \wedge C_i^3$. This guarantees that if ε' of the final circuits accept, then ε' of the C_1 circuits accept, as do ε' of the C_2 circuits and ε' of the C_3 circuits. Observe that these circuits are over variables $X \cup \overline{X} \cup \overline{Y} \cup \overline{Z}$.

This completes the description of the new assignment tester. Before we prove completeness and soundness, let us examine its parameters. The number of output circuits is indeed $R^{O(\ell \log \ell)}$. Each output circuit queries $O(1)$ variables. The maximum size of an output circuit is $\leq O(\ell \cdot \mathsf{s})$.

Completeness is immediate: Given an assignment to X that satisfies φ, it can be extended (due to the completeness of $\mathcal{A}$) to the Y variables so that all ψ_i are satisfied. This naturally extends to an assignment for Z, and then for $\overline{X}, \overline{Y}, \overline{Z}$.

LEMMA 4.12 (soundness). *Given an assignment σ for $X \cup \overline{X} \cup \overline{Y} \cup \overline{Z}$, if $\sigma|_X$ is 2δ-far from satisfying φ, then at least γ' of the output circuits reject.*

Proof. Let $F_x \stackrel{def}{=} \sigma|_{\overline{X}} : \overline{X} \to \Sigma_X$ be the restriction of σ to $\overline{X}$, and similarly, $F_y \stackrel{def}{=} \sigma|_{\overline{Y}}$ and $F_z \stackrel{def}{=} \sigma|_{\overline{Z}}$. Recall that $\mathcal{D}_x$ (resp., $\mathcal{D}_y, \mathcal{D}_z$) was the distribution according to which the consistency test of $\overline{X}$ (resp., $\overline{Y}, \overline{Z}$) was performed. Let $f_x = f_{F_x, \mathcal{D}_x}$, $f_y = f_{F_y, \mathcal{D}_y}$, $f_z = f_{F_z, \mathcal{D}_z}$ be the plurality functions of F_x, F_y, F_z, respectively, as in Definition 4.9. In the following proof the reader may wish to regard $\mathcal{D}_x, \mathcal{D}_y, \mathcal{D}_z$ as being uniform, although in general this need not be the case. We will, however, require that $\mathcal{D}_x$ be uniform, so let us explain why this can be assumed with no loss of generality. The distribution $\mathcal{D}_x$ would be uniform if the sequence of $\psi_i^{(j)}$ circuits access each X-variable the same number of times. To enforce this condition, we slightly modify the ψ_i circuits. We introduce a new set of variables X' that is supposed to be the exact copy of X, and apply the ψ_i circuits on the assignment to X' (instead of the assignment to X). In addition, we add consistency checks to test that the assignments to X and to X' are δ-close, by checking equality between a random variable in X and its copy in X'. It is easy to argue that the new circuits still define an assignment tester (with a small increase in the distance parameter, but otherwise parameters almost identical to the original one, $\mathcal{A}$). In addition, since the consistency checks between X and X' access each X-variable the same number of times, we get the additional desired property.

Recall that $\sigma|_X$ is 2δ-far from satisfying φ. We prove that either γ' of the circuits in C_3 reject, or γ' of the circuits in C_2 reject, or γ' of the circuits in C_1 reject.

PROPOSITION 4.13. *If $\mathsf{dist}_X(f_x, SAT(\varphi)) \leq \delta$, then γ' of the circuits in C_3 reject.*

Proof. If $\mathsf{dist}_X(f_x, SAT(\varphi)) \leq \delta$, then since $\mathsf{dist}_X(\sigma|_X, SAT(\varphi)) > 2\delta$, the triangle inequality implies $\mathsf{dist}_X(f_x, \sigma|_X) > \delta$. Let $x \in X$ be such that $\sigma(x) \neq f_x(x)$. By definition of plurality (and since $\mathcal{D}_x$ is the uniform distribution!), $f_x = f_{F_x}(x)$ is defined to be the value that is assigned most often to x, among all tuples that contain

x. Thus, assuming $f_x \neq \sigma(x)$, the value of $\sigma(x)$ must occur in no more than half of the tuples containing x. Thus, at least half of the comparison circuits (in C_3) that access x reject; altogether, $\frac{\delta}{2} > \gamma'$ of all of the comparison circuits reject. □

Recall that an ℓ-tuple $\bar{\mathbf{x}} \in \overline{X}$ is *bad* with respect to F_x if $F_x(\bar{\mathbf{x}})$ disagrees with $(f_x(x_1), \ldots, f_x(x_\ell))$ on more than $\sqrt[3]{\ell}$ locations. A similar definition applies to bad tuples in $\overline{Y}$ and in $\overline{Z}$.

PROPOSITION 4.14. *If more than $\frac{1}{8}$ fraction of the tuples (in each of $\overline{X}, \overline{Y}$, or $\overline{Z}$) are bad, then at least γ' of the circuits in C_2 reject.*

Proof. Assume there are more than $\frac{1}{8}$ bad tuples with respect to F_x, F_y, or F_z; then Theorem 4.10 guarantees that for some absolute constant c the consistency test for the particular table will reject with probability $\geq \frac{1}{8c}$. Thus, at least $\frac{1}{24c} \geq \gamma'$ of the circuits in C_2 will reject. □

Now, assume that the conditions in both Propositions 4.13 and 4.14 fail to hold. This will enable us to argue that the circuits in C_1 are emulating the serial repetition of $\{\psi_i^{(j)}\}$, and to deduce that many of them reject. Failure of the condition of Proposition 4.13 means that $\mathsf{dist}_X(f_x, SAT(\varphi)) > \delta$. Together with Proposition 4.11, this implies that at least γ/q of the $\{\psi_i^{(j)}\}$ circuits reject the assignment $f_x \cup f_y \cup f_z$. Failure of the condition of Proposition 4.14 means that the variables $\overline{X}, \overline{Y}, \overline{Z}$ are *consistent* with the plurality assignments f_x, f_y, f_z. So, for example, reading the assignment for a tuple $(x_1, \ldots, x_\ell) \in \overline{X}$ "emulates" reading the assignment f_x in locations $x_1, \ldots, x_\ell$.

We now combine the above to deduce that at least a γ' fraction of C_1 reject.

Indeed, choose ℓ random circuits $\psi_{i_1}^{(j_1)}, \ldots, \psi_{i_\ell}^{(j_\ell)} \in \{\psi_i^{(j)}\}$, and let $\overline{\psi} = (\psi_{i_1}^{(j_1)}, \ldots, \psi_{i_\ell}^{(j_\ell)})$. Under assignment $f_x \cup f_y \cup f_z$, we expect to see at least $\ell \cdot \frac{\gamma}{\mathsf{q}}$ of $\overline{\psi}$'s components reject. Moreover, ℓ is chosen exactly so that this expectation is sufficiently above $3\sqrt[3]{\ell}$, so most tuples $\overline{\psi}$ see more than $3\sqrt[3]{\ell}$ rejecting components. Indeed the following proposition follows from a standard tail inequality.

PROPOSITION 4.15. *Assume the failure of the condition of Proposition* 4.13. *For some $\ell = O((\frac{\mathsf{q}}{\gamma})^{3/2})$,*

$$\Pr_{\overline{\psi}}\left[\overline{\psi} \text{ contains} \leq 3\sqrt[3]{\ell} \text{ circuits that reject } f_x \cup f_y \cup f_z\right] \leq 1/8.$$

We now examine which circuits in C_1 can possibly accept. A circuit $C \in \mathsf{C}_1$ corresponding to some $\overline{\psi} = (\psi_{i_1}^{(j_1)}, \ldots, \psi_{i_\ell}^{(j_\ell)})$ reads three variables $\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}$ and rejects unless the following hold:

1. At least one of $\bar{\mathbf{x}}, \bar{\mathbf{y}}$, or $\bar{\mathbf{z}}$ is bad, or
2. $\overline{\psi}$ has at most $3\sqrt[3]{\ell}$ components that reject $f_x \cup f_y \cup f_z$.

Indeed, otherwise there must be at least one coordinate $1 \leq t \leq \ell$ for which $\psi_{i_t}^{(j_t)}$ rejects, and also the tth coordinate of $F_x(\bar{\mathbf{x}})$ equals the plurality function $f_x(x_t)$, and similarly for $F_y(\bar{\mathbf{y}})$ and $F_z(\bar{\mathbf{z}})$; thus, by definition, C rejects.

Proposition 4.15 bounds the probability of the second item by $\frac{1}{8}$. The probability of the first item is bounded by the probability of hitting a bad tuple in any of the tables F_x, F_y, or F_z. Each of $\bar{\mathbf{x}}, \bar{\mathbf{y}}$, or $\bar{\mathbf{z}}$ is a random (according to the corresponding distribution $\mathcal{D}_x, \mathcal{D}_y, \mathcal{D}_z$) entry in the corresponding table. Therefore, by the bound on the number of bad tuples (Proposition 4.14), the probability of the first item does not exceed $3 \cdot \frac{1}{8}$. Altogether, the probability that $\overline{\psi}$ doesn't reject is bounded by $\frac{1}{8} + \frac{3}{8} = \frac{1}{2}$, which means that $\overline{\psi}$ rejects with probability at least $1/2 > \gamma'$.

In conclusion, at least γ' of the final output circuits reject, completing the proof of Lemma 4.12 (soundness). □

Now that we have established the soundness of the assignment tester, Theorem 4.8 follows. □

5. The PCP theorem—An alternate proof. In this section we give a new proof for the PCP theorem [2, 1]. Our proof involves composition, and a combination of the combinatorial transformations described in section 4, applied on a "weak" assignment tester, that is assumed to be given.

THEOREM 5.1 (weak tester). *For some constants $\beta > 0, c_1$, and q_β there exists an assignment tester $\mathcal{A}_\beta$ with the following parameters:*

- *number of output circuits,* $\mathsf{R}(\mathsf{n}) = \mathsf{n}^{c_1}$,
- *size of output circuits,* $\mathsf{s}(\mathsf{n}) = O(\mathsf{n}^{1-\beta})$,
- *number of queries,* $\mathsf{q}(\mathsf{n}) = q_\beta$,
- *error probability,* $\varepsilon(\mathsf{n}) = 0.1$,
- *distance,* $\delta(\mathsf{n}) = 0.1$.

Such an assignment tester (and even stronger) can be constructed, for example, by relying on algebraic techniques already present in [13], particularly by taking a low-degree extension with a *constant* number of dimensions (as opposed to the way it is usually invoked, with a logarithmic number of dimensions) and then performing the [22] "sum-check" procedure (while relying on a *weak* low-degree test). It is interesting to note that although an assignment tester seems stronger than just a PCP verifier, all known constructions give the stronger object.

Our main theorem in this section is the following.

THEOREM 1.1 (formal statement). *Assuming $\mathcal{A}_\beta$ as in Theorem* 5.1*, there exists an assignment tester $\mathcal{A}$ with the following parameters:*

- *number of output circuits,* $\mathsf{R}(\mathsf{n}) = poly(\mathsf{n})$,
- *size of output circuits,* $\mathsf{s}(\mathsf{n}) = O(1)$,
- *number of queries,* $\mathsf{q}(\mathsf{n}) = O(1)$,
- *error probability,* $\varepsilon(\mathsf{n}) = 0.1$,
- *distance,* $\delta(\mathsf{n}) = 0.1$.

The PCP theorem now follows as an immediate corollary, as given next.

COROLLARY 5.2 (PCP theorem). *Given a set of Boolean constraints $\psi_1, \ldots, \psi_{\mathsf{R}}$ over Boolean variables Z, such that each read a constant number of variables, it is NP-hard to distinguish between the following two cases:*

1. [*completeness*] *there is an assignment to Z satisfying all of $\psi_1, \ldots, \psi_{\mathsf{R}}$.*
2. [*soundness*] *every assignment to Z satisfies at most* 10% *of $\psi_1, \ldots, \psi_{\mathsf{R}}$.*

Proof. We reduce from SAT. On input of a SAT formula φ over variables X, feed it to the assignment tester algorithm $\mathcal{A}$ given by Theorem 1.1. The output is a list of $\mathsf{R}(|\varphi|)$ circuits $\psi_1, \ldots \psi_{\mathsf{R}}$ such that each reads a constant number of bits from $Z = X \cup Y$.

If φ is satisfiable, say by an assignment $a : X \to \{0,1\}$, then there is some b such that together $a \cup b$ satisfy all of $\psi_1, \ldots, \psi_{\mathsf{R}}$. If φ is not satisfiable, at most an $\varepsilon = 0.1$ fraction of $\{\psi_1, \ldots, \psi_{\mathsf{R}}\}$ can accept. Otherwise, the soundness condition of $\mathcal{A}$ implies that a is δ-close to a satisfying assignment for φ, and in particular it means that φ is satisfiable. □

The naive approach for proving Theorem 1.1, the main theorem of this section, is to compose $\mathcal{A}_\beta$ with itself $\log \log n$ times. Starting with an input circuit of size n, the output circuits have size $\mathsf{n}^{1-\beta}$, $(\mathsf{n}^{1-\beta})^{1-\beta} = \mathsf{n}^{(1-\beta)^2}$, and so on to $\mathsf{n}^{(1-\beta)^i}$ at step i. Therefore, the circuit size of the output circuits is constant for $i = O(\log \log n)$. While already achieving nontrivial parameters, this construction does not quite give the PCP theorem. The reason has to do with the error probability, or its complement, the

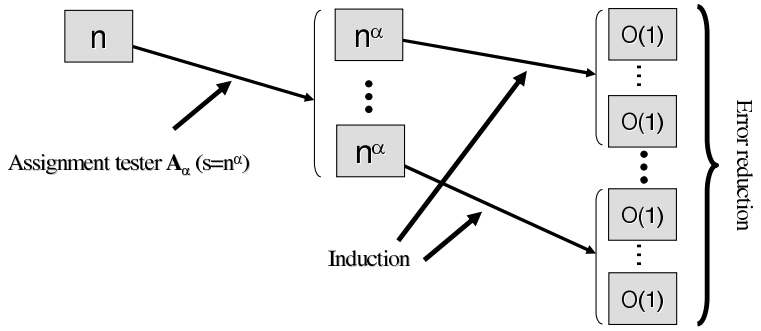


FIG. 4. *The inductive step. Error reduction encapsulates the additional parameter-fixing steps, which compensate for the error-increase of the composition step.*

detection probability ($\gamma = 1 - \varepsilon$), which will be more convenient to work with in this section. The detection probability becomes $\gamma_1\gamma_2, \ldots, \gamma_i = (\gamma_\beta)^{\log\log n} = O(1/\log n)$, rather than constant as we would like. Our proof of Theorem 1.1 will have the same general structure as in the naive approach but with a more elaborate recursive step, as follows.

It may seem at first that there is an additional more inherent problem with the naive approach. After all, to compose an assignment tester with itself we need $\delta < 1/\mathsf{q}$, which seems impossible. However, recall that the number of queries q of an assignment tester can be easily made a small fixed constant (e.g., 3) without harming the distance parameter δ (see Proposition 4.11).

Inductive step—Quick overview (*see also Figure* 4). Our construction is bottom-up. We assume having constructed an assignment tester with the desired parameters that works for inputs of size smaller than n, and show how to extend it for inputs of size n. Therefore, by induction, we obtain the desired assignment tester for all input sizes up to and including n.

Our inductive step begins with composition of the black-box assignment tester (or rather, a variant of it denoted A_α) with the one assumed by induction. This composition, as discussed above, increases the error of the assignment tester. To correct this we next perform error-reduction by "parallelization" (as in Theorem 4.8), which will take care of the error but will have two undesirable side effects: The distance parameter δ will increase, and the size of the output circuits will increase as well. The next step will be to perform a distance reduction (as in Lemma 4.1). Finally, the last step will be to reduce back the size of the output circuits, to achieve the inductive hypothesis. This is done by composition (again) but this time with a constant size assignment tester. (The inputs to this assignment tester will always be of some bounded constant size.) We note that this second assignment tester (called $\mathcal{A}_0$ below) can be obtained, e.g., by instantiation of the black-box tester (of Theorem 5.1) for a fixed input size.

Before proceeding to the formal proof, it will be convenient to strengthen somewhat the parameters of $\mathcal{A}_\beta$ given in Theorem 5.1.

COROLLARY 5.3. *Assuming $\mathcal{A}_\beta$ as in Theorem* 5.1*, there exists a constant $\bar{q}$ such that for every $\alpha > 0$ there exists c_1 and an assignment tester $\mathcal{A}_\alpha$ with the following parameters:* $\mathsf{R}(\mathsf{n}) = \mathsf{n}^{c_1}$, $\mathsf{s}(\mathsf{n}) = O(\mathsf{n}^\alpha)$, $\mathsf{q}(\mathsf{n}) = \bar{q}$, $\delta(\mathsf{n}) = 0.1$, $\varepsilon(\mathsf{n}) = 0.1$.

This strengthening follows by an appropriate sequence of transformations similar to those below. We defer the proof to Appendix B.

The rest of this section is devoted to proving Theorem 1.1 by following the outline

above. We first describe the building blocks used to construct the recursive algorithm $\mathcal{A}$, and then formally describe $\mathcal{A}$; see also the procedure in section 5.2.

5.1. Building-block testers and their parameters. As discussed above, we consider three testers in this construction. $\mathcal{A}$ itself, which is constructed recursively; $\mathcal{A}_\alpha$, which is essentially the assignment tester that we assume as a black box; and finally $\mathcal{A}_0$, the constant size assignment tester. It is important to note that there are various interdependencies between the parameters of these three testers, for example, to allow their composition (as Theorem 3.7 makes some restriction on parameters of the composed testers). The main purpose of the following definitions and discussion is to make explicit the dependencies between the various parameters, so as to make sure that we do not run into circular definitions. The reader may choose to ignore reading these subtleties initially and skip to the construction of $\mathcal{A}$ (provided in section 5.2).

First, $\mathcal{A}_\alpha$ (guaranteed by Corollary 5.3) has the parameters $(\mathsf{R}(\mathsf{n}) = \mathsf{n}^{c_1},\ \mathsf{s}(\mathsf{n}) = O(\mathsf{n}^\alpha),\ \mathsf{q}(\mathsf{n}) = \bar{q},\ \delta(\mathsf{n}) = 0.1,\ \varepsilon(\mathsf{n}) = 0.1)$. We will select α later and remember that c_1 is a constant and is the only parameter that depends on α. We stress that $\bar{q}$ is an absolute constant. Since $\mathcal{A}_\alpha$ will be applied only on input circuits of size larger than some n_0 (that can be made large enough), we can assume that $\mathsf{s}(\mathsf{n}) = \mathsf{n}^\alpha$. Denote the error probability and distance parameter by $\varepsilon_\alpha = \delta_\alpha = 0.1$, and the detection probability by $\gamma_\alpha = 1 - \varepsilon_\alpha = 0.9$.

Regarding the two additional testers $\mathcal{A}$ and $\mathcal{A}_0$, we first set their distance parameters in a way that will allow the composition needed by the construction. Set the distance parameter of $\mathcal{A}$ to be $\delta = \min(0.1, \frac{1}{c_2\bar{q}})$, where c_2 is the constant from Theorem 3.7. Recall that composition of two assignment testers is allowed (Theorem 3.7) if the distance parameter δ of the second (inner) is related to the query parameter $\bar{q}$ of the first, by $\delta \le \frac{1}{c_2\bar{q}}$. We remark that although in Theorem 1.1 the claimed distance parameter of $\mathcal{A}$ is 0.1, we constrain it here further, so that $\mathcal{A}$ can be invoked recursively. In foresight, set $\mathsf{q}_1 = O(\frac{1}{\delta}\log\frac{1}{\varepsilon_\alpha})$ (the constant in the "O" notation is an absolute constant that will be determined by the analysis below). Set $\delta_0 = \frac{1}{c_2\mathsf{q}_1}$. As we can set $\mathsf{q}_1 \ge \bar{q}$, we obtain $\delta_0 \le \delta$. We can now state the parameters of $\mathcal{A}_0$ (and prove its existence).

PROPOSITION 5.4. *For any $\alpha_0 > 0$ and any n_0 which is large enough (as a function of α_0 and δ_0), there exist $\mathsf{R}_0, \mathsf{s}_0$, and q_0 with $\mathsf{s}_0 < (n_0)^{\alpha_0}$, and there exists an assignment tester $\mathcal{A}_0$ such that for input circuits of size $\le \mathsf{n}_0$, $\mathcal{A}_0$ has parameters $(\mathsf{R}_0, \mathsf{s}_0, \mathsf{q}_0, \delta_0, \varepsilon_\alpha)$. (By definition, $q_0 \le s_0$.)*

Proof. Set $\alpha_0' = \alpha_0/3$, and let $\mathcal{A}_{\alpha_0'}$ be the assignment tester guaranteed by Corollary 5.3. $\mathcal{A}_0$ will be the result of applying the distance-reduction transformation (given by Lemma 4.1) to $\mathcal{A}_{\alpha_0'}$, reducing its distance parameter to δ_0. All we need to verify is the condition on s_0. By Corollary 5.3 and Lemma 4.1 we have that the size of the output circuits of $\mathcal{A}_0$ for input circuits of size $\le \mathsf{n}_0$ is at most $\mathsf{s}_0 = O(\mathsf{n}_0/\delta_0)^{\alpha_0'} + O(\frac{1}{\delta_0}\log\frac{1}{\varepsilon_\alpha})$. By setting n_0 to be large enough, we get that indeed $s_0 < (n_0)^{\alpha_0}$. ∎

In the proof we will need to set α_0 to be a small enough constant; this in turn will set n_0 and also R_0 to be large enough constants.

5.2. The recursive construction. The following lemma restates Theorem 1.1, giving explicit parameters for $\mathcal{A}$ that will be obtained by induction. The constants s_0 and q_0 are as in Proposition 5.4, and recall we set $\delta = \min(0.1, \frac{1}{c_2\bar{q}})$ and $\gamma_\alpha = 0.9$.

LEMMA 5.5. *There exists $C > 0$ and an assignment tester $\mathcal{A}$ with the following parameters: $\mathsf{R}(\mathsf{n}) \le \mathsf{n}^C$, $\mathsf{s}(\mathsf{n}) = \mathsf{s}_0, \mathsf{q}(\mathsf{n}) = \mathsf{q}_0, \delta(\mathsf{n}) = \delta, \varepsilon(\mathsf{n}) = 1 - (\gamma_\alpha)^2$.*

TABLE 1

Evolution of parameters during one inductive step. Recall that $\mathsf{q}_1 = O(\frac{1}{\delta}\log\frac{1}{\varepsilon_\alpha})$. *In addition,* $\mathsf{q}'_1 = O(\frac{1}{\delta_\alpha}\log\frac{1}{\varepsilon_\alpha})$ *and* $\ell = \mathrm{poly}(\frac{\mathsf{q}_0}{\gamma_\alpha})$. *The main composition step* (*step* I), *reduces the circuit size to* s_0 *at the price of reducing the detection probability from* $(\gamma_\alpha)^2$ *to* $(\gamma_\alpha)^3$. *The error-reduction step* (*step* II(a)), *increases the detection probability back* (*even above its final intended value*). *It is also crucial that the number of queries is also reduced, i.e.,* $\mathsf{q}'_1 < \mathsf{q}_0$. *However, both* s *and* δ *increase. This is corrected in steps* II(b) *and* II(c).

Name	Description	System-size (R)	Circuit-size (s)	q	Distance (δ)	Detection (γ)
$\mathcal{A}_\alpha$	Black-box	n^{c_1}	n^α	$\bar{q}$	δ_α	γ_α
$\mathcal{A}$	Induction	n^C	s_0	q_0	δ	$(\gamma_\alpha)^2$
$\mathcal{A}_0$	Const. size	R_0	s_0	q_0	δ_0	γ_α
$\mathcal{A}_I$	Reduce s	$\mathsf{n}^{c_1}\cdot O(\mathsf{n}^\alpha)^C$	s_0	q_0	δ_α	$(\gamma_\alpha)^3$
$\mathcal{A}_{IIa}$	& raise γ reduce q	$(\mathsf{n}^{c_1+\alpha C})^{O(\ell)}$	$\ell\cdot\mathsf{s}_0\cdot\mathsf{q}'_1$	q'_1	$2\delta_\alpha$	γ_α
$\mathcal{A}_{IIb}$	Reduce δ	$(\mathsf{n}^{c_1+\alpha C})^{O(\ell)}$	$\ell\cdot\mathsf{s}_0\cdot\mathsf{q}_1$	q_1	δ	γ_α
$\mathcal{A}_{IIc}$	Reduce s	$(\mathsf{n}^{c_1+\alpha C})^{O(\ell)}$	s_0	q_0	δ	$(\gamma_\alpha)^2$

Proof. We give a full description of $\mathcal{A}$ and prove (using induction) that $\mathcal{A}$ is well defined and has the desired final parameters. A brief outline of $\mathcal{A}$ is given next.

Algorithm 1. The Assignment Tester $\mathcal{A}$.

Let the input φ of $\mathcal{A}$ be a circuit of size n. If $|\varphi| \le \mathsf{n}_0$, then $\mathcal{A}$ just runs $\mathcal{A}_0$. Otherwise, we describe the algorithm $\mathcal{A}$ via several "intermediate" transformations ($\mathcal{A}_I \Rightarrow \mathcal{A}_{IIa} \Rightarrow \mathcal{A}_{IIb} \Rightarrow \mathcal{A}_{IIc} = \mathcal{A}$):

I. Initial composition (major size reduction): Let $\mathcal{A}_I = \mathcal{A}_\alpha \circ \mathcal{A}$ be the result of composing $\mathcal{A}_\alpha$ with a recursive invocation of $\mathcal{A}$ using Theorem 3.7. That is, the recursive invocation refers to circuits of size at most $\mathsf{s}(\mathsf{n}) = O(\mathsf{n}^\alpha)$.

II. Error-reduction:
   (a) (error and query reduction) Let $\mathcal{A}_{IIa}$ be the outcome of reducing the error parameter of $\mathcal{A}_I$ to ε_α using Theorem 4.8.
   (b) (distance reduction) Let $\mathcal{A}_{IIb}$ be the outcome of reducing the distance parameter of $\mathcal{A}_{IIa}$ to δ using Lemma 4.1.
   (c) (auxiliary size reduction) Let $\mathcal{A}_{IIc} = \mathcal{A}_{IIb} \circ \mathcal{A}_0$ be the composition of $\mathcal{A}_{IIb}$ with $\mathcal{A}_0$, again using Theorem 3.7, obtaining circuits of size s_0. $\mathcal{A}$ will run $\mathcal{A}_{IIc}$ on φ.

Let the input φ of $\mathcal{A}$ be a circuit of size n. If $|\varphi| \le \mathsf{n}_0$, then $\mathcal{A}$ just runs $\mathcal{A}_0$. In this case, the base of the induction is established by Proposition 5.4. As long as the constant C is chosen to be large enough so that $\mathsf{R}(\mathsf{n}_0) = \mathsf{n}_0^C \ge \mathsf{R}_0$, we have that the parameters of $\mathcal{A}_0$ (on input size at most n_0) are only better than required.

Assume by induction that Lemma 5.5 holds for all circuits of size smaller than n. (In fact, we will only use that it holds for circuits of size $\mathsf{s}(O(\mathsf{n})) = O(\mathsf{n}^\alpha)$.) We verify that each of the following steps (I, II(a), II(b), II(c) below) is well defined, and that we get the claimed parameters. Table 1 may assist the reader in following the evolving parameters.

I. $\mathcal{A}_I$ is the composition of $\mathcal{A}_\alpha$ with the recursive invocation of $\mathcal{A}$, using Theorem 3.7. This is well defined because $\mathcal{A}$ uses distance parameter $\delta \le \frac{1}{c_2\bar{q}}$ and since $\mathcal{A}$ is defined (inductively) for all inputs of size smaller than $\mathsf{s}(\mathsf{n}) < \mathsf{n}$. Based on Theorem 3.7, we have that $\mathcal{A}_I = \mathcal{A}_\alpha \circ \mathcal{A}$ produces $\mathsf{n}^{c_1}\cdot(O(\mathsf{n}^\alpha))^C$ circuits and has detection probability $\gamma_\alpha\cdot(\gamma_\alpha)^2$. As for the additional parameters, we have circuit size s_0, number of queries q_0, and error parameter δ_α.

II(a). We now apply error-reduction to $\mathcal{A}_I$ to increase the detection probability from $(\gamma_\alpha)^3$ to γ_α, using Theorem 4.8. For $\mathsf{q}'_1 = O(\frac{1}{\delta_\alpha}\log\frac{1}{\varepsilon_\alpha})$ and $\ell = \text{poly}(\frac{\mathsf{q}_0}{\gamma_\alpha})$, we have that the resulting assignment tester $\mathcal{A}_{IIa}$ produces $poly(1/\varepsilon_\alpha)\cdot(\mathsf{n}^{c_1+\alpha C})^{O(\ell)} = (\mathsf{n}^{c_1+\alpha C})^{O(\ell)}$ circuits. Its circuit size is $\ell\cdot\mathsf{s}_0\cdot\mathsf{q}'_1$, the number of queries is q'_1, the distance parameter is $2\delta_\alpha$, and the detection probability is γ_α, as desired.

II(b). Next, we get $\mathcal{A}_{IIb}$ by reducing the distance parameter of $\mathcal{A}_{IIa}$ from $2\delta_\alpha$ to δ according to Lemma 4.1. As discussed in Remark 4.4, this will require applying $\mathcal{A}_{IIa}$ on circuits of size $M = O(\frac{2\delta_\alpha}{\delta}\mathsf{n}) = O(\mathsf{n})$ (recall that $\delta = \Omega(1/\bar{q})$ is a constant). We note that as long as α is small enough, this will require invoking $\mathcal{A}$ recursively only on circuits of size smaller than $\mathsf{s}(M) < \mathsf{n}$. The number of circuits that $\mathcal{A}_{IIb}$ produces is $poly(1/\varepsilon_\alpha)(M^{c_1+\alpha C})^{O(\ell)} = (\mathsf{n}^{c_1+\alpha C})^{O(\ell)}$. The number of queries is $\mathsf{q}_1 = \mathsf{q}'_1 + O(\frac{1}{\delta}\log\frac{1}{\varepsilon_\alpha}) = O(\frac{1}{\delta}\log\frac{1}{\varepsilon_\alpha})$. Note that this is consistent with our previous definition of q_1. The circuit size is $\ell\cdot\mathsf{s}_0\cdot\mathsf{q}'_1 + O(\frac{1}{\delta}\log\frac{1}{\varepsilon_\alpha}) < \ell\cdot\mathsf{s}_0\cdot\mathsf{q}_1$ (by setting $\mathsf{q}_1 = O(\frac{1}{\delta}\log\frac{1}{\varepsilon_\alpha})$ to be large enough). The detection probability remains γ_α, and the distance parameter is δ, as desired.

II(c). Finally, for the composition $\mathcal{A}_{IIc} = \mathcal{A}_{IIb}\circ\mathcal{A}_0$ to be well defined we need to have $\mathsf{n}_0 \geq O(\ell\cdot\mathsf{s}_0\cdot\mathsf{q}_1)$. Note that $\ell\cdot\mathsf{s}_0\cdot\mathsf{q}_1 = poly(s_0)$. (There are hidden constants here which depend on $\delta_\alpha, \varepsilon_\alpha$, and $\bar{q}$.) Since $s_0 < (n_0)^{\alpha_0}$ the condition that $\mathsf{n}_0 \geq O(\ell\cdot\mathsf{s}_0\cdot\mathsf{q}_1)$ is satisfied, provided that α_0 is small enough (which translates to n_0 and R_0 being large enough). In addition, we have that δ_0 satisfies the condition of Theorem 3.7 and allows the composition of $\mathcal{A}_{IIb}$ with $\mathcal{A}_0$. The parameters obtained by $\mathcal{A}_{IIc}$ are the number of circuits $\mathsf{R}_0\cdot(\mathsf{n}^{c_1+\alpha C})^{O(\ell)} = (\mathsf{n}^{c_1+\alpha C})^{O(\ell)}$, the circuit size s_0, the number of queries q_0, the detection probability $(\gamma_\alpha)^2$, and the distance parameter δ.

It remains to verify that the parameters of $\mathcal{A}_{IIc}$ are as good as those required of $\mathcal{A}$ (and so $\mathcal{A}$ will simply simulate $\mathcal{A}_{IIc}$ on inputs with $|\varphi| > \mathsf{n}_0$, and this will complete the proof by induction). This already holds directly for all parameters apart from the number of circuits, which is supposed to be n^C. Therefore, the condition that is imposed is $(\mathsf{n}^{c_1+\alpha C})^{c'\cdot\ell} \leq \mathsf{n}^C$, for some constant c' derived from the analysis above. For that we set $\alpha < 1/(2c'\cdot\ell)$ (which causes the constant c_1 that controls the number of circuits output by $\mathcal{A}_\alpha$ to increase but still remain constant). The required condition now holds as long as $C > 2c_1\cdot c'\cdot\ell$. □

6. A combinatorial construction. In this section we describe a fully combinatorial construction of assignment testers with parameters $\mathsf{R} = n^{\text{polylog}n}$ and $\mathsf{s} = O(1)$. This implies a combinatorial proof for $NP \subseteq PCP[\text{polylog}, 1]$. Although not as strong as the PCP theorem, such a result still has quite powerful implications, for example that approximating Max-3-SAT is quasi-NP-hard.

In the construction of the assignment tester $\mathcal{A}$, we follow the recursion style of section 5. Namely, we compose an assignment tester $\mathcal{A}_\alpha$ that produces circuits of size $O(\mathsf{n}^\alpha)$, with a recursive call to $\mathcal{A}$ itself on circuits of this size. We then reduce the error of the resulting tester to maintain the induction hypothesis (as described in step II of Assignment Tester $\mathcal{A}$). The main difference is that here we can no longer assume the existence of $\mathcal{A}_\alpha$ as a black box that is given to the construction. We will therefore have to construct $\mathcal{A}_\alpha$ ourselves.

The first observation we make is the following: When defining $\mathcal{A}$ on input circuits of size n, we are allowed to assume that $\mathcal{A}$ is already well defined for inputs of size smaller than n. (In fact, we are already making this assumption in the construction of

section 5.) For this reason, we do not really have to build $\mathcal{A}_\alpha$ from scratch. Instead, in the definition of $\mathcal{A}_\alpha$ on inputs of size $\mathsf{n}' = O(\mathsf{n})$ we are allowed to invoke $\mathcal{A}$ itself, as long as we invoke it only on inputs that are smaller than n. In the presentation below we will concentrate on this task of constructing $\mathcal{A}_\alpha$ given $\mathcal{A}$ which is only defined on smaller inputs (formally stated in Lemma 6.2). This will include all of the new ideas not already described in section 5.

We note that, for the basis of the induction, we rely on a given constant-size tester, say like the one in Proposition 5.4. While in section 5 we obtained such a constant-size tester using the given black-box tester, this is no longer available. Instead we can choose any known PCP construction, even the extremely inefficient constructions of PCPs like that based on the long-code [6]. (As mentioned earlier, the constant-size assignment tester can also be found via exhaustive search, but for such an approach to work we need a proof of the *existence* of assignment testers, which is currently only known via *explicit* constructions.)

6.1. Overview–Constructing $\mathcal{A}_\alpha$ and oblivious testers. We now provide an overview of the construction of $\mathcal{A}_\alpha$. This will also allow us to introduce a new desired property of assignment testers that lies at the heart of our new construction; namely, we will discuss the notion of *oblivious* assignment testers.

Recall the task at hand: We would like to construct a tester $\mathcal{A}_\alpha$ that on input circuits of size n produces circuits of size $O(n^\alpha)$. Furthermore, we need $\mathcal{A}_\alpha$ to have constant query complexity. Namely, each of the circuits produced by $\mathcal{A}_\alpha$ should only read a constant number of variables. On the other hand, we have at our disposal a tester $\mathcal{A}$ that produces circuits of constant size (and in this respect is much better than what we need of $\mathcal{A}_\alpha$), albeit it is defined only on input circuits of size smaller than n.

Our approach is to decompose the input for $\mathcal{A}_\alpha$ into smaller pieces and apply $\mathcal{A}$ on each piece separately. If the decomposition has some nice properties, then we will be able to combine the outcomes of the different runs of $\mathcal{A}$ to obtain the desired $\mathcal{A}_\alpha$.

Ignoring various important issues (which we will soon address), the construction goes as follows: On an input circuit φ of size n, the first step will be to *decompose* φ into the disjunction of s_1 smaller circuits $\bigwedge_{\mathbf{i}} \varphi_{\mathbf{i}}$, each of size s_2. This decomposition is not necessarily syntactic, and in particular, $s_1 s_2 \gg \mathsf{n}$. The important property of this decomposition is that *there is a partition of φ's variables into blocks of size at most s_2, such that each $\varphi_{\mathbf{i}}$ accesses variables from at most $O(1)$ blocks.*

Now we apply the assignment tester $\mathcal{A}$ on each one of the $\varphi_{\mathbf{i}}$'s to obtain $\psi_{\mathbf{i},1}, \ldots, \psi_{\mathbf{i},\mathsf{R}(s_2)}$, where each $\psi_{\mathbf{i},k}$ has constant size. It may be useful to think of $\psi_{\mathbf{i},1}, \ldots, \psi_{\mathbf{i},\mathsf{R}(s_2)}$ as the $\mathbf{i}$th row of a matrix (with s_1 rows and $\mathsf{R}(s_2)$ columns). At this point, we will turn each column of this matrix into a new test. For the kth column we define the test $\psi^k = \bigwedge_{\mathbf{i}} \psi_{\mathbf{i},k}$. These new circuits have size $O(s_1)$, as they are composed of s_1 circuits of size $O(1)$. The transformation $\varphi \Rightarrow \{\psi^k\}$ defines the desired $\mathcal{A}_\alpha$. (In other words, on input φ, the tester $\mathcal{A}_\alpha$ outputs $\{\psi^k\}$.)

In the informal description above, we have ignored two major issues, to which we now draw attention.

6.1.1. Lack of robustness. This issue comes up when trying to argue soundness for $\mathcal{A}_\alpha$, say as a PCP verifier: In the case that φ is not satisfiable, then any assignment will fail to satisfy at least one of the $\varphi_{\mathbf{i}}$'s. We now want to argue that this implies that for this $\mathbf{i}$ many of the $\psi_{\mathbf{i},k}$'s will not be satisfied, which will finally imply that many of the ψ^k's will not be satisfied. Our problem is that, while the given assignment will not satisfy one of the $\varphi_{\mathbf{i}}$'s, it may be *close* to satisfying each and every one of them. In

that case, applying $\mathcal{A}$ does not guarantee anything regarding the $\psi_{\mathbf{i},k}$'s. Similarly to the proof for the composition theorem for assignment testers, we solve this difficulty by applying $\mathcal{A}$ on a "robust version" of the $\varphi_{\mathbf{i}}$'s rather than on the $\varphi_{\mathbf{i}}$'s themselves. It is possible to construct such a robust version, due to the special property of the decomposition (i.e., that each $\varphi_{\mathbf{i}}$ depends on a constant number of blocks of variables of φ). The robust circuits $\varphi'_{\mathbf{i}}$ will be defined such that when φ is not satisfiable then any assignment will be far from satisfying at least one of the $\varphi'_{\mathbf{i}}$'s. (In fact, we will need a bit more of the $\varphi'_{\mathbf{i}}$'s in order to argue soundness as an assignment tester rather than as a PCP verifier.)

We omit from this overview more specific details of how the $\varphi_{\mathbf{i}}$ and $\varphi'_{\mathbf{i}}$ circuits are defined and instead turn our attention to the second and potentially more devastating difficulty of our construction.

6.1.2. Huge query complexity. It is indeed true that the tester $\mathcal{A}_\alpha$ outlined above produces circuits ψ^k of size $O(s_1)$ as desired. However, in the general case, the query complexity of $\mathcal{A}_\alpha$ is also $O(s_1)$ rather than $O(1)$ as required. If we were able to partition the variables of $\mathcal{A}_\alpha$ into blocks of size $O(s_1)$, and ensure that the queries made by each ψ^k are contained in $O(1)$ "blocks" of variables, then this problem would be resolved by simply clustering these blocks into new larger variables (details below).

We obtain such a behavior from $\mathcal{A}_\alpha$ if the assignment tester $\mathcal{A}$ is *oblivious*. An oblivious assignment tester produces output circuits that have the property that *the names of the variables queried by each circuit are a function only of $|\varphi|$ and not of φ.* (Looking ahead, when such an $\mathcal{A}$ is applied on $\varphi_{\mathbf{i}}$ to yield $\{\varphi_{\mathbf{i},k}\}_k$, the names of the variables queried by $\varphi_{\mathbf{i},k}$ are the same for every $\mathbf{i}$.[8]) Formally, we have the following.

DEFINITION 6.1. *Let $\mathcal{A}$ be an assignment tester that, on input circuit φ over the variables $x_1, \ldots, x_{n_1}$, produces circuits $\psi_1, \ldots, \psi_{\mathsf{R}(\mathsf{n})}$ over the variables $x_1, \ldots, x_{n_1}$, $x_{n_1+1}, \ldots, x_{n_1+n_2}$. $\mathcal{A}$ is called* oblivious *if for every n there exist $\mathsf{q} = \mathsf{q}(\mathsf{n})$ functions*

$$\nu_1, \ldots, \nu_{\mathsf{q}} : \{1, \ldots, \mathsf{R}(\mathsf{n})\} \to \{1, \ldots, n_1 + n_2\}$$

such that for every $k \in [\mathsf{R}]$ the circuit ψ_k depends on the q variables indexed by $\nu_1(k), \ldots, \nu_{\mathsf{q}}(k)$.

In other words, the identity of the variables read by ψ_k depends on $|\varphi|$ but not on φ. Note that the *predicate* evaluated by each ψ_k may certainly depend on φ and not only on $|\varphi|$. Also note that we do not make any requirement on the efficiency of computing the ν_i's. (We point out, however, that their efficiency does follow from the efficiency of $\mathcal{A}$.)

Intuitively, the reason that obliviousness of $\mathcal{A}$ is useful in reducing the query complexity of $\mathcal{A}_\alpha$ is that it implies a very regular structure of the variables read by the $\psi_{\mathbf{i},k}$'s that compose the final circuit ψ^k. This implies a partition of the variables into blocks, as mentioned above, and allows us to cluster the variables read by the output circuits ψ^k into larger, $O(s_1)$-bit long, "column" variables, such that each ψ^k depends only on *a constant number* of those larger variables.

Finally it remains to address the typical case where $\mathcal{A}$ is *not oblivious*. Luckily, we show that it is not hard to turn *every assignment tester* into an oblivious one.

6.2. Constructing $\mathcal{A}_\alpha$. We now formalize the construction of $\mathcal{A}_\alpha$ for inputs of size N_α based on $\mathcal{A}$ which is only defined for input circuits whose size is smaller than N.

[8] Assuming for simplicity that $|\varphi_{\mathbf{i}}|$ is the same for all $\mathbf{i}$.

LEMMA 6.2. *For every $\alpha < \frac{1}{3}$, given an assignment tester $\mathcal{A}$ defined for inputs of size n such that $\mathsf{n} < N$, with parameters $\mathsf{R}(\mathsf{n}), \mathsf{s}(\mathsf{n}) = O(1), \mathsf{q}(\mathsf{n}) = O(1), \delta(\mathsf{n}) = O(1)$, and $\varepsilon(\mathsf{n}) = O(1)$, we can construct $\mathcal{A}_\alpha$ defined for inputs of size at most N_α, where N_α is such that $(N_\alpha)^{1-\alpha}\text{polylog}N_\alpha < N$, with circuit size $\mathsf{s}_\alpha(\mathsf{n}) = O(\mathsf{n}^{3\alpha})$, and other parameters*

$$\begin{aligned}
\mathsf{R}_\alpha(\mathsf{n}) &= \mathsf{R}(\mathsf{n}^{1-\alpha}\text{polylog}\mathsf{n}) \cdot \mathsf{n} \cdot \mathit{poly}(1/\varepsilon),\\
\mathsf{q}_\alpha(\mathsf{n}) &= O\left(\frac{1}{\delta}\log\frac{1}{\varepsilon}\right),\\
\delta_\alpha(\mathsf{n}) &= 24\delta,\\
\varepsilon_\alpha(\mathsf{n}) &= \varepsilon.
\end{aligned}$$

Note that the main advantage of $\mathcal{A}_\alpha$ over $\mathcal{A}$ is that it is defined for significantly larger inputs (i.e., we may have $N_\alpha \gg N$). In the application of Lemma 6.2, we will need it to be defined only for inputs of length at most $N_\alpha = N \cdot \text{polylog}N$. Also note that the assignment tester $\mathcal{A}_\alpha$ outputs circuits whose size is $O(\mathsf{n}^{3\alpha})$. In order to be perfectly consistent with the notation of section 5 we should have called it $\mathcal{A}_{3\alpha}$, but in order to simplify notation we call it $\mathcal{A}_\alpha$.

Proof. Let φ, $|\varphi| = \mathsf{n} \le N_\alpha$, be the input circuit over a set of variables X. First, we decompose it as the disjunction of smaller circuits $\varphi_{\mathbf{i}}$.

Decomposing φ. We begin with a standard transformation, taking our input circuit φ into a $3SAT$ formula $C_1 \wedge \cdots \wedge C_m$, where each variable appears only a constant number of times: For each internal gate of φ add a new variable. In addition, add new variables for multiple copies of X variables without degree larger than one (in the circuit φ). Some of the clauses C_i of the new $3SAT$ formula will correspond to gates in φ. These clauses verify that the assignment for any particular gate variable is consistent with the assignment for its input-variables (that may either be X-variables or other gate variables). In addition, for variables $y_{i_1}, y_{i_2}, y_{i_3}, \ldots$, that are supposed to be copies of a variable x_i, we have clauses verifying that $x_i = y_{i_1}$ and that $y_{i_j} = y_{i_{j+1}}$. Let Y denote the set of all new variables that were added. Clearly, $|X \cup Y| \le \mathsf{n} = |\varphi|$. The transformation also guarantees that an assignment to X could be extended to an assignment to $X \cup Y$ that satisfies $C_1 \wedge \cdots \wedge C_m$ if and only if the assignment to X satisfies φ.

Assume that $\mathsf{n}_1 = \mathsf{n}^{1-\alpha}$ is an integer. Arrange the variables in an n^α-by-n_1 matrix V such that the X-variables fill the first $\frac{|X|}{\mathsf{n}_1}$ rows. (For simplicity, assume that n_1 divides $|X|$.) Each row of V consists of n_1 variables. Denote the rows by V_i, $i = 1, \ldots, \mathsf{n}^\alpha$. See also Figure 5.

For every $\mathbf{i} = (i_1, i_2, i_3) \in [\mathsf{n}^\alpha]^3$, define $\varphi_{\mathbf{i}}$ to be the AND of all of the clauses C_i whose three variables are contained in $V_{i_1} \cup V_{i_2} \cup V_{i_3}$. Let us see that this is a decomposition of φ with the property mentioned in the overview. First note that

$$C_1 \wedge \cdots \wedge C_m = \bigwedge_{\mathbf{i}=(i_1,i_2,i_3)} \varphi_{\mathbf{i}}.$$

The number of circuits $\varphi_{\mathbf{i}}$ is $\mathsf{n}^{3\alpha}$. Each $\varphi_{\mathbf{i}}$ depends on (at most) three blocks (rows) of variables (i.e., the blocks V_{i_1}, V_{i_2}, and V_{i_3}), and the size of each block is at most $\mathsf{n}_1 = \mathsf{n}^{1-\alpha}$. Finally, the size of each $\varphi_{\mathbf{i}}$ is at most $O(\mathsf{n}_1)$ (since it depends on $3\mathsf{n}_1$ variables and each variable appears in a constant number of clauses C_i of the $3SAT$ formula). Going back to the overview, we have decomposed φ into $s_1 = \mathsf{n}^{3\alpha}$ circuits,

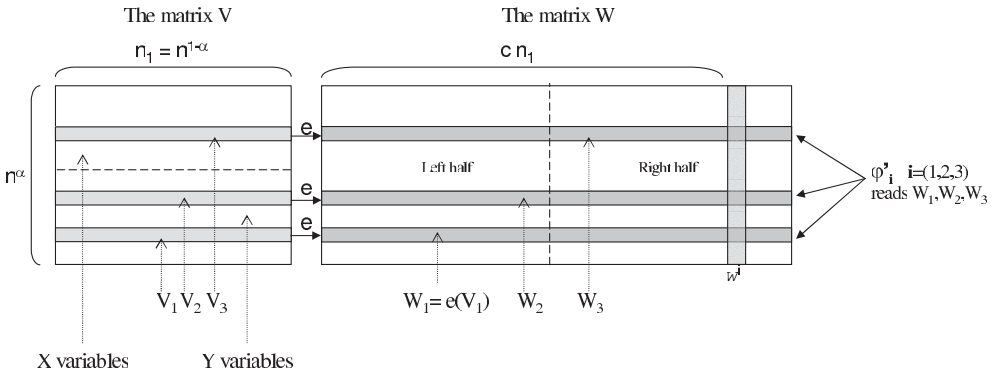


FIG. 5. *The variables $V = X \cup Y$ and W. The left half of W corresponds to encoding according to e_1, and the right half corresponds to encoding according to e_2.*

each of size at most $s_2 = O(\mathsf{n}^{1-\alpha})$, such that each circuit $\varphi_{\mathbf{i}}$ depends on three blocks of variables (and the size of each block is at most s_2).

Making the $\varphi_{\mathbf{i}}$'s robust. Next, we prepare the $\varphi_{\mathbf{i}}$'s for composition by making them robust (similarly to the proof of Lemma 3.6). Define new circuits $\varphi'_{\mathbf{i}}$ over new "encoding" variables W_i in a natural way as follows. Let $e : \{0,1\}^{\mathsf{n}_1} \to \{0,1\}^{c\mathsf{n}_1}$ be an error-correcting code, defined below. For every $i = 1, \ldots, n^\alpha$, let W_i be a set of $c\mathsf{n}_1$ new Boolean variables, supposedly representing the encoding via e of V_i. Let

$$(1) \quad \forall \mathbf{i} = (i_1, i_2, i_3) \in [\mathsf{n}^\alpha]^3, \qquad \varphi'_{\mathbf{i}} \text{ be a circuit over variables } W_{i_1} \cup W_{i_2} \cup W_{i_3}$$

that accepts only those assignments that are correct encodings (via e) of assignments that would have made $\varphi_{\mathbf{i}}$ accept.

With foresight, we choose an encoding e that, in addition to having good rate and distance, also has the following "local-checkability" property. Given an assignment a for the original X variables, and an assignment b for the encoding variables, it is easy to verify (with two queries) that either b is close to $e(a)$ or it is far from any codeword. This property will allow us to test consistency of the assignment for the new variables with the assignment to the original X-variables (which is used in the proof of Lemma 6.7 to argue that $\mathcal{A}_\alpha$ is an assignment tester, rather than just a PCP verifier). Note that this property is needed only when encoding the rows of V that are composed of X-variables, whereas for the rows containing Y-variables a more standard error-correcting code would have sufficed (for simplicity, however, we use the same code for encoding all rows of V).

We obtain the aforementioned local-checkability property by choosing e to be the string-concatenation of two error-correcting codes e_1 and e_2, where $e_1 : \{0,1\}^{\mathsf{n}_1} \to \{0,1\}^{\frac{c}{2}\mathsf{n}_1}$ is an error-correcting-code as defined in Lemma 2.1, and $e_2 : \{0,1\}^{\mathsf{n}_1} \to \{0,1\}^{\frac{c}{2}\mathsf{n}_1}$ is the trivial "repetition" encoding; i.e., it outputs $c/2$ repetitions of each bit in the input. (The choice of $c/2$ as the number of repetitions, though convenient, is somewhat arbitrary; even a single repetition would have worked, though with worse parameter.) The distance of e is at least the distance of e_1, i.e., at least n_1. By Lemma 2.1, the size of $\varphi'_{\mathbf{i}}$ is linear in that of $\varphi_{\mathbf{i}}$; i.e., it is $O(\mathsf{n}_1)$.

Let W be an $n^\alpha \times c\mathsf{n}_1$ matrix of variables whose rows are W_i. The first $\frac{c}{2}\mathsf{n}_1$ columns (the "left" half) of W correspond to encoding according to e_1, and the last $\frac{c}{2}\mathsf{n}_1$ columns (the "right" half) of W correspond to $\frac{c}{2}$ copies of V (recall that V is the matrix that contains the X-variables in its first rows). The following proposition

uses only the properties of the left-side code.

PROPOSITION 6.3. *Let $b : W \to \{0,1\}$, and define $a_b : X \to \{0,1\}$ to be the maximum-likelihood decoding*[9] *of b. If a_b doesn't satisfy φ, then there exists some $\mathbf{i}$ such that $\varphi'_{\mathbf{i}}$ is $\frac{1}{6c}$-far from being satisfied by b.*

Proof. If a_b doesn't satisfy φ, then there is some $\varphi_{\mathbf{i}}$, $\mathbf{i} = (i_1, i_2, i_3)$, that falsifies a_b. This means that at least one of $W_{i_1}, W_{i_2}, W_{i_3}$ needs to be changed *in more than half of the code's distance,* n_1 number of bits, so that together they encode a value that satisfies $\varphi'_{\mathbf{i}}$. The number of bits that need to be changed is at least $\mathsf{n}_1/2$, which is at least a $\frac{1}{6c}$ fraction of the $3 \cdot c\mathsf{n}_1$ input bits of $\varphi'_{\mathbf{i}}$. □

Applying $\mathcal{A}$. As discussed in the overview, before applying $\mathcal{A}$ on the $\varphi'_{\mathbf{i}}$ circuits we need to turn $\mathcal{A}$ into an oblivious tester. Fortunately this is can be done by the following lemma, whose proof we defer to section 6.4.

LEMMA 6.4. *There exists some constant $c_1 > 0$ such that any assignment tester $\mathcal{A}$ can be made into an oblivious assignment tester $\mathcal{A}'$, and the parameters of $\mathcal{A}'$ are equal to $\mathcal{A}$'s parameters computed on input size $\mathsf{n}' = \mathsf{n} \cdot (\log \mathsf{n})^{c_1}$. Furthermore, on an n-bit input, $\mathcal{A}'$ needs to invoke $\mathcal{A}$ only on one n'-bit input.*

Let $\mathcal{A}'$ be the oblivious version of $\mathcal{A}$ guaranteed by Lemma 6.4. Next, apply $\mathcal{A}'$ on each $\varphi'_{\mathbf{i}}$ obtained in (1), denoting the output by $\psi_{\mathbf{i},1}, \ldots, \psi_{\mathbf{i},\mathsf{R}_1}$. Note that this application of $\mathcal{A}'$ requires applying $\mathcal{A}$ only on inputs of size $\mathsf{n}_1 \mathrm{polylog} \mathsf{n}_1 < N$ (by Lemma 6.4 and the definition of N_α and n_1), so $\mathsf{R}_1 = \mathsf{R}(|\varphi'_{\mathbf{i}}| \,\mathrm{polylog}\, |\varphi'_{\mathbf{i}}|) = \mathsf{R}(\mathsf{n}_1 \mathrm{polylog} \mathsf{n}_1)$. The circuits $\psi_{\mathbf{i},1}, \ldots, \psi_{\mathbf{i},\mathsf{R}_1}$ are over variables from at most three rows in W and over new variables $Z_{\mathbf{i}}$. Each circuit has size $O(1)$.

Circuit and variable aggregation. The next step is to think of $\psi_{\mathbf{i},1}, \ldots, \psi_{\mathbf{i},\mathsf{R}_1}$ as a row in a matrix, and to define a new test (i.e., circuit) for each column:

$$\forall k = 1, \ldots, \mathsf{R}_1, \qquad \psi^k = \bigwedge_{\mathbf{i}} \psi_{\mathbf{i},k}.$$

There are $m = \mathsf{n}^{3\alpha}$ rows, and since the size of each $\psi_{\mathbf{i},k}$ is $O(1)$, these new circuits have size $m \cdot O(1) = O(\mathsf{n}^{3\alpha})$. Moreover, as long as the distance parameter of $\mathcal{A}$ is $\delta \le \frac{1}{6c}$, the next result follows immediately.

PROPOSITION 6.5. *Let b, a_b be as in Proposition 6.3 above. Let b_1 be any assignment for $\bigcup_{\mathbf{i}} Z_{\mathbf{i}}$. If a_b doesn't satisfy φ, then at most ε of $\psi^1, \ldots, \psi^{\mathsf{R}}$ are satisfied by $b \cup b_1$.*

The circuits $\psi^1, \ldots, \psi^{\mathsf{R}}$ are therefore close to being the desired output of $\mathcal{A}_\alpha$. They do, however, each depend on many (i.e., $\Theta(m)$) Boolean variables, rather than on a constant number of possibly larger-range variables. This is where the obliviousness of $\mathcal{A}'$ comes into play. We can simply cluster the variables into "columns" so that each ψ^k will depend on a constant number of clusters (and the clusters are not too large).

For any $\mathbf{i} = (i_1, i_2, i_3)$, the circuits $\psi_{\mathbf{i},1}, \ldots, \psi_{\mathbf{i},\mathsf{R}_1}$ are defined over variables in three rows of the matrix W (having n^α rows), namely $W_{i_1} \cup W_{i_2} \cup W_{i_3}$, and also over $Z_{\mathbf{i}}$, the auxiliary variables generated by running $\mathcal{A}'$ on $\varphi'_{\mathbf{i}}$. Let Z be the matrix whose rows are $Z_{\mathbf{i}}$ (i.e., Z has $\mathsf{n}^{3\alpha}$ rows). Now, for each k, the circuit $\psi'_{\mathbf{i},k}$ depends on some constant q locations in the string $W_{i_1} \cup W_{i_2} \cup W_{i_3}$, and $Z_{\mathbf{i}}$. Since $\mathcal{A}'$ is oblivious, the *positions queried in these four strings are a function only of the index k* (and are independent of $\mathbf{i}$). The punchline is therefore the following.

FACT 6.6. *For each circuit ψ^k, all of the variables read by ψ^k are contained in q columns of W and Z, where $\mathsf{q} = O(1)$ is the query complexity of $\mathcal{A}$.*

[9] As in Notation 2.2, the decoding of a word σ is $e^{-1}(\hat{\sigma})$, where $\hat{\sigma}$ is the codeword with minimal Hamming distance to σ (where ties can be broken arbitrarily).

The next step is now clear. We replace each column of variables (in both matrices W and Z) by a new (non-Boolean) variable that has width n^α or $\mathsf{n}^{3\alpha}$, respectively. Let $w^1, \ldots, w^{m_1}$ be new variables representing the columns of W, $m_1 = c\mathsf{n}_1$. Similarly let $z^1, \ldots, z^{m_2}$ be new variables representing the columns of Z, $m_2 \leq \mathsf{R}_1$. The tests simulate the previous ones as follows. For every $k \in [\mathsf{R}_1]$, recall that ψ^k was the conjunction of $\psi_{\mathbf{i},k}$ for all $\mathbf{i}$. We define $\bar{\psi}^k$ to be the test that verifies the predicate ψ^k by reading the appropriate $\leq \mathsf{q}$ column variables that, by Fact 6.6, contain all of the variables read by ψ^k.

The main point is that every variable in $W \cup Z$ appears in *exactly one* new column variable. Therefore, consistency is not an issue, as there is a one-to-one correspondence between assignments to the old variables and assignments to the new ones.

Consistency with the assignment to X. Finally, we must add tests that compare the values of $\{w^i\}_i \cup \{z^i\}_i$ to the assignment for the original X-variables. Recall that each variable x belongs to some row V_i, and that $\frac{c}{2}$ entries in the right half of W_i are supposed copies of x. For every $x \in X$ and every one of the $\frac{c}{2}$ columns in W that are supposed to contain a copy of x, we introduce a basic $\mathsf{compare}$ test that reads x and the column variable, and compare the value of x against its supposed copy.

We now amplify the detection probability of the compare tests. Let $d = O(\frac{1}{\delta}\log\frac{1}{\varepsilon})$; we then define a set of tests by applying Corollary 2.5 on the sequence of the above compare tests, with parameters $\mu = 6\delta$ and $\beta = \varepsilon$. The number of compare tests is $\frac{c}{2}|X| = O(\mathsf{n})$, and the size of each one is $O(\mathsf{n}^\alpha)$ (because it reads one bit of X, one variable w^i of width n^α, and later only compares two of its input bits). Denote the new tests by $\{\mathsf{compare}'_1, \ldots, \mathsf{compare}'_{M_1}\}$. By Corollary 2.5, $M_1 = poly(1/\varepsilon)\mathsf{n}$, and each $\mathsf{compare}'_i$ is the AND of d $\mathsf{compare}$ tests. Therefore it accesses $O(d)$ variables, and its size is $O(\mathsf{n}^{3\alpha})$.

Final circuits and their soundness. By appropriate replication, we can assume that the number (M_1) of tests in $\{\mathsf{compare}'_i\}$ is equal to the number of tests in $\{\bar{\psi}^i\}$, and both are equal to $poly(1/\varepsilon)\mathsf{nR}_1$. Let the ith final output circuit of $\mathcal{A}_\alpha$ be the AND of $\bar{\psi}^i$ and $\mathsf{compare}'_i$. We next prove the soundness condition of $\mathcal{A}_\alpha$ (completeness of $\mathcal{A}_\alpha$, its claimed parameters, and the input sizes for which it is defined are easy to verify).

LEMMA 6.7 (soundness). *Let $\delta_1 = 24\delta$, and let $a : X \to \{0,1\}$. If a is δ_1-far from satisfying φ, then for every assignment b for $\{w^1, \ldots, w^{m_1}\} \cup \{z^1, \ldots, z^{m_2}\}$, at most ε of the final circuits can be satisfied.*

Proof. The assignment b can be read as an assignment for $W \cup Z$, since w^i and z^i are variables that represent columns of W and Z, respectively. Define $a_b : X \to \{0,1\}$ to be the maximum-likelihood decoding according to e of $b|_W$:

$$\forall V_i \subset X, \qquad a_b(V_i) = e^{-1}(b(W_i)).$$

- If a_b does not satisfy φ, then we proceed just like in Proposition 6.5: There must be some $\varphi_{\mathbf{i}}$ falsified by a_b, so let us consider $\varphi'_{\mathbf{i}}$. The restriction of b to the input of $\varphi'_{\mathbf{i}}$ needs to be changed in at least $\mathsf{n}_1/2$ number of bits, in order to satisfy $\varphi'_{\mathbf{i}}$. By the definition of $\mathcal{A}$ this implies that regardless of what b assigns to the $\mathbf{i}$th row in Z, at most ε of the circuits $\{\psi_{\mathbf{i},k}\}_k$ will accept. Therefore, at most ε of the circuits in $\{\bar{\psi}^i\}$ will accept.
- Otherwise, a_b is δ_1-far from a. We must consider also the following "majority" assignment, which can be defined according to b. Recall that the right half of W is supposedly the repetition encoding of V. Let $a_{b,maj} : X \to \{0,1\}$ be the majority decoding (understood to be defined on the appropriate coordinates)

of b restricted to the right half of W (denoted W^r with rows W_i^r):

$$\forall V_i \subset X, \qquad a_{b,maj}(V_i) = majority(b(W_i^r)).$$

We have three assignments for X: $a, a_{b,maj}$, and a_b. If the first two are $\delta_1/2$-far from each other, then by definition an ε-fraction of the compare tests will reject, and we are done. We know that a and a_b are δ_1-far from each other, so the only other possibility is that the last two are $\delta_1/2$-far from each other. In this case, there must be some V_i for which $a_b(V_i)$ disagrees with $a_{b,maj}(V_i)$ on at least $\frac{\delta_1}{2} \cdot \mathsf{n}_1$ entries. This, by definition of $a_b, a_{b,maj}$, and e, implies that W_i is at least $\frac{\delta_1}{8}$-far from a legal codeword. Thus every circuit that reads W_i (say, for example, $\varphi'_{\mathbf{i}}$ with $\mathbf{i} = (i, i_2, i_3)$) sees an input that is at least $\frac{\delta_1}{24}$-far from a satisfying input. Since $\delta = \frac{\delta_1}{24}$ is the distance parameter of $\mathcal{A}$, this means that at most ε of $\psi_{\mathbf{i},1}, \ldots, \psi_{\mathbf{i},\mathsf{R}_1}$ are satisfied by b, which implies the same for $\psi^1, \ldots, \psi^{\mathsf{R}_1}$. □

This completes the proof of Lemma 6.2. □

6.3. Proof of Theorem 1.2. As discussed above, given the construction of $\mathcal{A}_\alpha$, completing the proof of Theorem 1.2 goes along the same lines as the construction of section 5. In the following, we formalize this idea.

THEOREM 1.2 (formal statement). *There exist constants* $\mathsf{s}_0, \mathsf{q}_0 > 0$ *and* $\varepsilon, \delta < 1$ *and an explicit combinatorial construction of an assignment tester* $\mathcal{A}$ *with parameters* $\mathsf{R}(\mathsf{n}) = n^{\mathrm{polylog}n}$, $\mathsf{s}(\mathsf{n}) = \mathsf{s}_0$, $\mathsf{q}(\mathsf{n}) = \mathsf{q}_0$, $\delta(\mathsf{n}) = \delta$, $\varepsilon(\mathsf{n}) = \varepsilon$.

Proof. The proof follows by induction on the input size. As discussed above, for the base of the induction, we rely on a given constant-size tester, say the one in Proposition 5.4 of section 5. Denote this tester by $\mathcal{A}_0$, and let its parameters for input circuits of size at most n_0 be $(\mathsf{R}_0, \mathsf{s}_0, \mathsf{q}_0, \delta_0, \varepsilon_0)$.[10]

Now we assume by induction that $\mathcal{A}$ has already been constructed for inputs of length smaller than N. Its parameters are $(\mathsf{R}(\mathsf{n}), \mathsf{s}(\mathsf{n}) = \mathsf{s}_0, \mathsf{q}(\mathsf{n}) = \mathsf{q}_0, \delta(\mathsf{n}) = \delta, \varepsilon(\mathsf{n}) = \varepsilon)$, with $\mathsf{s}_0, \mathsf{q}_0, \varepsilon$, and δ being fixed constants (δ will be set by the proof to a small enough constant). Applying Lemma 6.2 for some fixed $\alpha < \frac{1}{3}$, we get an assignment tester $\mathcal{A}_\alpha$ that is defined for inputs of size up to N_α (for N_α such that $(N_\alpha)^{1-\alpha}\mathrm{polylog}N_\alpha \leq \mathsf{n}$), and such that $\mathcal{A}_\alpha$'s output circuit size is $\mathsf{s}(\mathsf{n}) = O(\mathsf{n}^{3\alpha})$. The rest of $\mathcal{A}_\alpha$'s parameters are $\mathsf{R}_\alpha(\mathsf{n}) = \mathsf{R}(\mathsf{n}^{1-\alpha}\mathrm{polylog}\mathsf{n}) \cdot \mathsf{n} \cdot \mathrm{poly}(1/\varepsilon) = O(\mathsf{R}(\mathsf{n}^{1-\alpha}\mathrm{polylog}\mathsf{n}) \cdot \mathsf{n})$; $\mathsf{q}_\alpha(\mathsf{n}) = O(\frac{1}{\delta}\log\frac{1}{\varepsilon})$, $\varepsilon_\alpha(\mathsf{n}) = \varepsilon$; $\delta_\alpha(\mathsf{n}) = 24\delta$.

For simplicity, let us assume that the circuits output by $\mathcal{A}_\alpha$ each make three queries. This can be guaranteed[11] as described in the proof of Theorem 4.8; see, in particular, Proposition 4.11.

We can now use $\mathcal{A}_\alpha$ to extend $\mathcal{A}$ to n-bit inputs in exactly the same way followed in section 5. That is, we apply steps I, II(a), II(b), II(c) described in Assignment Tester $\mathcal{A}$. The analysis is as in the proof of Lemma 5.5, but a bit less delicate since we are already allowing ourselves quasi-polynomial R (and thus are less constrained in setting the relations between the various parameters). We therefore quickly go through the details.

I. $\mathcal{A}_I$ is the composition of $\mathcal{A}_\alpha$ with the recursive invocation of $\mathcal{A}$, using Theorem 3.7. This is well defined as long as we set δ (the distance parameter of

[10] Such an $\mathcal{A}_0$ can be derived from an even less efficient tester $\mathcal{A}'_0$, which produces circuits of size smaller than $(n_0)^{1-\beta_0}$ for some constant β_0 rather than circuits that are smaller than $(n_0)^{\alpha_0}$ for arbitrarily small α_0. The desired $\mathcal{A}_0$ can now be obtained as in the proof of Corollary 5.3.

[11] This replacement causes the error probability to go up, but this will be compensated for in the next steps.

$\mathcal{A}$) to be a small enough constant. $\mathcal{A}_I$ is defined for inputs of length $O(N)$. The number of circuits produced by $\mathcal{A}_I$ is $\mathsf{R}_I(\mathsf{n}) = O(\mathsf{R}(\mathsf{n}^{1-\alpha}\mathrm{polylog}\mathsf{n}) \cdot \mathsf{n} \cdot \mathsf{R}(O(\mathsf{n}^{3\alpha})))$. $\mathcal{A}_I$ has a (small) constant detection probability γ_I, circuit size s_0, number of queries q_0, and distance parameter $\delta_\alpha = 24\delta$.

II(a). We now apply error-reduction to $\mathcal{A}_I$ to increase the detection probability from γ_I to $\sqrt{\gamma}$, using Theorem 4.8, where $\gamma = 1 - \varepsilon$.
Since both γ_I and γ are constants we have that the resulting assignment tester $\mathcal{A}_{IIa}$ produces $\mathsf{R}_I(\mathsf{n})^{O(1)}$ circuits. Its circuit size is $O(\mathsf{s}_0)$, the number of queries is $O(1)$, the distance parameter has doubled to $2\delta_\alpha$, and the detection probability is $\sqrt{\gamma}$, as desired.

II(b). Next, we get $\mathcal{A}_{IIb}$ by reducing the distance parameter of $\mathcal{A}_{IIa}$ from $2\delta_\alpha$ to δ according to Lemma 4.1. As discussed in Remark 4.4, this will require applying $\mathcal{A}_{IIa}$ on circuits of size $M = O(\frac{2\delta_\alpha}{\delta}\mathsf{n}) = O(\mathsf{n})$. This is fine, as $\mathcal{A}_I$ is defined for such input lengths.
The number of circuits that $\mathcal{A}_{IIb}$ produces is $\mathsf{R}_I(M)^{O(1)} = \mathsf{R}_I(O(\mathsf{n}))^{O(1)}$. The number of queries is $O(1)$, the circuit size is $O(\mathsf{s}_0)$, the detection probability remains $\sqrt{\gamma}$, and the distance parameter is δ, as desired.

II(c). Finally, we define $\mathcal{A}_{IIc}$ as the composition $\mathcal{A}_{IIc} = \mathcal{A}_{IIb} \circ \mathcal{A}_0$. This is well defined as long as n_0 is a large enough constant with respect to s_0 (a more delicate analysis shows that here too it is enough to have $\mathsf{n}_0 = \mathrm{poly}(\mathsf{s}_0)$) and δ_0 is a small enough constant that satisfies the condition of Theorem 3.7. The parameters obtained by $\mathcal{A}_{IIc}$ are number of circuits $\mathsf{R}_0 \cdot \mathsf{R}_I(O(\mathsf{n}))^{O(1)} = \mathsf{R}_I(O(\mathsf{n}))^{O(1)}$; circuit size s_0; number of queries q_0; detection probability $\sqrt{\gamma} \cdot (1-\varepsilon_0)$, which is larger than γ in case ε_0 is small enough; distance parameter δ.

Finally, $\mathcal{A}$ is simply defined to run $\mathcal{A}_{IIc}$ on circuits of length $\mathsf{n} \leq N$. It remains to verify that the parameters obtained by $\mathcal{A}_{IIc}$ for such circuits are as good as those required of $\mathcal{A}$. Note that this holds trivially for all parameters apart from the number of circuits. The number of circuits are $\mathsf{R}_I(O(\mathsf{n}))^{O(1)}$, which opens to the following recursion formula: $\mathsf{R}(N) = (O(\mathsf{R}(\mathsf{n}^{1-\alpha}\mathrm{polylog}\mathsf{n}) \cdot \mathsf{n} \cdot \mathsf{R}(O(\mathsf{n}^{3\alpha}))))^{O(1)}$. Letting $a \geq 1$ be such that $(1-\alpha)^{a+1} + (3\alpha)^{a+1} < 1$, $\mathsf{R}(\mathsf{n}) = O(\mathsf{n}^{(\log \mathsf{n})^a})$ solves this recursive formula. □

6.4. Making testers oblivious. We now return to proving Lemma 6.4, which shows how every tester can be turned oblivious.

Proof of Lemma 6.4. We start by showing the existence of the following "universal circuit": C_U takes as input some encoding of a Boolean circuit φ, an assignment a to the input variables X of φ, and an assignment b to some additional "help variables." The following hold: (a) The encoding of φ can be computed in polynomial time; (b) if a satisfies φ, then there exists an assignment b (computed from φ and a in polynomial time) such that $C_U(\varphi, a, b) = 1$; (c) if a does not satisfy φ, then for any b, $C_U(\varphi, a, b) = 0$; (d) for input circuits φ of size n, the size of C_U is bounded by $\mathsf{n} \cdot \mathrm{polylog}\mathsf{n}$ (and this, in particular, bounds the size of the encoding of φ).

Let us first see how such a C_U can produce an oblivious tester $\mathcal{A}'$. We will then describe how to construct C_U using standard ideas.

Instead of applying $\mathcal{A}$ to φ we would like to apply it to the universal circuit C_U of the appropriate length. This will ensure obliviousness since, as $\mathcal{A}$ does not even "know" φ, the variables read by the circuits it produces cannot depend on φ. As a sanity check, we note that the input to C_U includes the description of φ, and therefore the circuits produced by $\mathcal{A}$ will read part of this description and will evaluate a predicate that does depend on φ. One (rather minor) difficulty in this approach is

that the tester $\mathcal{A}$ guarantees only that the assignment to the input of C_U is close to satisfying C_U. Unfortunately, this does not translate to saying that the assignment to the X-variables is close to satisfying φ. Instead, it is quite possible that the assignment to the X-variables will pass the test since it satisfies a circuit whose description is close to that of φ. To make sure that φ cannot be altered in our analysis, we revise C_U a bit as follows.

As a first step, $\mathcal{A}$ is applied to C'_U that takes the following inputs: an encoding of φ with the error-correcting code of Lemma 2.1, and a repetition encoding of the assignment to the X-variables, and a repetition encoding of the assignment to the Y-variables. In both cases, the number of repetitions is set so that the encoding of φ and the two assignments are of equal length. C'_U verifies that all of the encodings are legal. It then decodes the values it got and applies C_U. Now $\mathcal{A}'$ will run as follows. $\mathcal{A}'$ first applies $\mathcal{A}$ on C'_U. Then $\mathcal{A}'$ evaluates the description φ that C_U expects, and hardwires the encoding of this description to the circuits produced by $\mathcal{A}$ (i.e., whenever these circuits query a value in the encoded description of φ, this value is assigned by $\mathcal{A}'$ and the circuit is possibly simplified). Now the circuits that are obtained are just over the assignment to the X- and Y-variables. More accurately, the circuits are over the repetition encoding of these variables. However, whenever an assignment to a copy of one of the variables is required, we use the original assignment. (The purpose of the repetition encoding—here and elsewhere—is to balance the three kinds of variables on which C'_U was defined, so that $\mathcal{A}$ "notices" distance in each of the three types of input. Therefore in the output circuits we can safely replace all of the repeated copies with the original variable.)

The correctness of $\mathcal{A}'$ follows from the correctness of $\mathcal{A}$ and the properties of C_U. It essentially has the parameters of $\mathcal{A}$ (when applied to inputs of quasi-linear size). Finally, it is oblivious, as the locations accessed by the circuits produced do not depend on φ at all (as these circuits were first produced by applying $\mathcal{A}$ to the universal circuit).

We now describe how to construct the universal circuit C_U. The only subtlety comes from the requirement that $|C_U|$ be quasi-linear in its input size. This can be achieved relying on well-known ideas. Since we have not been able to find a proper reference, we include a sketch of the construction.

The circuit C_U will operate over a preprocessed φ, transformed into a $3SAT$ formula $C_1 \wedge \cdots \wedge C_m$, over variables X and Y (with $|X \cup Y| \leq \mathsf{n}$), where each variable appears only a constant number of times. This is achieved as in the proof of Lemma 6.2.

There is a very efficient (essentially linear) universal machine for such a $3SAT$ formula, if the machine is allowed *random access* to its input. Simply, read the description of the clauses C_i one-by-one and then read the three variables contained in the clause to evaluate it. When translating this universal machine naively to the setting of circuits, the universal circuit obtained is of size $O(\mathsf{n}^2)$ (as "fetching" each variable costs $O(\mathsf{n})$). To reduce the size to be quasi-linear, we use the following standard trick.

The evaluation of the formula by the smaller universal circuit will be done in three phases. First we order the m clauses by the order of their variable of smallest index (i.e., first the clauses that contain x_1 (or its negation), then those that contain x_2 and do not contain x_1, and so on). Together with the clauses we order the variable names and their assignments. Now that both are jointly ordered, we can assign values to one variable from each clause. At this point, the location of each clause in the joint

order is a constant distance away from the location of an assignment to one of its variables. Therefore, evaluating a variable from each clause can be done by a circuit of polylogarithmic size (as the input for the circuit has a logarithmic number of bits—the descriptions of a constant number of clauses and variables). Continue in the same manner to assign values to the two additional variables in each clause. Once all variable values are assigned, the value of each clause is determined by a constant size circuit which easily implies the value of the entire formula (by evaluating the AND of all clause values). It remains to argue that there are circuits of quasi-linear size for sorting. This, however, follows immediately from the existence of quasi-linear sorting networks (where any one of a number of simple sorting networks will do; see, e.g., [10, section 55]). □

7. Discussion and further research. We have introduced the notion of assignment testers and provided a simple and truly modular composition theorem for testers. We feel that it is beneficial to state even the original proof of the PCP theorem via assignment testers, as their composition seems to us simpler and more natural. In addition, we provided various generic transformations for assignment testers: (a) making assignment testers "robust" (section 3.4), (b) reducing the distance parameter (section 4.1), (c) error reduction via a new method of combinatorial aggregation (section 4.3).

Our first construction of assignment testers (given in section 5) provides a new proof of the PCP theorem. It relies on the existence of a relatively weak assignment tester, which is provided as a black box. One advantage of this construction over the original proof of the PCP theorem is that the algebraic building block requires only the algebraic techniques that are already present in [4, 13] (in particular, it only needs a weak form of the low-degree test, and it does not use aggregation via low-degree curves). More importantly, *the algebraic techniques are confined to the construction of the black box*. The way the black box was constructed can then be forgotten, and we care only about its parameters. Finally, we use a building-block PCP of only one particular kind (as opposed to the original proof, which also used Hadamard-based PCP). This is made possible through using a nonconstant number of composition steps (to which our composition theorem readily yields itself). On the other hand, one may also consider the superconstant number of recursive steps to be a disadvantage. It is indeed harder to know "what's going on" by such a construction and particularly to track how the final variables relate to the original ones.

Our second construction is fully combinatorial, and it relies only on standard objects such as error-correcting codes and hitting sets (which both follow from expander graphs). It also relies on a constant-size tester, which is easy to construct due to its allowed inefficiency. In this respect the construction is even simpler (even though the combinatorial part of the construction is somewhat more complicated). The major disadvantage is of course the quasi-polynomial size. We note, however, that such a result has applications similar to those of the PCP theorem (basing hardness of approximation on the still highly conservative assumption that NP is not contained in quasi-polynomial time).

On the robustness property. We mentioned above that a notion very related to assignment testers was independently introduced by Ben-Sasson et al. [8], where it was named "robust PCPs of proximity." Interestingly, their motivation was completely different, namely constructing length-efficient PCPs and locally testable codes, yet they came up with essentially the same object.

Just as in [8], robustness is an essential part of our composition. However, our

composition theorem (Theorem 3.7) applies directly to assignment testers that are not necessarily robust. For that we use in the proof of the composition theorem a generic transformation that turns any assignment tester into a robust one, with the robustness inversely related to the query complexity. Most of our work can therefore ignore this additional parameter of robustness. Emphasizing assignment testers rather than robust assignment testers has the advantage of staying closer to the original definition of a PCP verifier. In addition, the importance of the query complexity as the effective measure of robustness is emphasized. In particular, it seems more natural to state and prove the aggregation theorem, a central ingredient of our work, in terms of query complexity.

We note that the cost of a generic robustization transformation is too high in the context of [8]; hence they work only with assignment testers that are, by definition, robust (indeed they call these objects robust PCPs of proximity). Since [8] demonstrates the usefulness of having the robustness property as an explicit parameter, it could be interesting to study how our transformations on assignment testers behave in terms of this parameter.

On PCP testers and property testing. The notion of assignment testers is very related to the area of property testing [26, 18], and we were most likely inspired by property testing in coming up with this notion. An assignment tester can easily be converted into a test that checks whether an assignment is close to being a satisfying assignment of a circuit φ. The object being tested is the assignment (hence the name "assignment tester"), and the circuit φ is a description of the tested property. (Usually in property testing this is a fixed property such as graph connectivity.) Of course, the main difference between this and standard property testing is that assignment testers also rely on a proof (the assignment of the new variables) in order to perform the testing. This is a special case of the notion of proof-assisted testing of Ergün, Kumar, and Rubinfeld [12].

Adding proofs to property testing allows extremely efficient testing of any property computable in polynomial time. (As observed in [8], this easily extends to properties in NP.) Every such property can be tested by reading only $O(1)$ bits of an object X and a proof Y. This can loosely be interpreted as saying that every property has a highly testable representation. While being a very powerful statement, it is also a flat one, as it does not distinguish between different properties. In some sense, it reemphasizes that the richness of property testing is as a study of *specific representations*. (This was already well understood, as some properties behave very differently with respect to different representations.)

Locally testable codes. An interesting aspect of our construction, especially of the fully combinatorial one, is that we do not make any real use of locally testable codes (apart from the constant-size tester). Nevertheless, as was shown in [8], assignment testers easily imply locally testable codes (and also a relaxed form of locally decodable codes). The focus of [8] was to get *short* locally testable codes. Our Theorem 1.2 implies the first combinatorial construction of locally testable codes with subexponential (in fact, even quasi-polynomial) rate.

Further research. The most obvious problem that was left open by this work is coming up with a combinatorial proof of the PCP theorem. This has been recently obtained by Dinur [11]. Still, one may also hope to give an elementary construction for the constant-size testers used by the combinatorial constructions (constant-sized testers are used also in [11]). Nevertheless, the known constructions based on

Hadamard or Long codes are already rather simple (especially compared to other parts of the proof of the PCP theorem).

A task of a different nature is related to Raz's parallel repetition theorem [24]. Recall that our aggregation method bypasses the complexity of this theorem by adding a few additional "consistency queries." Nevertheless, we use this method only to reduce errors up to some constant probability. It seems possible that this approach could be extended to a simple, combinatorial way of reducing errors in an exponential rate. Though this will not provide a full substitute for the parallel repetition theorem, we still find it an interesting line for further research.

Appendix A. Analysis of the consistency test. As discussed in section 4, the heart of our aggregation/parallelization theorem (Theorem 4.8) is a test T for the consistency of a table $F : \overline{X} \to \Sigma^\ell$ that is supposed to contain the values of some function $f : X \to \Sigma$ on all ℓ-tuples of inputs. In other words, $F(x_1, \ldots, x_\ell)$ is supposed to equal $(f(x_1), \ldots, f(x_\ell))$, for some underlying function f. Such a "combinatorial" consistency test was given by Goldreich and Safra [19]. We described our test in Figure 3 and stated its properties in Theorem 4.10. This test is in fact a (somewhat stronger) version of Lemma 1.1 from [19], which was derived as a simple special case of a more elaborate test which is their main objective. The more sophisticated test is for tables containing a small "derandomized" subset of ℓ-tuples. Interestingly, in the context of our paper, the simple and "inefficient" version of the Goldreich–Safra test is sufficiently good. We now give a direct proof of Theorem 4.10. Our theorem proves that if the test accepts with high enough probability, then the table F is in fact consistent with the *plurality* function of F (as in Definition 4.9), which is the function f that maximizes individual agreements between F and f. This seems more natural than the two-stage plurality function obtained by the proof of [19]. We view the main contribution of this section to be the new and direct proof of the consistency test, which relies on a rather natural Markov-chain approach. Possibly, this proof can be generalized to work for a wider range of parameters (as mentioned in the open problems in section 7).

Proof of Theorem 4.10. Clearly, the completeness condition holds. We prove soundness (that is, we analyze the rejection probability when the table is sufficiently inconsistent with the plurality function).

We will concentrate on the test with respect to the uniform distribution on X (i.e., $\mathcal{D}$ in the statement of the theorem is uniform). The proof for general $\mathcal{D}$ is obtained by simple reduction to the uniform case, as follows.

PROPOSITION A.1. *Assuming a restricted version of Theorem* 4.10*, where $\mathcal{D}$ is the uniform distribution, the theorem holds for an arbitrary $\mathcal{D}$.*

Proof. We can translate any $\mathcal{D}$ over X into the uniform distribution over Z, where Z is obtained from X by duplicating elements in X according to their probability under $\mathcal{D}$. Elements that are not in support of $\mathcal{D}$ are simply dropped. It is clear that the projection of the uniform distribution on Z to the original set of variables X can be made arbitrarily close to $\mathcal{D}$ (and from now on we will simply assume that the two distributions are identical).

Let $\overline{Z} \equiv Z^\ell$. The duplication of variables naturally defines $\hat{F} : \overline{Z} \to \Sigma^\ell$, where for each $\bar{\mathbf{z}} \in \overline{Z}$ we let $\bar{\mathbf{x}}$ be the corresponding ℓ-tuple in $\overline{X}$ and define $\hat{F}(\bar{\mathbf{z}}) = F(\bar{\mathbf{x}})$. The soundness of the test of Figure 3 when applied to F with respect to $\mathcal{D}$ immediately reduces to the soundness of the same test applied to $\hat{F}$ *with respect to the uniform distribution*. (Note that, by definition, the plurality function of $\hat{F}$ agrees on any two

copies $\bar{\mathbf{z}}$ and $\bar{\mathbf{z}}'$ in $\overline{Z}$ that correspond to the same $\bar{\mathbf{x}} \in \overline{X}$.) ∎

Another simplifying convention is our assumption that F is rotation consistent. Recall that F is rotation consistent if, whenever $\bar{\mathbf{x}}'$ is obtained from $\bar{\mathbf{x}}$ using some cyclic shift of its ℓ components, F is consistent on these two entries (i.e., $F(\bar{\mathbf{x}}')$ can be obtained from $F(\bar{\mathbf{x}})$ by a similar shift). As discussed in section 4, in our setting we can indeed assume that F is rotation consistent. Moreover, it is easy to see that the general case can be reduced to the rotation consistent case.

PROPOSITION A.2. *Assuming that Theorem* 4.10 *holds for every F that is rotation consistent, the theorem holds for an arbitrary F, with the slightly revised test described in Figure* 3. ∎

Proof. The original test makes two queries into the table F for the values $F(\bar{\mathbf{x}})$ and $F(\bar{\mathbf{x}}')$. The revised test will select $\bar{\mathbf{x}}$ and $\bar{\mathbf{x}}'$ in the same manner, but instead of querying for $F(\bar{\mathbf{x}}')$, it will query for the value of F on a random cyclic rotation of $\bar{\mathbf{x}}'$. It is not hard to see that this is equivalent to performing the original test on a related *distribution* F' over rotation consistent tables. More specifically, for every set of ℓ entries $\bar{\mathbf{x}}^1, \ldots, \bar{\mathbf{x}}^\ell$ that are cyclic shifts of each other, the value of F' on *all these tuples* is defined as $F(\bar{\mathbf{x}}^i)$ for a random i. This implies that if F violates the soundness of the revised test, then at least one of those rotation consistent tables violates the soundness of the original test. ∎

Let us now recall the test T that we are analyzing and introduce some notations. Let $\bar{\mathbf{x}} = (x_1, \ldots, x_\ell)$ and suppose $F(\bar{\mathbf{x}}) = (a_1, \ldots, a_\ell)$; we write $\bar{\mathbf{x}}_i$ to mean x_i and $F(\bar{\mathbf{x}})_i$ to mean a_i.

1. Select $\bar{\mathbf{x}} \in \overline{X}$ uniformly at random.
2. Select a set of indices J_T such that each $j \in [\ell]$ is placed into J_T with probability $\alpha = 1/\sqrt[3]{\ell}$, independent of other choices.
3. Select $\bar{\mathbf{x}}' \in \overline{X}$ as follows: for each $j \in J_T$, we let $\bar{\mathbf{x}}'_j = \bar{\mathbf{x}}_j$; otherwise $\bar{\mathbf{x}}'_j$ is selected uniformly in X, independent of other choices.
4. Accept only if $F(\bar{\mathbf{x}})_j = F(\bar{\mathbf{x}}')_j$ for every $j \in J_T$; otherwise reject.

Let $f = f_F : X \to \Sigma$ be the plurality function of F. For every $\bar{\mathbf{x}} \in \overline{X}$ define the set of indices on which $F(\bar{\mathbf{x}})$ differs from f,

$$\mathsf{wrong}(\bar{\mathbf{x}}) \stackrel{def}{=} \{\, i \in [\ell] \mid F(\bar{\mathbf{x}})_i \neq f(\bar{\mathbf{x}}_i) \}\,.$$

Call a tuple $\bar{\mathbf{x}} \in \overline{X}$ bad if $|\mathsf{wrong}(\bar{\mathbf{x}})| > \frac{1}{\alpha} = \sqrt[3]{\ell}$. Let γ be the fraction of bad tuples. We will prove that the test T rejects with probability at least γ/c, where c is some absolute constant.

The general idea for our proof is the following: With probability γ a bad $\bar{\mathbf{x}}$ is selected by T. Fix some bad $\bar{\mathbf{x}}$; since $\mathsf{wrong}(\bar{\mathbf{x}}) > \frac{1}{\alpha}$ and each index is placed into J_T with probability α, we have that there exists some i in $\mathsf{wrong}(\bar{\mathbf{x}}) \cap J_T$ with constant probability. Recall that for every $i \in J_T$ the test T sets $\bar{\mathbf{x}}'_i = \bar{\mathbf{x}}_i$. Let $a_i = F(\bar{\mathbf{x}})_i$, by the definition of f, for a uniformly distributed $\bar{\mathbf{y}} \in \overline{X}$ such that $\bar{\mathbf{y}}_i = \bar{\mathbf{x}}_i$; we have that $\Pr[F(\bar{\mathbf{y}})_i = a_i \mid \bar{\mathbf{y}}_i = \bar{\mathbf{x}}_i] \leq 1/2$. Ideally, if $\bar{\mathbf{x}}'$ was distributed exactly like $\bar{\mathbf{y}}$, the test would reject with constant probability. Our argument is based on the fact that even though $\bar{\mathbf{x}}'$ has some dependency on $\bar{\mathbf{x}}$, it is still "sufficiently random," and therefore $F(\bar{\mathbf{x}}')_i \neq a_i$ (so the test rejects) with some constant probability. What do we mean by $\bar{\mathbf{x}}'$ being sufficiently random? Consider the stochastic process $G_{x,i}$ of selecting $\bar{\mathbf{x}}'$ given $\bar{\mathbf{x}}$ and conditioned on $i \in J_T$ and $\bar{\mathbf{x}}_i = x$ for some fixed i and x. We will show that this process has good expansion properties. We will later argue that these expansion properties are indeed sufficient to carry out the above intuition.

DEFINITION A.3. *For any $i \in [\ell]$ and $x \in X$ define the set $V_{x,i} \stackrel{def}{=} \{ \bar{\mathbf{x}} \in \overline{X} \mid \bar{\mathbf{x}}_i = x \}$. Note that $|V_{x,i}| = |X|^{\ell-1}$. Define $G_{x,i}$ to be the Markov process over $V_{x,i}$ where, starting at $\bar{\mathbf{x}} \in V_{x,i}$, we move to $\bar{\mathbf{x}}'$ selected as follows: $\bar{\mathbf{x}}'_i = x$, and for each $j \in [\ell] \setminus \{i\}$ with probability α we let $\bar{\mathbf{x}}'_j = \bar{\mathbf{x}}_j$; otherwise $\bar{\mathbf{x}}'_j$ is selected uniformly in X, independent of other choices. Let $A_{x,i}$ be the transition probability matrix corresponding to $G_{x,i}$.*

PROPOSITION A.4. *For every $i \in [\ell]$ and $x \in X$, the second eigenvalue of $A_{x,i}$ is $\lambda = \lambda(A_{x,i}) = \alpha$.*

Proof. Since in the space state of $G_{x,i}$ the value in the ith coordinate is fixed (to x), we can simply ignore this coordinate. On the other hand, $G_{x,i}$ acts on each one of the $\ell-1$ other coordinates independently (in an identical way). Consider the transition probability matrix A^0 corresponding to the action of $G_{x,i}$ on each one of the $\ell-1$ coordinates in $[\ell] \setminus \{i\}$. This matrix equals the convex sum $(1-\alpha)K_{|X|} + \alpha I_{|X|}$, where $K_{|X|}$ corresponds to moving to a uniformly distributed element (i.e., every entry in $K_{|X|}$ is $1/|X|$), and $I_{|X|}$ is the identity matrix. Consider any vector $P \in R^{|X|}$ which is perpendicular to the all one vector (i.e., the sum of the entries in P is zero); then $A^0 \cdot P = ((1-\alpha)K_{|X|} + \alpha I_{|X|})P = \alpha \cdot P$ (since $K_{|X|} \cdot P = \mathbf{0}$). This implies that the second eigenvalue of A^0 is α. To conclude, $G_{x,i}$ acts on $\ell-1$ coordinates independently according to a transition probability matrix A^0 that has second eigenvalue α. It is well known and not hard to show that in such a case the second eigenvalue of $A_{x,i}$ is also α.[12] □

Consider now the foregoing intuition. We know that a bad $\bar{\mathbf{x}}$ is selected by T with probability γ. Furthermore, we know that, with constant probability, some $i \in \mathsf{wrong}(\bar{\mathbf{x}})$ is selected into J_T, meaning that $\bar{\mathbf{x}}'_i = \bar{\mathbf{x}}_i$. We hope to argue that, with constant probability, this location i will reveal to T that the table F is inconsistent. Let $\bar{\mathbf{x}}_i = x$ and $a_i = F(\bar{\mathbf{x}})_i$; then we have that $\bar{\mathbf{x}} \in V_{x,i}$ (as in Definition A.3), and since $i \in \mathsf{wrong}(\bar{\mathbf{x}})$, for most $\bar{\mathbf{y}} \in V_{x,i}$ we have that $F(\bar{\mathbf{y}})_i \neq a_i$. Therefore, if $\bar{\mathbf{x}}'$ was a random element of $V_{x,i}$, the test T would reject with constant probability (conditioned on $\bar{\mathbf{x}}$ being selected and $i \in J_T$). However, the distribution of $\bar{\mathbf{x}}'$ is obtained by taking a random step from $\bar{\mathbf{x}}$ according to the process $G_{x,i}$ (again, conditioned on $\bar{\mathbf{x}}$ being selected and $i \in J_T$). With this choice of $\bar{\mathbf{x}}'$, we can no longer argue that, for any *fixed* choice of a bad $\bar{\mathbf{x}}$, with constant probability the test rejects. (It may very well be that for all the "neighboring" $\bar{\mathbf{x}}'$'s the value $F(\bar{\mathbf{x}}')$ is consistent with $F(\bar{\mathbf{x}})$.) We will therefore make an average argument that will exploit the expansion of $G_{x,i}$. We consider not one possible value of $\bar{\mathbf{x}}$ but rather a set $S_{x,i,a}$ that contains all the values of $\bar{\mathbf{x}}$ such that $\bar{\mathbf{x}}_i = x$ and $F(\bar{\mathbf{x}})_i = a$ (with $f(x) \neq a$). Conditioned on $\bar{\mathbf{x}} \in S_{x,i,a}$ and $i \in J_T$ we do have that with constant probability $\bar{\mathbf{x}}' \notin S_{x,i,a}$ (since $S_{x,i,a}$ contains at most half the tuples in $V_{x,i}$ and due to the expansion of $G_{x,i}$). Therefore, with this conditioning, the test will reject with constant probability (as with constant probability $F(\bar{\mathbf{x}}')_i \neq a$). Details follow.

Consider the process G that corresponds to T selecting $\bar{\mathbf{x}}'$ given $\bar{\mathbf{x}}$ (without further conditioning). The same transition $(\bar{\mathbf{x}}, \bar{\mathbf{x}}')$ is also an edge in various $G_{x,i}$. To complete the proof, we want to lower bound the weight of rejecting edges in each $G_{x,i}$ separately (as outlined above) and deduce a similar lower bound for G (which reflects the rejecting probability of T). However, this may not be sound, as the $G_{x,i}$'s

[12] Each eigenvector of $A_{x,i}$ corresponds to an $(\ell-1)$-tuple of eigenvectors of A^0; the corresponding eigenvalue of $A_{x,i}$ is the product of the $\ell-1$ corresponding eigenvalues of A^0. Therefore, the second eigenvalue of $A_{x,i}$ is α, and it is obtained as the product of $\ell-2$ times the eigenvalue one and the eigenvalue α once.

may not be a "uniform enough cover" of G. More specifically, consider a transition $(\bar{\mathbf{x}}, \bar{\mathbf{x}}')$, where $\bar{\mathbf{x}}$ is bad. For every $i \in \mathsf{wrong}(\bar{\mathbf{x}}) \cap J_T$, this corresponds to a transition in $G_{\bar{\mathbf{x}}_i,i}$ from some $S_{\bar{\mathbf{x}}_i,i,a}$ of density at most half in $V_{\bar{\mathbf{x}}_i,i}$. However, the cardinality of $\mathsf{wrong}(\bar{\mathbf{x}}) \cap J_T$ may vary quite a lot. Potentially, this could mean that by counting separately for each $G_{\bar{\mathbf{x}}_i,i}$, rejecting edges are counted many times, while accepting edges are counted only a few times. Indeed, if we were assured that the size of $\mathsf{wrong}(\bar{\mathbf{x}})$ would be either zero (for all of the good $\bar{\mathbf{x}}$'s) or some fixed value (for all of the bad $\bar{\mathbf{x}}$'s), then the cardinality of $\mathsf{wrong}(\bar{\mathbf{x}}) \cap J_T$ would not vary too much and the proof would become easier. Intuitively, the larger $|\mathsf{wrong}(\bar{\mathbf{x}})|$ is the better, since the test has "more opportunity" to detect an inconsistency and reject. However, the argument is much more subtle, due to the fact that we are not arguing for every value of $\bar{\mathbf{x}}$ separately but rather averaging over sets of values $S_{x,i,a}$.

To help us manipulate the conditional probabilities more elegantly, it is convenient to consider as a "mental experiment" the following revised test T':

1. Choose $\bar{\mathbf{x}} \in \overline{X}$ uniformly at random.
2. Set $k(\bar{\mathbf{x}}) = \max\{16/\alpha, |\mathsf{wrong}(\bar{\mathbf{x}})|\}$.[13] Select an index $i \in [\ell] \cup \{0\}$ such that each $i \in \mathsf{wrong}(\bar{\mathbf{x}})$ is selected with probability $1/k(\bar{\mathbf{x}})$ and with the remaining probability $i = 0$. If $i = 0$, then T' accepts and halts.
3. Let $x = \bar{\mathbf{x}}_i$. Take a random step from $\bar{\mathbf{x}}$ to $\bar{\mathbf{x}}'$ according to $G_{x,i}$.
4. If $F(\bar{\mathbf{x}}')_i \neq F(\bar{\mathbf{x}})_i$, reject; otherwise accept.

We note that T' is not efficiently implementable, and is not meant to be. T' is merely a tool of the analysis, used to bound the probability that T rejects. The main convenience of T' is that it concentrates on a single possible inconsistency between $F(\bar{\mathbf{x}})$ and $F(\bar{\mathbf{x}}')$, namely inconsistency on the ith coordinate. This way we rather naturally avoid overcounting the rejection probability associated with a particular choice of $\bar{\mathbf{x}}$ and $\bar{\mathbf{x}}'$. Such overcounting may arise by counting the same pair $(\bar{\mathbf{x}}, \bar{\mathbf{x}}')$ separately for every inconsistent coordinate. Note that if $\mathsf{wrong}(\bar{\mathbf{x}})$ is small, then $J_T \cap \mathsf{wrong}(\bar{\mathbf{x}})$ is likely to be empty, and thus T will accept. Therefore, if $\mathsf{wrong}(\bar{\mathbf{x}})$ is small, we let T' accept too with some probability (ignoring $F(\bar{\mathbf{x}}')$ altogether). We do that by allowing i to be set to zero with some probability that depends on the size of $\mathsf{wrong}(\bar{\mathbf{x}})$. We remark that if i is different than zero, it is uniformly distributed in $\mathsf{wrong}(\bar{\mathbf{x}})$.

To bound the rejection probability of T through the rejection probability of T' we would have liked to show that (a) $\Pr[T' \text{ rejects}] = O(\Pr[T \text{ rejects}])$, and (b) $\Pr[T' \text{ rejects}] = \Omega(\gamma)$. However, as we do not know how to argue that (a) holds, we first factor out a possible (but rare) bad event B, on which T' may reject with significantly higher probability than T. Taking B into account, we prove Lemmas A.8 and A.9, which are small variations on (a) and (b) above. These two lemmas immediately imply the soundness of T and therefore also Theorem 4.10.

For the definition of the bad event B, we will need a definition of a random variable $J_{T'}$ in analogy to the random variable J_T.

DEFINITION A.5. *Recall the definition of the index i selected by T'. We define the random variable $J_{T'}$ as follows: $J_{T'}$ is set to be empty if $i = 0$. Otherwise $J_{T'}$ is the set of indices $j \in [\ell]$ for which T' sets $\bar{\mathbf{x}}'_j = \bar{\mathbf{x}}_j$ (in particular $i \in J_{T'}$).*

We would like to compare the behavior of T and T', and it would be convenient to do so conditioned on the value of $\bar{\mathbf{x}}$ and on a particular value J of both J_T and $J_{T'}$. However, for sets J such that $J \cap \mathsf{wrong}(\bar{\mathbf{x}})$ is large, we have that the probability that $J_{T'} = J$ may be significantly larger than the probability that $J_T = J$. The reason is

[13] The purpose of the constant 16 is for the proof of Proposition A.11.

that as long as $i \in J \cap \mathsf{wrong}(\bar{\mathbf{x}})$, it is possible that $J_{T'} = J$. Therefore, the larger $|J \cap \mathsf{wrong}(\bar{\mathbf{x}})|$ is, the more ways there are to obtain $J_{T'} = J$. For this reason, we define the "bad event" B, where $|J_{T'} \cap \mathsf{wrong}(\bar{\mathbf{x}})|$ is too large. The exact threshold is meant to facilitate the proof of Lemma A.8 and is less important for now.

DEFINITION A.6. *Define the event B to be $|J_{T'} \cap \mathsf{wrong}(\bar{\mathbf{x}})| / k(\bar{\mathbf{x}}) > 9\alpha$.*

Recall that, by definition, $k(\bar{\mathbf{x}})$ is always positive, and hence B is well defined. We would now like to argue that B is indeed a rare event.

PROPOSITION A.7. *Conditioned on any particular value of $\bar{\mathbf{x}}$ and of i, the probability that B occurs is smaller than $2/9$.*

Proof. If $i = 0$, then $J_{T'}$ is empty, and the proposition follows trivially. Otherwise, each index in $\mathsf{wrong}(\bar{\mathbf{x}}) \setminus \{i\}$ is placed into $J_{T'}$ with probability α (and i is placed into $J_{T'}$ with probability one). Therefore, the expected size of $J_{T'} \cap \mathsf{wrong}(\bar{\mathbf{x}})$ is at most $1 + \alpha\,|\mathsf{wrong}(\bar{\mathbf{x}})| \leq 1 + \alpha \cdot k(\bar{\mathbf{x}}) < 2\alpha \cdot k(\bar{\mathbf{x}})$. (The last inequality follows from $k(\bar{\mathbf{x}}) \geq 16/\alpha > 1/\alpha$.) Now, by Markov's inequality, the event $B \equiv (|J_{T'} \cap \mathsf{wrong}(\bar{\mathbf{x}})| > 9\alpha \cdot k(\bar{\mathbf{x}}))$ has probability at most $2/9$. □

We can now relate T' to T.

LEMMA A.8. $\Pr[T' \text{ rejects} \wedge (\neg B)] = O(\Pr[T \text{ rejects}])$.

Proof. Both T and T' select $\bar{\mathbf{x}}$ uniformly at random. We prove the inequality separately for every value of $\bar{\mathbf{x}}$ (that is, conditioned on $\bar{\mathbf{x}}$ taking some arbitrary value). Therefore, fix the value of $\bar{\mathbf{x}}$ in an arbitrary way. We first argue that, for any value $J \subseteq [\ell]$,

$$\Pr[T' \text{ rejects} \mid J_{T'} = J] \leq \Pr[T \text{ rejects} \mid J_T = J]. \tag{2}$$

First note that the distribution of $\bar{\mathbf{x}}'$ is identical in both cases (i.e., $\bar{\mathbf{x}}'_j = \bar{\mathbf{x}}_j$ for $j \in J$, and $\bar{\mathbf{x}}'_j$ is uniform outside J). If $J \cap \mathsf{wrong}(\bar{\mathbf{x}}) = \phi$, then $\Pr[T' \text{ rejects} \mid J_{T'} = J] = 0$, and we are done. Otherwise, T will reject if for some $j \in J$ we have that $F(\bar{\mathbf{x}}')_j \neq F(\bar{\mathbf{x}})_j$, whereas T' will reject only if $F(\bar{\mathbf{x}}')_i \neq F(\bar{\mathbf{x}})_i$ (recall that in this case $i \neq 0$, as otherwise $J_{T'}$ is empty). This implies inequality (2).

We now want to show that $\Pr[(J_{T'} = J)] = O(\Pr[J_T = J])$. However, this may not be true in two cases, which fortunately enough we can ignore. The two cases to ignore are (a) $J \cap \mathsf{wrong}(\bar{\mathbf{x}}) = \phi$, in which case T' always accepts, and (b) $|J \cap \mathsf{wrong}(\bar{\mathbf{x}})| / k(\bar{\mathbf{x}}) > 9\alpha$, in which case the event B occurs. Let the collection of sets J satisfying either (a) or (b) be denoted $\mathcal{J}_{\mathsf{ignore}}$. For all other values of J, the event B does not occur and therefore,

$$\Pr[T' \text{ rejects} \wedge (\neg B)] = \sum_{J \notin \mathcal{J}_{\mathsf{ignore}}} \Pr[T' \text{ rejects} \mid J_{T'} = J] \cdot \Pr[J_{T'} = J].$$

In addition, for $J \notin \mathcal{J}_{\mathsf{ignore}}$ we have that

$$\begin{aligned}
\Pr[J_{T'} = J] &= \sum_{j \in J \cap \mathsf{wrong}(\bar{\mathbf{x}})} \Pr[i = j] \cdot \alpha^{|J|-1} \cdot (1-\alpha)^{\ell - |J|} \\
&= \sum_{j \in J \cap \mathsf{wrong}(\bar{\mathbf{x}})} (1/k(\bar{\mathbf{x}})) \cdot \alpha^{|J|-1} \cdot (1-\alpha)^{\ell - |J|} \\
&= (|J \cap \mathsf{wrong}(\bar{\mathbf{x}})| / k(\bar{\mathbf{x}})) \cdot \alpha^{|J|-1} \cdot (1-\alpha)^{\ell - |J|} \\
&\leq 9\alpha \cdot \alpha^{|J|-1} \cdot (1-\alpha)^{\ell - |J|} \\
&= 9 \cdot \Pr[J_T = J].
\end{aligned}$$

By inequality (2), we can now conclude that

$$\begin{aligned}\Pr[T' \text{ rejects} \wedge (\neg B)] &= \sum_{J \notin \mathcal{J}_{\mathsf{ignore}}} \Pr[T' \text{ rejects} \mid J_{T'} = J] \cdot \Pr[J_{T'} = J] \\ &\leq \sum_{J \notin \mathcal{J}_{\mathsf{ignore}}} \Pr[T \text{ rejects} \mid J_T = J] \cdot 9 \Pr[J_T = J] \\ &\leq 9 \Pr[T \text{ rejects}]. \qquad \square\end{aligned}$$

It remains to bound the probability that T' rejects (again, factoring out the bad event B), as follows.

LEMMA A.9. $\Pr[T' \text{ rejects} \wedge (\neg B)] = \Omega(\gamma)$.

Proof. Throughout this proof, $\bar{\mathbf{x}}$ will always denote the tuple selected by T' at step 1, and i the index selected at step 2. If, say, we consider the probability of this tuple being equal to a specific tuple $\bar{\mathbf{z}}$, we write $\Pr[\bar{\mathbf{x}} = \bar{\mathbf{z}}]$, etc.

If T' rejects, then in particular it selects $i \neq 0$. Recall that $\Pr[\bar{\mathbf{x}} \text{ is bad}] = \gamma$ by definition. By inspecting step 2 in the definition of T', and recalling that $\bar{\mathbf{x}}$ is bad means $|\mathsf{wrong}(\bar{\mathbf{x}})| > \frac{1}{\alpha}$, we have

$$\Pr[i \neq 0 \mid \bar{\mathbf{x}} \text{ is bad}] = \frac{|\mathsf{wrong}(\bar{\mathbf{x}})|}{k(\bar{\mathbf{x}})} = \frac{|\mathsf{wrong}(\bar{\mathbf{x}})|}{\max(|\mathsf{wrong}(\bar{\mathbf{x}})|, 16/\alpha)} \geq 1/16.$$

So we can conclude that $\Pr[i \neq 0] = \Pr[\bar{\mathbf{x}} \text{ is bad}] \cdot \Pr[i \neq 0 \mid \bar{\mathbf{x}} \text{ is bad}] \geq \gamma/16$. To complete the proof we will show that

$$\Pr[T' \text{ rejects} \wedge (\neg B) \mid i \neq 0] = \Omega(1). \tag{3}$$

We will do this by showing that for every fixed $i_0 \neq 0$ and $x_0 \in X$, conditioned on T' selecting $\bar{\mathbf{x}}$ and i such that $i = i_0$ and $\bar{\mathbf{x}}_i = x_0$, the test rejects (and B does not hold) with constant probability. Summing over all values of $i_0 \neq 0$ and $x_0 \in X$, this will complete the proof.

So let us fix an arbitrary $i = i_0 \neq 0$ and $x_0 \in X$. Denote $V_0 = V_{x_0,i_0}$, the set of tuples $\bar{\mathbf{z}}$ for which $\bar{\mathbf{z}}_{i_0} = x_0$ (as defined in Definition A.3). We will rely on the expansion of the Markov process $G_0 \stackrel{def}{=} G_{x_0,i_0}$ to show that the probability mass placed on tuples $\bar{\mathbf{z}} \in V_0$, for which $F(\bar{\mathbf{z}})_{i_0} \neq f(x_0)$, "spreads" after one step of G_0. Thus, starting from such an $\bar{\mathbf{x}}$, with high probability we arrive at an $\bar{\mathbf{x}}'$ for which $F(\bar{\mathbf{x}})_{i_0} \neq F(\bar{\mathbf{x}}')_{i_0}$. To do this, we will partition the tuples $\bar{\mathbf{z}} \in V_0$ according to the value of $F(\bar{\mathbf{z}})_{i_0} = a$, and show that each part in the partition has many outgoing transitions, each causing T' to reject.

For every $a \in \Sigma$, let us define

$$S_a = \{\, \bar{\mathbf{z}} \in V_0 \mid F(\bar{\mathbf{z}})_{i_0} = a \}\,.$$

The set S_a also depends on the specific choice of x_0 and i_0, but this is omitted from the notation.

Let $E_{x_0,i_0,a}$ denote the event that T' selects $i = i_0$, and $\bar{\mathbf{x}}$ for which $\bar{\mathbf{x}}_i = x_0$ and $F(\bar{\mathbf{x}})_{i_0} = a$. Observe that, conditioned on a specific value $i = i_0$ selected at step 2 of T', not all tuples in V_0 have positive probability of being chosen at step 1 of T'. Indeed, a tuple $\bar{\mathbf{x}} \in V_0$ for which $F(\bar{\mathbf{x}})_{i_0} = f(x_0)$ cannot be chosen, because the conditioning requires in particular that $i_0 \in \mathsf{wrong}(\bar{\mathbf{x}})$, which is equivalent to

$F(\bar{\mathbf{x}})_{i_0} \neq f(x_0)$. So $\Pr_{T'}[E_{x_0,i_0,a}] > 0$ implies $a \neq f(x_0)$. In that case let P_a denote the probability distribution over V_0, defined by

$$\forall \bar{\mathbf{z}} \in V_0, \quad P_a(\bar{\mathbf{z}}) = \Pr_{T'}[\bar{\mathbf{x}} = \bar{\mathbf{z}} \mid E_{x_0,i_0,a}].$$

Fix some value $a \in \Sigma$ such that $\Pr[E_{x_0,i_0,a}] > 0$. Note that $\Pr[T' \text{ rejects} \wedge (\neg B) \mid E_{x_0,i_0,a}] \geq \Pr[T' \text{ rejects} \mid E_{x_0,i_0,a}] - \Pr[B \mid E_{x_0,i_0,a}]$. By Proposition A.7, the probability of the bad event B is bounded by $2/9$ even if conditioned on any specific $\bar{\mathbf{x}}$ and i, so in particular: $\Pr[B \mid E_{x_0,i_0,a}] < 2/9$. It is therefore sufficient to show that

$$\Pr[T' \text{ rejects} \mid E_{x_0,i_0,a}] \geq 1/4 \tag{4}$$

(as the probability in (3) will be lower bounded by $1/4 - 2/9 > 0$).

By the definition of T', we take a random step from $\bar{\mathbf{x}}$ according to the process G_0 and arrive at $\bar{\mathbf{x}}'$. The probability distribution over $\bar{\mathbf{x}}'$ is given by $A_0 P_a$, where A_0 is the transition matrix of G_0 and $P_a \in \mathbb{R}^{V_0}$ is a vector of probabilities that corresponds to the initial choice of $\bar{\mathbf{x}}$ conditioned on $E_{x_0,i_0,a}$. Finally, T' rejects if $F(\bar{\mathbf{x}}')_{i_0} \neq a$, or in other words, if $\bar{\mathbf{x}}' \notin S_a$. In conclusion,

$$\Pr[T' \text{ rejects} \mid E_{x_0,i_0,a}] = \Pr[A_0 P_a \notin S_a]. \tag{5}$$

Following is a brief outline of how we lower bound (5). We have already established that $a \neq f(x)$. This implies that S_a cannot contain more than half of the elements in V_0 (because f is the plurality; this is formally shown in Proposition A.10). Next, observe that if $\bar{\mathbf{x}}$ had been distributed uniformly in S_a, then by expansion, one step according to G_0 leaves this set with good probability. Our situation is slightly more complicated because $\bar{\mathbf{x}}$ is not distributed uniformly over S_a but rather according to P_a. Proposition A.11 proves that P_a is "uniform enough" (or rather, that it has "enough entropy"). We deduce our bound from a (known) variant of the expander mixing lemma (Proposition A.12) that can handle slightly skewed distributions.

PROPOSITION A.10. $|S_a| / |V_0| \leq 1/2$.

Proof. Recall the definition of the plurality function (Definition 4.9). Since $f(x_0) \neq a$, we have that for less than half of $\{ (\bar{\mathbf{z}}, j) \mid j \in [\ell],\ \bar{\mathbf{z}} \in V_{x_0,j} \}$ it holds that $F(\bar{\mathbf{z}})_j = a$. Since F is rotation consistent (namely, if $\bar{\mathbf{z}}'$ is obtained from $\bar{\mathbf{z}}$ using some cyclic shift of its ℓ components, then $F(\bar{\mathbf{z}}')$ can be obtained from $F(\bar{\mathbf{z}})$ in the same way), it is easy to verify that, for any particular value of j, less than half of $V_{x_0,j}$ have $F(\bar{\mathbf{z}})_j = a$. Fixing $j = i_0$, for less than half of $V_0 = V_{x_0,i_0}$, it holds that $F(\bar{\mathbf{z}})_{i_0} = a$. The proposition follows. ∎

We first show that the distribution P_a is not too far from being uniform over S_a.

PROPOSITION A.11. *Let $\lambda(A_0)$ be the second largest eigenvalue of A_0. Then*

$$\max_{\bar{\mathbf{z}}} \{P_a(\bar{\mathbf{z}})\} \leq \frac{1}{16\,|S_a|\,\lambda(A_0)^2}.$$

Proof. Recall from Proposition A.4 that $\lambda(A_0) = \lambda(A_{x_0,i_0}) = \alpha$, and that $\alpha^3 = 1/\ell$. By definition of P_a, we may rewrite what we are trying to prove as

$$\forall \bar{\mathbf{z}}, \quad \Pr[\bar{\mathbf{x}} = \bar{\mathbf{z}} \mid E_{x_0,i_0,a}] \leq \frac{\alpha\ell}{16} \frac{1}{|S_a|}.$$

Clearly this probability is zero for all $\bar{\mathbf{z}} \notin S_a$. The point is to show that even though we have fixed i_0 and x_0 and a, this does not give too much information about $\bar{\mathbf{x}}$. We

will use Bayes' rule,

$$\Pr[\bar{\mathbf{x}} = \bar{\mathbf{z}} \,|\, E_{x_0,i_0,a}] = \Pr[E_{x_0,i_0,a} \,|\, \bar{\mathbf{x}} = \bar{\mathbf{z}}] \cdot \frac{\Pr[\bar{\mathbf{x}} = \bar{\mathbf{z}}]}{\Pr[E_{x_0,i_0,a}]}. \tag{6}$$

We now estimate each of the three factors on the right-hand side. Surely, with no conditioning, $\Pr[\bar{\mathbf{x}} = \bar{\mathbf{z}}] = 1/\left|\overline{X}\right|$. To estimate $\Pr[E_{x_0,i_0,a} \,|\, \bar{\mathbf{x}} = \bar{\mathbf{z}}]$ observe that, conditioned on $\bar{\mathbf{x}} = \bar{\mathbf{z}}$, the random choice of the index i (at step 2 of T') determines whether the event $E_{x_0,i_0,a}$ occurs or not (because $\bar{\mathbf{z}}$ and $F(\bar{\mathbf{z}})$ and therefore $\bar{\mathbf{z}}_i$ and $F(\bar{\mathbf{z}})_i$ are already fixed). The probability, for a given $\bar{\mathbf{x}} = \bar{\mathbf{z}}$, of selecting $i = i_0$ at step 2 of T' is exactly $1/k(\bar{\mathbf{z}})$. If i_0 was selected, the event occurs iff $\bar{\mathbf{z}} \in S_a$ (which means that $\bar{\mathbf{z}}_{i_0} = x_0$ and $F(\bar{\mathbf{z}})_{i_0} = a$). Thus, $\Pr[E_{x_0,i_0,a} \,|\, \bar{\mathbf{x}} = \bar{\mathbf{z}}] = 1/k(\bar{\mathbf{z}}) \leq \alpha/16$ if $\bar{\mathbf{z}} \in S_a$ and equals zero otherwise.

Finally, we estimate $\Pr[E_{x_0,i_0,a}]$. We write

$$\Pr[E_{x_0,i_0,a}] = \sum_{\bar{\mathbf{z}} \in \overline{X}} \Pr[E_{x_0,i_0,a} \,|\, \bar{\mathbf{x}} = \bar{\mathbf{z}}] \cdot \Pr[\bar{\mathbf{z}}].$$

Just as before, $\Pr[E_{x_0,i_0,a} \,|\, \bar{\mathbf{x}} = \bar{\mathbf{z}}] = 1/k(\bar{\mathbf{z}})$ if $\bar{\mathbf{z}} \in S_a$ and 0 otherwise. Plugging this into the above equation gives $\Pr[E_{x_0,i_0,a}] = \sum_{\bar{\mathbf{z}} \in S_a} \frac{1}{k(\bar{\mathbf{z}})} \cdot \Pr[\bar{\mathbf{z}}] = \sum_{\bar{\mathbf{z}} \in S_a} \frac{1}{k(\bar{\mathbf{z}})} \cdot \frac{1}{|\overline{X}|} \geq \frac{|S_a|}{|\overline{X}|} \cdot \frac{1}{\ell}$, because always $k(\bar{\mathbf{z}}) \leq \ell$. Plugging everything into (6), we find

$$\Pr[\bar{\mathbf{x}} = \bar{\mathbf{z}} \,|\, E_{x_0,i_0,a}] \leq \frac{\alpha}{16} \cdot \frac{\frac{1}{|\overline{X}|}}{\frac{|S_a|}{|\overline{X}|} \cdot \frac{1}{\ell}} = \frac{\alpha\ell}{16} \cdot \frac{1}{|S_a|}. \qquad \square$$

It remains to observe that, for every vector $\vec{p} = (p_1, \ldots, p_n)$, if $\sum_i p_i = 1$, then

$$\|\vec{p}\|_2^2 = \sum_i p_i^2 \leq \sum_i p_i \cdot \max_i p_i = \max_i p_i = \|\vec{p}\|_\infty.$$

So, in particular, the previous proposition gives $\|P_a\|_2^2 \leq \max_{\bar{\mathbf{z}}} P_a(\bar{\mathbf{z}}) \leq 1/(16\,|S_a|\,\lambda(A_0)^2)$.

We can now conclude the proof of Lemma A.9 and of the theorem by the following proposition, which is a slight generalization of the standard expander mixing lemma (for the case that the initial distribution is not uniform over some subset of the sample space).

PROPOSITION A.12. $\Pr[A_0 P_a \in S_a] \leq |S_a| / |V_0| + \lambda(A_0)\sqrt{|S_a| \cdot \|P_a\|_2^2}$.

Proof. Let $\chi_a \in \{0,1\}^{|V_0|}$ be the characteristic vector of S_a in V_0 (i.e., for every $\bar{\mathbf{z}} \in V_0$ we set $\chi_a(\bar{\mathbf{z}}) = 1$ iff $\bar{\mathbf{z}} \in S_a$). Let $U \in \{0,1\}^{|V_0|}$ be the vector corresponding to the uniform distribution, where all the entries in U equal $1/|V_0|$. As usual, we break P_a into two components, parallel to U and perpendicular to U. Let $P_a^\perp = P_a - U$. Note that $P_a^\perp$ is perpendicular to U because

$$\left\langle P_a^\perp, U \right\rangle = \langle P_a, U \rangle - \langle U, U \rangle = \sum_{\bar{\mathbf{x}} \in V_0} P_a(\bar{\mathbf{x}}) \cdot \frac{1}{|V_0|} - \sum_{\bar{\mathbf{x}} \in V_0} \frac{1}{|V_0|} \cdot \frac{1}{|V_0|} = \frac{1}{|V_0|}(1-1) = 0,$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product. Now we have that

$$\begin{aligned} \Pr[A_0 P_a \in S_a] &= \langle \chi_a, A_0 P_a \rangle \\ &= \langle \chi_a, A_0 U \rangle + \left\langle \chi_a, A_0 P_a^\perp \right\rangle. \end{aligned}$$

Since $A_0 U = U$, the first summand becomes $|S_a| / |V_0|$. To bound the second summand, we use Cauchy–Schwarz inequality, and get that $\left\langle \chi_a, A_0 P_a^\perp \right\rangle \leq \|\chi_a\|_2 \cdot \|A_0 P_a^\perp\|_2$

$\leq \sqrt{|S_a|} \cdot \|A_0 P_a^{\perp}\|_2$. The proposition follows since $\lambda(A_0)$ is the second largest eigenvalue of A_0,

$$\|A_0 P_a^{\perp}\|_2 \leq \|\lambda(A_0) P_a^{\perp}\|_2 \leq \lambda(A_0)\|P_a\|_2. \qquad \square$$

Using $|S_a| / |V_0| \leq \frac{1}{2}$ and $\lambda(A_0)\sqrt{|S_a| \cdot \|P_a\|_2^2} \leq \lambda(A_0)\sqrt{(1/16) \cdot \lambda(A_0)^2} \leq \frac{1}{4}$ (as established in Propositions A.10 and A.11), we lower bound (5) by $1 - (\frac{1}{2} + \frac{1}{4}) = \frac{1}{4}$ as needed. This concludes the proof of (4) and therefore of Lemma A.9, showing that $\Pr[T' \text{ rejects} \wedge (\neg B)] = \Omega(\gamma)$. $\square$

Combining Lemma A.9 with Lemma A.8, we obtain $\Pr[T \text{ rejects}] = \Omega(\Pr[T' \text{ rejects} \wedge (\neg B)]) = \Omega(\gamma)$, and the theorem follows. $\square$

Appendix B. Strengthening the black-box assignment tester from section 5.

Proof of Corollary 5.3. We would like to compose $\mathcal{A}_\beta$ with itself a constant number of times to reduce the size of the output circuits from $O(\mathsf{n}^{1-\beta})$ to $O(\mathsf{n}^{\alpha})$. For the composition of $\mathcal{A}_\beta$ with itself to be well defined we need the distance parameter to be small enough with respect to the number of queries. To do that we first apply to $\mathcal{A}_\beta$ the distance-reduction transformation given by Lemma 4.1, reducing its distance parameter to another constant $\delta' < 1/(3c_2)$, where c_2 is the constant from the composition theorem, Theorem 3.7. We then reduce the number of queries to three, as was done in Proposition 4.11. The result of these two steps is an assignment tester $\mathcal{A}'_\beta$ with parameters $(\mathsf{R}(\mathsf{n}) = \mathsf{n}^{O(1)},\ \mathsf{s}(\mathsf{n}) = O(\mathsf{n}^{1-\beta}),\ \mathsf{q}(\mathsf{n}) = 3,\ \delta(\mathsf{n}) = \delta',\ \varepsilon(\mathsf{n}) = \bar{\varepsilon})$, with $\bar{\varepsilon}$ being some fixed constant.

It is now possible to compose $\mathcal{A}'_\beta$ with itself, and doing it a constant number of times (which depend on α, of course) will reduce the output circuit size to $O(\mathsf{n}^{\alpha})$. This almost gives us the assignment tester we are after, with the only problem being that the new error is some small constant that depends on α. Reducing the error parameter to $\varepsilon_\alpha = 0.1$ using Theorem 4.8, will complete our proof, as we now have that the circuit size is indeed $O(\mathsf{n}^{\alpha})$, and the only parameter that depends on α is the number of circuits produced (which is polynomial for every fixed α). $\square$

Acknowledgments. We sincerely thank the authors of [8]—Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan—for sharing with us preliminary stages of their own work, and for making excellent suggestions for ours. We thank Sanjeev Arora, Amir Shpilka, Luca Trevisan, and Avi Wigderson for many useful discussions and comments. We would like to thank William Hesse and Ilan Newman for very helpful discussions regarding the circuit size of universal circuits. Special thanks too to Oded Goldreich for his valuable contribution to the presentation of this paper.

REFERENCES

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and the hardness of approximation problems*, J. ACM, 45 (1998), pp. 501–555.

[2] S. Arora and S. Safra, *Probabilistic checking of proofs: A new characterization of NP*, J. ACM, 45 (1998), pp. 70–122.

[3] L. Babai, *Trading group theory for randomness*, in Proceedings of the 17th ACM Symposium on Theory of Computing, Providence, RI, 1985, ACM, New York, 1985, pp. 421–429.

[4] L. Babai, L. Fortnow, L. Levin, and M. Szegedy, *Checking computations in polylogarithmic time*, in Proceedings of the 23rd ACM Symposium on Theory of Computing, New Orleans, LA, 1991, ACM, New York, 1991, pp. 21–31.

[5] L. Babai, L. Fortnow, and C. Lund, *Non-deterministic exponential time has two-prover interactive protocols*, Comput. Complexity, 1 (1991), pp. 3–40.

[6] M. Bellare, O. Goldreich, and M. Sudan, *Free bits, PCPs, and nonapproximability—Towards tight results*, SIAM J. Comput., 27 (1998), pp. 804–915.
[7] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson, *Multi prover interactive proofs: How to remove intractability assumptions*, in Proceedings of the 20th ACM Symposium on Theory of Computing, Chicago, IL, 1988, ACM, New York, 1988, pp. 113–121.
[8] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan, *Robust PCPs of proximity, shorter pcps and applications to coding*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, 2004, ACM, New York, 2004, pp. 1–10.
[9] M. Capalbo, O. Reingold, S. Vadhan, and A. Wigderson, *Randomness conductors and constant-degree lossless expanders*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, Montréal, QC, 2002, ACM, New York, 2002, pp. 659–668.
[10] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press/McGraw-Hill, New York, 1990.
[11] I. Dinur, *The PCP theorem by gap amplification*, in Proceedings of the 38th ACM Symposium on Theory of Computing, 2006, ACM, New York, pp. 241–250.
[12] F. Ergün, R. Kumar, and R. Rubinfeld, *Fast approximate PCPs*, in Proceedings of the 31st ACM Symposium on Theory of Computing, Atlanta, GA, 1999, ACM, New York, 1999, pp. 41–50.
[13] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, *Approximating clique is almost NP-complete*, J. ACM, 43 (1996), pp. 268–292.
[14] U. Feige and J. Kilian, *Two prover protocols—Low error at affordable rates*, in Proceedings of the 26th ACM Symposium on Theory of Computing, Montréal, QC, 1994, ACM, New York, 1994, pp. 172–183.
[15] L. Fortnow, J. Rompel, and M. Sipser, *On the power of multi-prover interactive protocols*, Theoret. Comput. Sci., 134 (1994), pp. 545–557.
[16] R. G. Gallager, *Low Density Parity Check Codes*, MIT Press, Cambridge, MA, 1963.
[17] O. Goldreich, *A sample of samplers—A computational perspective on sampling (survey)*, Electronic Colloquium on Computational Complexity (ECCC), 4 (1997); available online at http://eccc.hpi-web.de/eccc-reports/1997/TR97-020/index.html.
[18] O. Goldreich, S. Goldwasser, and D. Ron, *Property testing and its connection to learning and approximation*, J. ACM, 45 (1998), pp. 653–750.
[19] O. Goldreich and S. Safra, *A combinatorial consistency lemma with application to proving the PCP theorem*, SIAM J. Comput., 29 (2000), pp. 1132–1154.
[20] S. Goldwasser, S. Micali, and C. Rackoff, *The knowledge complexity of interactive proofs*, SIAM J. Comput., 18 (1989), pp. 186–208.
[21] A. Lubotzky, R. Phillips, and P. Sarnak, *Ramanujan graphs*, Combinatorica, 8 (1988), pp. 261–277.
[22] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, *Algebraic methods for interactive proof systems*, J. ACM, 39 (1992), pp. 859–868.
[23] A. Polishchuk and D. Spielman, *Nearly linear size holographic proofs*, in Proceedings of the 26th ACM Symposium on Theory of Computing, Montréal, QC, 1994, ACM, New York, 1994, pp. 194–203.
[24] R. Raz, *A parallel repetition theorem*, SIAM J. Comput., 27 (1998), pp. 763–803.
[25] O. Reingold, S. Vadhan, and A. Wigderson, *Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors*, in Proceedings of the 41st Annual Symposium on Foundations of Computer Science, Redondo Beach, CA, 2000, IEEE Press, Piscataway, NJ, 2000, pp. 3–13.
[26] R. Rubinfeld and M. Sudan, *Robust characterizations of polynomials with applications to program testing*, SIAM J. Comput., 25 (1996), pp. 252–271.
[27] A. Shamir, IP = PSPACE, J. ACM, 39 (1992), pp. 869–877.
[28] M. Sipser and D. A. Spielman, *Expander codes*, IEEE Trans. Inform. Theory, 42 (1996), pp. 1710–1722.
[29] D. A. Spielman, *Linear-time encodable and decodable error-correcting codes*, IEEE Trans. Inform. Theory, 42 (1996), pp. 1723–1731.
[30] M. Szegedy, *Many-valued logics and holographic proofs*, in Proceedings of ICALP'99, Prague, Czech Republic, 1999, Lecture Notes in Comput. Sci. 1644, Springer, New York, 1999, pp. 676–686.
[31] R. M. Tanner, *A recursive approach to low complexity codes*, IEEE Trans. Inform. Theory, IT-27 (1981), pp. 533–547.

SIAM J. COMPUT.
Vol. 36, No. 4, pp. 1025–1071

RULING OUT PTAS FOR GRAPH MIN-BISECTION, DENSE k-SUBGRAPH, AND BIPARTITE CLIQUE*

SUBHASH KHOT†

Abstract. Assuming that NP $\not\subseteq \cap_{\epsilon>0}$ BPTIME(2^{n^ϵ}), we show that graph min-bisection, dense k-subgraph, and bipartite clique have no polynomial time approximation scheme (PTAS). We give a reduction from the minimum distance of code (MDC) problem. Starting with an instance of MDC, we build a *quasi-random probabilistically checkable proof (PCP)* that suffices to prove the desired inapproximability results. In a quasi-random PCP, the query pattern of the verifier *looks random* in a certain precise sense. Among the several new techniques we introduce, the most interesting one gives a way of certifying that a given polynomial belongs to a given linear subspace of polynomials. As is important for our purpose, the certificate itself happens to be another polynomial, and it can be checked probabilistically by reading a constant number of its values.

Key words. probabilistically checkable proofs (PCPs), hardness of approximation, approximation algorithms

AMS subject classifications. 68Q17, 68Q25

DOI. 10.1137/S0097539705447037

1. Introduction.

Several optimization problems of theoretical and practical importance are NP-hard, meaning there is no efficient (i.e., polynomial time) algorithm to obtain exact solutions to these problems unless P = NP. However, for many of these problems, it is possible to obtain approximate solutions efficiently. An approximation algorithm with ratio C computes a solution that is guaranteed to be within a factor of C of the optimal solution (the approximation ratio for minimization and maximization problems is defined appropriately so that $C > 1$). In the best scenario, a problem may have an approximation algorithm with ratio $1+\epsilon$ for arbitrarily small constant $\epsilon > 0$. Many packing and scheduling problems have this property (e.g., the knapsack problem), and they are said to have a polynomial time approximation scheme (PTAS). For an excellent treatment of approximation algorithms, please refer to Vazirani's book [37].

Existence of a PTAS is difficult to prove in general. For example, it wasn't known till the early 1990s whether MAX-3SAT had a PTAS. The celebrated probabilistically checkable proof (PCP) theorem [4], [7] settled this question in the negative. Such results that rule out the possibility of an approximation algorithm within a certain factor (assuming of course a hypothesis like P $\neq$ NP) are known as inapproximability results or hardness results. After the discovery of the PCP theorem, there was tremendous progress in proving hardness results for various problems. A very incomplete list includes [6], [8], [16] [25], [26], [18], [15], [23], [13], [14], [28]. For some problems like MAX-3SAT [26], clique [25], and set-cover [16], optimal hardness results have been proved. A recent survey of Trevisan [36] gives a nice exposition for this area (see also [3]).

*Received by the editors February 15, 2005; accepted for publication (in revised form) January 2, 2006; published electronically December 15, 2006. This material is based upon work supported by the National Science Foundation under agreement DMS-0111298. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

http://www.siam.org/journals/sicomp/36-4/44703.html

†Georgia Institute of Technology, Atlanta, GA 30332 (khot@cc.gatech.edu). This work was done while the author was at the Institute for Advanced Study.

The PCP theorem has an equivalent formulation in terms of probabilistic checking of proofs. The theorem states that every NP-statement has a short proof that can be checked very efficiently by a probabilistic verifier. The verifier reads only a constant number of bits from the proof and has the following completeness and soundness properties: every correct statement has a proof that the verifier accepts with probability 1, and *every* proof of an incorrect statement is rejected with high probability. The connection between the notion of probabilistic proof checking and inapproximability results for optimization problems has been one of the most exciting developments in theoretical computer science.

In spite of great success of PCP tools towards proving inapproximability results, there are some notorious problems which so far have resisted all attempts to prove *good* inapproximability results. Examples include vertex cover, traveling salesman, and graph coloring. In this paper, we make progress on one set of notorious problems, namely, the graph min-bisection, dense k-subgraph, and bipartite clique problems. The best-known algorithms for these problems have approximation ratios $O(\log^2 n)$ (see [20]), $O(n^{1/3})$ (see [21]), and $n/(\log n)^{O(1)}$ (folklore), respectively, whereas no inapproximability results were known. Ruling out a PTAS for these problems has been considered a major open question in inapproximability theory.

Recently, Feige [17] showed that these problems have no PTAS, assuming a certain hypothesis about average-case hardness of random 3SAT (see section 1.7). The main result in this paper rules out a PTAS for these problems assuming only a (fairly) standard assumption that NP $\not\subseteq \cap_{\epsilon>0}$ BPTIME(2^{n^ϵ}) (i.e., NP does not have randomized algorithms that run in subexponential time).

Typically, inapproximability results are proved by *reducing* a known NP-hard problem to the target problem. The PCP theorem itself can be viewed as a (rather sophisticated) NP-hardness reduction. We too obtain our result via a reduction. We use a reduction from the minimum distance of linear code (MDC) problem. The inapproximability result for MDC has been obtained by Dumer, Micciancio, and Sudan [15] (see section 1.2).

1.1. Problem definitions and results.

Graph min-bisection. Given a graph $G(V, E)$ with $|V|$ even, partition V into two equal parts V' and V'' so as to minimize the number of *crossing* edges, i.e., edges with one endpoint each in V' and V''.

Dense k-subgraph. Given a graph $G(V, E)$ and a parameter k, find a subset $V' \subseteq V$ of size k so as to maximize the number of edges whose both endpoints are in V'.

Bipartite clique. Given a bipartite graph $G(V, W, E)$, maximize k such that there exist $V' \subseteq V$ and $W' \subseteq W$, each of size k, and the subgraph of G induced on the set of vertices $V' \cup W'$ is a complete bipartite graph.

Graph min-bisection is used as a subroutine by many graph algorithms based on divide-and-conquer strategy. Such algorithms partition the given graph into two equal parts, solve the desired optimization problem recursively on the two parts, and then *merge* the two solutions to obtain an overall solution (it usually suffices to partition the graph into two pieces of comparable size instead of exactly equal size). Dense k-subgraph and bipartite clique are natural graph theoretic problems which may also have algorithmic applications (e.g., the bipartite clique problem is the same as finding large monochromatic 1-squares in a 0-1 array; this may have applications to vision in processing arrays of pixels).

The main result in this paper is an inapproximability result for the above three problems. A formal statement of the result follows.

THEOREM 1.1. *Let $\epsilon > 0$ be an arbitrarily small constant. Assume that SAT does not have a probabilistic algorithm that runs in time 2^{n^ϵ} on an instance of size n. Then there is no polynomial time (possibly randomized) algorithm for graph min-bisection, dense k-subgraph, or bipartite clique that achieves an approximation ratio of $1 + \frac{1}{2^{O(1/\epsilon \log(1/\epsilon))}}$. In particular, assuming* NP $\not\subseteq \cap_{\epsilon>0}$ BPTIME (2^{n^ϵ})*, these three problems have no PTAS.*

Note that the inapproximability factor $1 + \frac{1}{2^{O(1/\epsilon \log(1/\epsilon))}}$ approaches 1 as ϵ tends to 0. Thus we have a (strange) trade-off between the quality of the inapproximability result and the complexity assumption that SAT has no 2^{n^ϵ} time algorithm. The inapproximability factor for bipartite clique can be *boosted* via randomized graph products (this is done as in [10], [11]; a proof appears in Appendix D). This gives the following theorem.[1]

THEOREM 1.2. *Let $\epsilon > 0$ be an arbitrarily small constant. Assume that SAT does not have a probabilistic algorithm that runs in time 2^{n^ϵ} on an instance of size n. Then there is no polynomial time (possibly randomized) algorithm for bipartite clique that achieves an approximation ratio of $N^{\epsilon'}$ on graphs of size N where $\epsilon' = \frac{1}{2^{O(1/\epsilon \log(1/\epsilon))}}$.*

We would like to mention that, independent of our work, Feige and Kogan [19] obtained the following (weaker) results for the bipartite clique problem.

THEOREM 1.3 (see [19]). *The bipartite clique problem is hard to approximate within factor $2^{(\log n)^\delta}$ for some $\delta > 0$ assuming that 3-SAT $\notin$ DTIME$(2^{n^{3/4+\epsilon}})$ for some $\epsilon > 0$. Also, the problem is NP-hard to approximate within some constant factor if the clique problem is NP-hard to approximate within factor $n/2^{c\sqrt{\log n}}$ for some $c > 0$.*

1.2. Minimum distance of code problem. As mentioned earlier, our hardness results are proved via a reduction from the minimum distance of (linear) code problem that we state next.

Minimum distance of code (MDC). Given a $N \times n'$ matrix $\mathbf{A}$ over a field $\mathbb{F}$, find a nonzero vector $\mathbf{z} \in \mathbb{F}^{n'}$ that minimizes the fraction of nonzero coordinates in the vector $\mathbf{Az}$. If one interprets the columns of $\mathbf{A}$ as the basis for a linear code, the problem is to find the (relative) minimum distance of the code. Let $OPT(\mathbf{A})$ denote the optimum value.

Remark 1.4. All instances of MDC in this paper are assumed to have the following properties: (1) The columns of $\mathbf{A}$ are linearly independent and $N \geq n'$; otherwise the problem becomes trivial. (2) The field $\mathbb{F}$ has characteristic 2. This enables us to concatenate the code with a binary code. (3) $N \leq |\mathbb{F}| \leq N^2$.

A result of Dumer, Micciancio, and Sudan [15] says that MDC is NP-hard to approximate within any constant factor on binary asymptotically good codes. The following result can be obtained from Dumer et al.'s result (a proof appears in Appendix B). It is the starting point for our reduction.

THEOREM 1.5. *There exists an absolute constant C such that the following holds. For every integer K, there is a reduction from SAT to MDC satisfying the following (think of K as a large constant):*

1. *The MDC instance is over a field $\mathbb{F}$ of characteristic 2 and has N rows, and $N \leq |\mathbb{F}| \leq N^2$.*
2. *(Completeness.) YES instance of SAT is mapped to an MDC instance $\mathbf{A}$ with $OPT(\mathbf{A}) \leq \frac{1}{2^K}$.*

[1] The FOCS 2004 version of this paper contains a bug. We erroneously claimed that ϵ' could be made independent of ϵ in Theorem 1.2.

3. *(Soundness.) NO instance of SAT is mapped to an MDC instance* $\mathbf{A}$ *with* $OPT(\mathbf{A}) \geq 1 - \frac{1}{2^{2^K}}$.
4. *The reduction runs in time* n^{CK} *and, in particular,* $N \leq n^{CK}$.

We would like to note that Dumer et al.'s reduction and therefore reduction in Theorem 1.5 is randomized (the completeness and soundness properties hold with high probability). The PCPs in this paper are constructed from Theorem 1.5 and their construction is randomized too, though we do not explicitly state so.

1.3. PCPs. We state the PCP theorem for future reference. The theorem was proved by Arora and Safra [4] and Arora et al. [7]. See Arora's Ph.D. thesis [2] for a comprehensive introduction to the subject. The PCP theorem states that languages in NP have short membership proofs that can be efficiently checked by a probabilistic constant-query verifier.

THEOREM 1.6 (PCP theorem). *For every language* $L \in$ NP*, there is a polynomial time probabilistic verifier* V *that has access to input* x *and a supposed proof* Π*. The size of proof* Π *is polynomial in the size of the input* $|x| = n$*. The verifier uses* $O(\log n)$ *random bits, reads* $d = O(1)$ *queries from* Π *(choice of query locations depends on verifier's random bits), performs a predetermined test on the query bits, and then accepts or rejects. The verifier has the following completeness and soundness properties:*

- *(Completeness/YES case.)* $x \in L \implies \exists\ \Pi \quad s.t. \quad \Pr[V(\Pi) = accept] = 1.$
- *(Soundness/NO case.)* $x \notin L \implies \forall\ \Pi \quad s.t. \quad \Pr[V(\Pi) = accept] \leq \frac{1}{10}.$

Here is an equivalent formulation of the PCP theorem as an inapproximability result.

THEOREM 1.7 (PCP theorem). *There exists an absolute constant* $c < 1$ *and a polynomial time reduction from* 3*SAT to MAX-*3*SAT mapping instance* ϕ *of* 3*SAT to instance* ψ *of MAX-*3*SAT with these properties:*

- *(YES case.) If* ϕ *is satisfiable, then so is* ψ*, i.e.,* $OPT(\psi) = 1$.
- *(NO case.) If* ϕ *is unsatisfiable, then* $OPT(\psi) \leq c$*, i.e., no assignment to* ψ *satisfies more than a fraction* c *of its clauses.*

*In particular, MAX-*3*SAT is hard to approximate within a factor better than* $1/c$ *unless* P = NP.

1.4. Quasi-random PCPs. In this section, we introduce the notion of quasi-random PCPs. The main contribution of the paper is a construction of a quasi-random PCP that immediately implies the inapproximability results for problems in Theorem 1.1. In section 1.5, we explain why the known PCPs are not quasi-random, and in section 1.6, we explain our new ideas for building quasi-random PCPs.

Quasi-random PCPs focus on the distribution of queries made by the verifier. We require that the distribution depends on whether the input to the PCP verifier is a YES input or a NO input. In the NO case, the queries are required to be distributed *randomly* over the proof, and in the YES case, the distribution is required to be *far from being random*. This is quite counterintuitive and mystical at first sight. The verifier does not know whether the input is YES or NO; then how can he make his query pattern depend on the YES/NO case? However, looking at some known PCPs makes this less mysterious. For example, let us look at Holmerin's [27] 4-query PCP, which has the following properties: (i) In the YES case, there exists a set of half the bits in the proof such that no PCP test accesses all 4 queries from this set. (ii) In the NO case, for any set of half the bits in the proof, at least $\frac{1}{16} - \epsilon$ fraction of the tests access all 4 queries from this set.

Thus it is conceivable (and necessary for proving inapproximability results) that the query pattern depends on the YES/NO case, without the verifier knowing which case it is. The quasi-random PCPs deal with one specific random-like property of query pattern. Here is the definition.

DEFINITION 1.8 (quasi-random PCP). *A PCP with d nonadaptive Boolean queries is called quasi-random if*[2]

- *in the YES case there exists a set of half the bits Π_0 in the proof Π such that at least $(1-o(1))\frac{1}{2^{d-1}}$ fraction of tests accesses all d queries from Π_0;*
- *in the NO case for any set Π_* of half the bits in the proof Π, the fraction of tests that accesses all d queries from Π_* is $(1\pm o(1))\frac{1}{2^d}$.*

Thus the query pattern has random-like property in the NO case, and in the YES case the property is violated.

Note that the definition of quasi-random PCPs includes conditions in both the YES and NO cases. However, by abuse of terminology, we will often refer only to the condition in the NO case. The following theorem states the main PCP construction in the paper.

THEOREM 1.9. *For every $\epsilon > 0$, there exists an integer $d = O(1/\epsilon \log(1/\epsilon))$ such that there is a PCP verifier for SAT instance of size n satisfying the following:*

1. *The proof Π for the verifier is of size 2^{n^ϵ}.*
2. *The verifier queries d bits from the proof. Let Q denote the set of query bits.*
3. *Every query is uniformly distributed over Π (two different queries are of course correlated).*
4. *(YES case/completeness.) Suppose the SAT instance is a YES instance and Π is a correct proof. Let Π_0 be the set of 0-bits in the proof (it contains half the bits from the proof). Then*

$$\Pr_Q[Q \subseteq \Pi_0] \geq (1 - O(1/d))\ \frac{1}{2^{d-1}}.$$

   The probability is taken over the random test of the verifier.
5. *(NO case/soundness.) Suppose the SAT instance is a NO instance, and let Π_* be any set of half the bits from Π. Then*

$$\frac{1}{2^d} - \frac{1}{2^{20d}} \ \leq\ \Pr_Q[Q \subseteq \Pi_*] \ \leq\ \frac{1}{2^d} + \frac{1}{2^{20d}}.$$

Note that Theorem 1.9 says nothing about the test the verifier performs on the query bits or the acceptance probability in completeness/soundness case. When we actually construct the PCP, we will use a homogeneous linear test on query bits.

The inapproximability results for graph min-bisection, dense k-subgraph, and bipartite clique follow easily from the PCP in Theorem 1.9 (see Appendix C). The reductions follow the same outline as Feige's reductions [17].

In the next two sections, we explain why the known PCPs are not quasi-random and give our new ideas for building quasi-random PCPs.

1.5. Why known PCPs are not quasi-random. Consider Håstad's 3-query PCP [26] as an illustration. The PCP can be abstracted out in the following manner:

[2]This definition is specific to the PCP that we construct in this paper. Using PCPs over alphabet $\mathbb{F}_p$ (rather than $\mathbb{F}_2$), the techniques in this paper can be extended to construct PCPs where the gap in the YES and NO cases is $\approx p$. To be precise, in the YES case, there is a set of $1/p$ fraction of positions in the proof such that $\approx 1/p^{d-1}$ fraction of tests falls into this set. In the NO case, for any set of $1/p$ fraction of positions in the proof, the fraction of tests that falls into this set is $\approx 1/p^d$.

The proof Π is partitioned into blocks:

$$\Pi = (A_1, A_2, \ldots, A_M\ ; B_1, B_2, \ldots, B_{M'}).$$

Every block A_i, B_j is of constant length and is supposed to be a long code. This is, however, irrelevant for the present discussion. The blocks B_j's are *bigger* than the blocks A_i's. The verifier picks a pair of blocks (A_i, B_j) and reads three bits from the proof: one from block A_i and two from block B_j.

It is easy to see why Håstad's PCP fails to be quasi-random (to be precise, it fails the NO case of Definition 1.8). Construct a set Π_* containing half the bits from proof Π as follows: for every block A_i or B_j, we include the entire block in Π_* with probability $\frac{1}{2}$ and exclude it from Π_* with probability $\frac{1}{2}$. Clearly, the probability that all three queries access bits from Π_* equals $\frac{1}{4}$, whereas the quasi-randomness condition requires this probability to be close to $\frac{1}{8}$.

Why does Håstad's verifier access two bits from the bigger block B_j? When one analyzes the PCP using Fourier analysis, it is necessary that the Fourier coefficient of the block B_j appears in squared form. Reading two bits from block B_j is a way to introduce redundancy and ensure that we have the Fourier coefficient $\widehat{B}_j^2$. However, as we observed, this form of redundancy leads to loss of quasi-randomness.

Though the above discussion applies only to one specific PCP (namely, Håstad's PCP) and only one specific analysis technique (namely, Fourier analysis), it turns out that all known PCPs and known techniques suffer from the same problem. On the high level, the proof is partitioned into blocks, the verifier reads q queries, and reads more than one query from at least one block. This typically happens when the verifier, instead of reading a desired bit, reads two or more related bits and infers the value of the desired bit via error correction. (PCPs, almost by definition, must use error correction in some form. See, for example, Hadamard code testing [12] and long code testing [8].) In particular, the verifier accesses less than q blocks and therefore fails to be quasi-random. Let us formulate this as an observation.

FACT 1.10. *Any q-query PCP where the proof is partitioned into blocks and the verifier accesses less than q blocks is not quasi-random.*

Thus we do need redundant bits and some form of error correction; but at the same time, we do not want to store all these bits at the same physical location (e.g., the same block B_j as in Håstad's PCP).

1.6. Basic idea behind our PCPs. Now we explain how we achieve the seemingly impossible task of constructing a quasi-random PCP. Here we present only the basic idea that has to fit in a much bigger context.

We look at the low degree test, which is an important building block of the PCP theorem (see [35], [4]). Given a function $f : \mathbb{F}^m \mapsto \mathbb{F}$, the test verifies that f is a degree $d-1$ polynomial (think of d as a constant). The verifier has access to a table of values of f, call it a *points table*. In addition, the verifier has access to a *lines table* which contains, for every line ℓ in $\mathbb{F}^m$, a degree $d-1$ univariate polynomial $f|_\ell$. The polynomial $f|_\ell$ is supposed to be the restriction of f to the line ℓ. The verifier works as follows: Pick a random line ℓ and a random point $\overline{v}$ on the line. Read $f(\overline{v})$ from the points table and read the polynomial $f|_\ell$ from the lines table. Accept if and only if $f(\overline{v})$ and $f|_\ell$ are consistent, i.e., if and only if the value produced by $f|_\ell$ at point $\overline{v}$ is the same as the value $f(\overline{v})$.

In our PCP construction, however, the verifier has no access to the additional lines table (this feature is also used independently in a recent paper of Ben-Sasson et al. [9]). Instead, the verifier proceeds as follows:

- Pick a random line ℓ and $d+1$ random points $\overline{v}_1, \overline{v}_2, \ldots, \overline{v}_{d+1}$ on the line. Read the values $f(\overline{v}_1), f(\overline{v}_2), \ldots, f(\overline{v}_{d+1})$.
- Let $f|_\ell$ be the unique degree $d-1$ univariate polynomial that takes values $f(\overline{v}_2), f(\overline{v}_3), \ldots, f(\overline{v}_{d+1})$ at points $\overline{v}_2, \overline{v}_3, \ldots, \overline{v}_{d+1}$, respectively.
- Accept if and only if $f(\overline{v}_1)$ and $f|_\ell$ are consistent.

Thus the verifier reads $f(\overline{v}_1)$ and then reads additional d random points on the line and *reconstructs* the line polynomial. This is the so-called outer verifier.

Now let us see why this PCP is quasi-random (the queries are nonboolean though). Let Π_* be any set of half the points from $\mathbb{F}^m$. What is the probability that all $d+1$ queries of the verifier fall in Π_*? It is easy to show that lines in $\mathbb{F}^m$ have excellent *mixing property*. For almost all lines ℓ, the fraction of points on ℓ that are in Π_* is very close to $\frac{1}{2}$. Once the line is chosen, the verifier picks $d+1$ random points on the line. Therefore, the probability that all the queries fall in Π_* is very close to $\frac{1}{2^{d+1}}$ and the PCP is quasi-random!

However, the queries are elements of $\mathbb{F}$, whereas we want to have boolean queries. We assume that $\mathbb{F}$ has characteristic 2 and therefore the field elements can be represented as bit strings of length $\log|\mathbb{F}|$. We use a standard technique for constructing a PCP with boolean queries. The verifier expects the proof to contain, instead of a value $f(\overline{v})$, the *Hadamard code* of the bit string $f(\overline{v})$. We can build an appropriate inner verifier that reads one bit from the (supposed) Hadamard code of $f(\overline{v}_i)$ for $i = 1, 2, \ldots, d+1$. The test of the outer verifier is replaced by a meaningful test on the $d+1$ bits read.

The inner verifier is analyzed using Fourier analysis. Recall that in Håstad's PCP, one needs to have the squared Fourier coefficient $\widehat{B}_j^2$, and this is achieved by reading two bits from block B_j. Our Fourier analysis also needs something similar. However, we cannot read more than one bit from any block (= Hadamard code); otherwise we lose the quasi-randomness property. We get around this problem by a clever trick. We modify the verifier as follows: As before, the verifier picks a random line ℓ and $d+1$ points on it $\overline{v}_1, \overline{v}_2, \ldots, \overline{v}_{d+1}$ and performs the test described. In addition, the verifier picks *another* set of $d+1$ random points on the line, say $\overline{u}_1, \overline{u}_2, \ldots, \overline{u}_{d+1}$. He verifies that the degree d polynomial reconstructed from values $f(\overline{u}_1), f(\overline{u}_2), \ldots, f(\overline{u}_{d+1})$ is identical to the polynomial reconstructed from $f(\overline{v}_1), f(\overline{v}_2), \ldots, f(\overline{v}_{d+1})$ (this polynomial actually happens to be of degree $d-1$). This is exactly the analogue of reading two bits from block B_j in Håstad's PCP. The additional queries look wasteful, but this is precisely how we introduce redundancy (= error correction) without losing quasi-randomness. Unfortunately, the exact significance of this trick will not be clear unless one looks at the actual Fourier analysis.

Remark 1.11. Definition 1.8 requires that the query pattern of the PCP verifier is different in the YES and NO cases. We would like to stress that the query pattern in the YES and NO cases is the same at the level of the (modified) outer verifier since in both the YES and NO cases, the verifier picks random points on a randomly selected line. However, quite magically, the difference in the query pattern in the YES and NO cases manifests itself at the inner verifier level.

1.7. Comparison with Feige's hypothesis. Feige [17] makes the following hypothesis about average-case hardness of 3SAT.

FEIGE'S HYPOTHESIS. *Let ϕ be a 3SAT formula with n variables and Cn clauses where every clause is picked at random from the set of all possible clauses. Then for an arbitrarily large constant C, there is no polynomial time algorithm that* (1) *says YES if ϕ is satisfiable and* (2) *says NO at least* 50% *of the times for a random formula ϕ.*

Assuming this hypothesis, Feige shows that graph min-bisection, dense k-subgraph, and bipartite clique have no PTAS (Alekhnovich [1] later shows these results assuming random instances of 3-linear-equations are hard).

Let us see how we are able to bypass Feige's hypothesis and still prove these inapproximability results. Consider the following PCP constructed from an instance ϕ of Gap-3SAT. The proof supposedly consists of a satisfying assignment to the 3SAT formula ϕ. The verifier picks one clause at random, queries the three variables in the clause, and checks if the clause is satisfied. Now what if the formula ϕ was a random instance of 3SAT? Then the query pattern of the verifier would be *truly random*. Thus Feige's hypothesis, in some sense, says that there exists a PCP whose query pattern is *truly random*.

On the other hand, quasi-randomness is only one specific property which is satisfied by a truly random query pattern. Applied to 3SAT, the quasi-randomness property says that for any set of half the variables, $(1 \pm o(1))\frac{1}{8}$ fraction of the clauses have all their three variables from this set. We make the following observation: In order to prove inapproximability results for graph theoretic problems, it suffices to have hard instances of 3SAT (or any constraint satisfaction problem (CSP) with constraints on a bounded number of variables) with a quasi-randomness property. One does not really need the 3SAT instances to be truly random, as in Feige's paper. We are able to construct such quasi-random PCPs, thereby obtaining the desired inapproximability results.

1.8. Overview of the paper. The main result in this paper is Theorem 1.9. The proof of this result is quite lengthy, involving a sequence of reductions (see Figure 1).

We introduce a new problem called the homogeneous algebraic CSP (HomAlgCSP). (Algebraic CSPs have been used before; see [24], for example. Our CSPs have the additional property that all constraints are homogeneous linear and we do not allow a trivial solution that is identically zero.) Our proof can be divided into two parts: First we reduce MDC to HomAlgCSP. This reduction appears in sections 3, 4, 5, and 6. This is perhaps the most interesting part of the paper. The modules used in this reduction, namely, the generalized sum-check protocol, the splitability certificate, and the subspace membership certificate, could be of independent interest.

Then we construct a PCP verifier from the HomAlgCSP instance. The verifier consists of an outer verifier (section 7 and 8) and an inner verifier (section 10). We actually need to modify the outer verifier before constructing the inner verifier. This modified outer verifier appears in section 9.

We recommend that in the first reading, the paper be read in the following way: (i) Browse through section 2, which gives an overview of techniques and the high level view of the reduction. Also take a look at Appendix A, which gives some of the facts/tools used. (ii) Read the beginning of section 3 and the statement of Theorem 3.4. Skip the rest of section 3 as well as sections 4, 5, and 6, which are devoted to proving Theorem 3.4. Though this is where the meat of the paper lies, it is rather lengthy and one might feel lost in the first reading. (iii) Read section 7 and Theorem 7.6. Skip section 8. (iv) Read sections 9 and 10.

2. Techniques and high level view of the reduction. In this section, we give an overview of techniques in the paper. This is a general overview that hides many subtleties involved in our reduction.

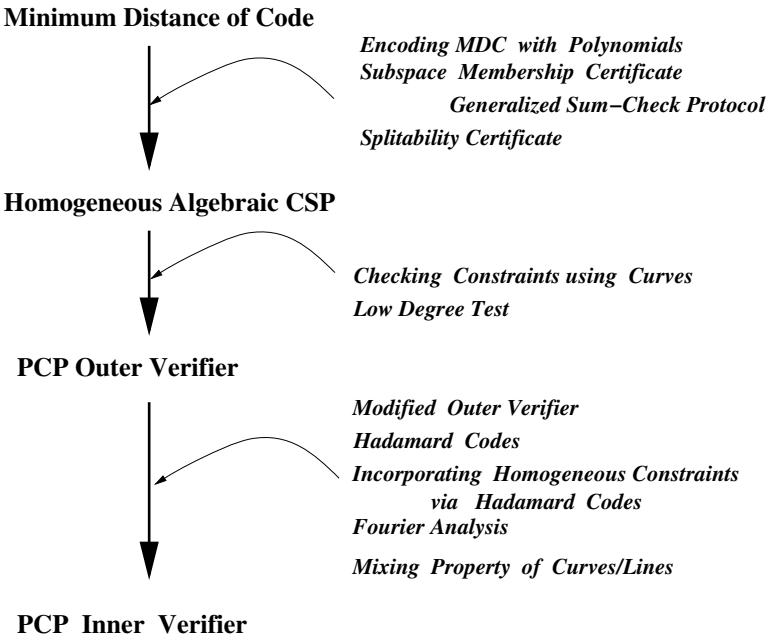


FIG. 1. *Overview of full reduction.*

2.1. MDC to homogeneous algebraic CSP. HomAlgCSP is the following problem (see Definition 3.1): Given space $\mathbb{F}^m$, the goal is to find a nonzero low degree m-variate polynomial f that satisfies a maximum number of constraints. Every constraint is a homogeneous linear constraint on $O(1)$ values of f at specific points in $\mathbb{F}^m$. As explained in Remark 3.2, HomAlgCSP is a very similar problem to MDC. A reduction from MDC to HomAlgCSP transforms (see Theorem 3.4) the *global* constraints of MDC to *local* constraints of HomAlgCSP that depend only on $O(1)$ values of f at specific points. This global-to-local transformation is a major step in our reduction (as must be the case with any PCP).

The reduction from MDC to HomAlgCSP works as follows: It is easy to encode MDC as an instance of the following problem: Given a subspace of low degree polynomials $\mathcal{B}$ and a set of points $\mathcal{P} \subseteq \mathbb{F}^m$, find a nonzero polynomial $f \in \mathcal{B}$ that vanishes at a maximum number of points in $\mathcal{P}$. Note that we now have local constraints, one for every point $\overline{p} \in \mathcal{P}$, saying that f vanishes at $\overline{p}$. However, the main trouble is, how do we enforce the condition that $f \in \mathcal{B}$?

We do this by constructing what we call a *subspace membership certificate*. It is a low degree polynomial g that supposedly certifies that $f \in \mathcal{B}$. The polynomial g needs to be of a certain type and this is certified by constructing another polynomial h which we call the *splitability certificate*. We add a collection of local constraints on values of f, g, and h that enforce the consistency between the three polynomials. The polynomials g and h are constructed as follows.

We observe that $f \in \mathcal{B}$ if and only if the value of f at any point $\mathbf{q} \in \mathbb{F}^m$ is a *weighted sum* of its values at some well-chosen points. Thus we can check whether this *sum-check* holds for a random point $\mathbf{q}$. We employ a generalization of the sum-check protocol in order to carry out the sum-check. The usual sum-check protocol sums over values of f over the boolean cube $\{0,1\}^m$, whereas our protocol sums over

an arbitrary subset of the boolean cube. We choose the subset to consist of only low Hamming-weight points of the boolean cube so that the *sum-check polynomials* remain of low degree.

The polynomial g (i.e., the subspace membership certificate) is a linear combination of all the sum-check polynomials. For the sum-check protocol to work, the sum-check polynomials need to be independent in some sense. The polynomial h (i.e., the splitability certificate) certifies that this is indeed so. The polynomials g and h together form a certificate that $f \in \mathcal{B}$. Finally, we *club together* the three polynomials f, g, and h into a single polynomial ϕ that serves as an instance of HomAlgCSP. The constraints are all homogeneous linear on $O(1)$ values of ϕ at specific points.

Let us illustrate the power of subspace membership certificate construction by an example. Let us say we have a low degree m-variate polynomial f that is given as a table of values. We want to verify that f is multilinear (i.e., f has no monomial that looks like $x_1^2 x_2 x_3$). How many queries do we need to the table of values of f (and possibly to an additional proof)?

The straightforward way is to pick a random point in $\mathbb{F}^m$ and check that on *every* axis-parallel line passing through this point, f behaves as a linear function (this is precisely what [22] does). However, this needs $O(m)$ queries. On the other hand, our construction gives a query-efficient way of certifying that f is multilinear. The certificate consists of a pair of new low degree polynomials (g, h), and we need only $O(1)$ queries to tables of values of f, g, and h! Moreover, the test is homogeneous linear. The fact that the certificate itself is a polynomial is crucial for our purpose.

2.2. HomAlgCSP to outer verifier. We construct a PCP outer verifier from the HomAlgCSP instance. Let f be the polynomial required in the HomAlgCSP instance. The outer verifier expects as a proof a table of values of f. The verifier needs to verify the following: (1) f is indeed a nonzero low degree polynomial, and (2) values of f satisfy constraints of HomAlgCSP.

It is well known how to handle the first task, namely, verifying that f is a degree d polynomial. The so-called low degree test (LDT) expects an additional proof: a table that gives for every line in $\mathbb{F}^m$ a univariate polynomial that is supposedly the restriction of f to the line. The verifier picks a random line and a point on the line, and checks consistency between the table of points and table of lines. For our purposes, however, the verifier has no access to the *lines table*. The verifier, instead, reads $d+1$ values of f on a line ℓ and reconstructs by himself the restriction of f to line ℓ. In this paper, the degree d of polynomials is a constant, and therefore the verifier still makes only a constant number of queries. We need the strong analysis of the LDT by Arora and Sudan [5].

The next task of the verifier is to verify that f satisfies certain constraints. The verifier picks one constraint at random that depends on k values of f at specific points, say $\{\overline{p}_i\}_{i=1}^k \in \mathbb{F}^m$. In principle, the verifier could just read off these values and verify the constraint. However, it is well known in PCP literature that one needs to have some form of *error correction* and infer the values of f at points $\{\overline{p}_i\}_{i=1}^k$ from the global behavior of f. We use another well-known primitive to handle this. We pick a random *curve* L that passes through the points $\{\overline{p}_i\}_{i=1}^k$. Restriction of f to the curve L is again a low degree univariate polynomial that we reconstruct by reading sufficiently many values of f on the curve. After reconstructing this *curve polynomial*, we get values of f at points $\{\overline{p}_i\}_{i=1}^k$ and verify that they indeed satisfy the constraint.

We in fact combine the two tasks of the verifier. We pick a random pair of a curve and a line that intersect at a point. The verification procedures described above are

carried out for both the line and the curve. We in addition check that the line and the curve are consistent at the intersection point. This completes the description of the outer verifier.

Now let us see what exactly the outer verifier achieves. Let us restrict ourselves to the first task (i.e., the variant of line-point LDT where the line polynomial is reconstructed by reading several values on the line). Recall that our eventual goal is to build a quasi-random PCP. The definition of the quasi-random PCP requires that if Π_* is any subset of the proof of half the size, and the number of queries is q, then the probability that all queries fall in Π_* is essentially $\frac{1}{2^q}$.

Let us illustrate that the outer verifier achieves the quasi-randomness property. Let Π_* be any set of half the points from $\mathbb{F}^m$. It is easy to show that lines in $\mathbb{F}^m$ have excellent *mixing property*, i.e., for any such set Π_*, for almost every line ℓ, the fraction of points on ℓ belonging to Π_* is very close to $\frac{1}{2}$. Now recall that the outer verifier picks a random line and q random points on the line. Thus the probability that all his queries fall into Π_* is essentially $\frac{1}{2^q}$. A similar mixing property holds for curves as well (see Appendix A.4), and therefore the outer verifier is quasi-random.

2.3. Outer verifier to modified outer verifier. The purpose of the modified outer verifier is exactly as explained in sections 1.5 and 1.6. It is the analogue of reading two bits from block B_j in Håstad's 3-query PCP. The modified outer verifier introduces redundancy (= error correction), and this enables us to bound the Fourier terms in the analysis of the inner verifier. This works in exactly the same way that the squared Fourier coefficient $\widehat{B}_j^2$ enables the analysis of Håstad's PCP.

Recall that the outer verifier reads values of f on $d+1$ points from a line and a curve and reconstructs the line and curve polynomials. The modified outer verifier reads values of f on additional $d+1$ points from the line and the curve and checks that he again gets the *same* line and curve polynomials. Thus one introduces redundancy, but without losing quasi-randomness.

2.4. Modified outer verifier to inner verifier. The final step is to build the inner verifier and prove the quasi-randomness property. Note that the proof for the modified outer verifier is a table of values of a polynomial $f : \mathbb{F}^m \mapsto \mathbb{F}$. We work only with fields of characteristic 2. Therefore denoting $|\mathbb{F}| = 2^l$, we can think of elements of $\mathbb{F}$ as l-bit strings. The inner verifier expects as a proof the Hadamard codes of these l-bit strings. Let us say the modified outer verifier makes q queries. Then the inner verifier also makes q queries, reading one bit each from the corresponding Hadamard code. As pointed out earlier, it is crucial that one reads only one bit from each code/block; otherwise the quasi-randomness property fails.

The quasi-randomness of the inner verifier is proved using Fourier analysis. We prove the desired two properties: (1) In the completeness case, let Π_0 be the set of 0-bits in the proof. This consists of half the bits in the proof. The probability that all q queries fall in Π_0 is essentially $\frac{1}{2^{q-1}}$. (2) In the soundness case, we assume on the contrary that there is a set Π_* of half the bits in the proof such that with probability $\frac{1}{2^q} \pm \delta$, all queries fall in Π_*. Then we show that one can *decode* the proof and construct a table $f : \mathbb{F}^m \mapsto \mathbb{F}$ that the outer verifier accepts with probability $\delta^{O(1)}$. This is a contradiction if the outer verifier is chosen to have sufficiently small soundness. This is ensured by choosing the MDC instance (and therefore the HomAlgCSP instance) with small soundness. The decoding is the standard probabilistic decoding in PCP literature. A supposed Hadamard code is decoded to α with probability $\widehat{A}_\alpha^2$, i.e., proportional to the square of the corresponding Fourier coefficient. We use several

nice properties of Hadamard codes that we list below (the popular long codes do not work; the Hadamard code–based inner verifier was used earlier in [29]):

1. The modified outer verifier reads q values of f and performs homogeneous linear tests. There is a natural way of accessing Hadamard codes so that only those Fourier coefficients survive which satisfy all the homogeneous linear constraints (this is where we need homogeneity). Thus, as far as the inner verifier is concerned, this testing comes for free.
2. If we have access to Hadamard codes of x and y, then we essentially have access to the Hadamard code of the concatenated string $x \circ y$. The process of accessing a bit from the Hadamard code of $x \circ y$ can be simulated by reading one bit each from codes of x and y.
3. The zero Fourier coefficient of a supposed Hadamard code equals the fraction of 1's in the supposed code.

There is a nice interpretation of the Fourier coefficients in the analysis. The zero coefficient corresponds to the term $\frac{1}{2^q}$ in the soundness probability. The nonzero coefficients are used to *decode* the proof and define a strategy for the outer verifier. This in turn gives a solution to the HomAlgCSP (and therefore the MDC) instance. Recall that the HomAlgCSP instance demands a *nonzero* polynomial f.

As mentioned in Remark 1.11, the difference in the query pattern of the verifier in the YES and NO cases is *created* at the inner verifier level. At the outer verifier level, the query pattern is the same, irrespective of the YES or NO case (in both cases, random points are chosen on a randomly selected line/curve).

Remark 2.1. Our reduction was discovered by way of *reverse engineering*. We want to use the combination of interpolation and Hadamard codes to get a boolean PCP that does not make two queries in one block (as explained in sections 1.5 and 1.6). For this, it turns out that the outer PCP test must be linear and homogeneous. This is why we start with MDC, which involves a set of homogeneous constraints, and we transform them into *local* homogeneous constraints.

Another remark on the size of the construction: we want to have a constant query PCP, and hence the polynomials must have constant degree (when restricted to a line). This forces us to work with huge numbers of dimensions (i.e., n^{ϵ}) and the size of the construction becomes subexponential.

3. MDC to HomAlgCSP: Encoding MDC with polynomials. Now we begin our construction of the quasi-random PCP. The first step is defining the HomAlgCSP problem.

DEFINITION 3.1. *Let the HomAlgCSP $\mathcal{A}(f, k, d, m, \mathbb{F}, \mathcal{C})$ be the following problem (think of k as a fixed integer like* 21*, and d as a large constant integer):*

1. *f is (supposed to be) an m-variate degree d polynomial over field $\mathbb{F}$.*
2. *$\mathcal{C}$ is a system of constraints where every constraint is on k values of the polynomial f at certain points. This constraint is a conjunction of homogeneous linear constraints. Thus a typical constraint looks like*

$$\sum_{i=1}^{k} \gamma_{ij} f(\overline{p_i}) = 0 \quad \text{for } j = 1, 2, \ldots, \qquad \text{where } \overline{p}_i \in \mathbb{F}^m \text{ and } \gamma_{ij} \in \mathbb{F}.$$

   A constraint $C \in \mathcal{C}$ will be denoted by $C(\{\overline{p}_i\}_{i=1}^{k})$, thus indicating only the points on which the constraint is defined. The actual homogeneous constraints will be implicit.
3. *$\mathcal{C}$ has $|\mathbb{F}|^{O(m)}$ constraints.*

The goal is to find a polynomial f, not identically zero, so as to maximize the fraction of constraints satisfied. Let $OPT(\mathcal{A})$ denote the optimum.

Remark 3.2. HomAlgCSP and MDC are similar problems. In MDC, we want to find a nonzero assignment to unknown variables that minimizes the number of unsatisfied homogeneous linear constraints. HomAlgCSP is essentially the same problem if we view the coefficients of the polynomial f as unknown variables (one minor difference is that HomAlgCSP is a maximization problem and we allow a constraint to be an arbitrary conjunction of homogeneous linear constraints).

However, the main difference is the following: for MDC, every constraint could depend on an arbitrary number of variables, whereas for HomAlgCSP, every constraint has a compact/local representation in terms of values of f at a constant number of points. Thus a reduction from MDC to HomAlgCSP transforms global constraints to local ones. This is what PCPs are all about!

Remark 3.3. It may help to look at the HomAlgCSP problem from a PCP viewpoint. A PCP would be the table of values of the (unknown) polynomial f. The verifier would pick a random constraint $C(\{\overline{p_i}\}_{i=1}^k) \in \mathcal{C}$, read the values $\{f(\overline{p_i})\}_{i=1}^k$, and check whether the constraint is satisfied. Thus, the HomAlgCSP problem is equivalent to finding a proof that maximizes the acceptance probability of the verifier.

The following theorem states the main result in this and the next three sections. It gives a gap-preserving reduction from MDC to HomAlgCSP.

THEOREM 3.4. *Let $\mathbf{A}$ be an instance of MDC with size $N \times n'$ over field $\mathbb{F}$. Assume that $N \le |\mathbb{F}| \le N^2$. Let $\epsilon > 0$ be an arbitrarily small constant, $m = N^\epsilon$, and let $d = \frac{100}{\epsilon}$ be an integer. There is a reduction from $\mathbf{A}$ to an instance of HomAlgCSP $\mathcal{A}(f, k = 21, d, m, \mathbb{F}, \mathcal{C})$ such that*

$$1 - OPT(\mathbf{A}) \;\le\; OPT(\mathcal{A}) \;\le\; \max\left\{1 - OPT(\mathbf{A}), O\left(\frac{d}{|\mathbb{F}|}\right)\right\}. \tag{1}$$

We begin the proof of Theorem 3.4. The first step is to encode MDC as the following problem: Given a set of points $\mathcal{P}$ in $\mathbb{F}^m$, find a nonzero degree d polynomial f that is nonzero at minimum number of points from $\mathcal{P}$. In addition, f is required to be in a certain subspace of polynomials.

The next step is to define two more polynomials g and h that are supposed to serve as a *certificate* that f indeed belongs to the required subspace of polynomials. Then we define a collection of constraints all of which are homogeneous linear constraints on k' values of f, g, and h. These constraints enforce the consistency between f, g, and h as well as the condition that f takes nonzero values at a minimum number of points from $\mathcal{P}$. In the end, we will *club together* the three polynomials f, g, and h into a single polynomial ϕ and all the constraints are homogeneous constraints on k values of ϕ. This would prove Theorem 3.4.

Remark 3.5. Actually Theorem 3.4 holds for a more general setting of parameters. For any choice of integers m and d such that $\binom{m}{d} \ge N$, the reduction produces an instance of HomAlgCSP $\mathcal{A}(f, k = 21, d' = 10d, m' = O(m^3 d), \mathbb{F}, \mathcal{C})$ such that (1) holds. However, we will stick to the setting of the parameters in Theorem 3.4 used in this paper.

3.1. Encoding MDC with polynomials. Let $\mathbf{A} = \{a_{ji}\}$ be an $N \times n'$ instance of MDC. We assume without loss of generality (w.l.o.g.) that the first n' rows of $\mathbf{A}$ form an identity matrix, i.e., $a_{ji} = 0$ if $1 \le i \ne j \le n'$ and $a_{ii} = 1$ for $1 \le i \le n'$. This can be achieved by rearranging rows so that the first n' rows are linearly

independent and then multiplying $\mathbf{A}$ by a suitable $n' \times n'$ matrix. This transformation doesn't change $OPT(\mathbf{A})$.

Remark 3.6. In the following, the polynomial f and the parameters ϵ, m, d are related to but *not* the same as those in Theorem 3.4. They keep changing somewhat over the course of the reduction.

Fix an arbitrarily small constant $\epsilon > 0$ and let $m = N^{\epsilon}$. Choose an integer d such that $N \leq \binom{m}{d}$. Thus $d = \frac{2}{\epsilon}$ suffices. Let $\mathcal{F}_0$ be a family of subsets of $[m]$ of size d such that $|\mathcal{F}_0| = N$. Let $\mathcal{F} \subseteq \mathcal{F}_0$ be a subfamily with $|\mathcal{F}| = n'$. The columns of the matrix $\mathbf{A}$ are indexed by $I \in \mathcal{F}$ and the rows are indexed by $J \in \mathcal{F}_0$, where the first n' rows are indexed by $I \in \mathcal{F}$.

For $J \in \mathcal{F}_0$, let $\overline{p}_J \in \mathbb{F}^m$ be the point with $\{0,1\}$-coordinates that is the characteristic vector of set J. Let $\mathbf{x}_J$ be the degree d monomial $\prod_{j \in J} x_j$. Let $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ denote a vector of formal variables. The important property is that the monomial $\mathbf{x}_J$ evaluates to 1 at point $\overline{p}_J$ and evaluates to 0 at any point $\overline{p}_{J'}$ for $J' \neq J, |J'| = d$.

We will encode the columns of matrix $\mathbf{A}$ by polynomials in $\mathbb{F}[\mathbf{x}]$. For column I, define a degree d polynomial $f_I \in \mathbb{F}[\mathbf{x}]$ as

$$f_I := \sum_{J \in \mathcal{F}_0} a_{JI} \mathbf{x}_J \qquad (a_{JI} = a_{ji}).$$

It is clear that $f_I(\overline{p}_J) = a_{JI}$ for $J \in \mathcal{F}_0, I \in \mathcal{F}$. In other words, the coordinates of the column indexed by I are given by $f_I(\overline{p}_J)\ \forall\ J \in \mathcal{F}_0$.

Let $span\{f_I | I \in \mathcal{F}\}$ denote the linear span of polynomials f_I's. Thus the MDC problem can be reformulated as follows.

MDC (reformulation). Find a nonzero polynomial $f \in span\{f_I | I \in \mathcal{F}\}$ so as to minimize the fraction of nonzero entries in the vector

$$\{\ f(\overline{p}_J)\ | J \in \mathcal{F}_0\ \}.$$

How do we ensure that f belongs to the span of polynomials $\{f_I | I \in \mathcal{F}\}$? The next few sections are devoted to the construction of two polynomials g and h that serve as a certificate that f indeed belongs to the span.

Overview of the construction of certificate polynomials. We make a simple but important observation (Lemma 3.7): f belongs to the span of $\{f_I | I \in \mathcal{F}\}$ if and only if for all points $\mathbf{q} \in \mathbb{F}^m$, the value $f(\mathbf{q})$ is a weighted sum of the values $\{f(\overline{p}_I) | I \in \mathcal{F}\}$. So we check whether this property holds for a randomly selected point $\mathbf{q}$. The checking procedure could be carried out by reading the values $f(\mathbf{q})$ and $\{f(\overline{p}_I) | I \in \mathcal{F}\}$. This, however, would take too many queries, whereas we can afford only a constant number of queries. The idea is to use the sum-check protocol used in the proof of the PCP theorem. It turns out that the protocol needs values of linear combinations of *sum-check polynomials*. The first *certificate polynomial* g is the Hadamard code over the sum-check polynomials. However, for the sum-check protocol to work, the sum-check polynomials need to be *independent* of each other in some sense. The second *certificate polynomial* h serves as a proof of this independence. We define the notion of a *splitable polynomial*, and h is actually a certificate that the polynomial g is splitable. We call g and h the *subspace membership certificate* and *splitability certificate*, respectively.

3.2. Basic observation for constructing the subspace membership certificate. Let $f(\mathbf{x}) \in \mathbb{F}[\mathbf{x}]$ be the desired polynomial. Let $\mathbf{q}$ be a vector of formal

variables. Consider the polynomial

$$\psi(\mathbf{q}, \mathbf{x}) := \sum_{I \in \mathcal{F}} f_I(\mathbf{q}) \mathbf{x}_I. \tag{2}$$

Note that f_I's are of degree d; therefore ψ is of degree $2d$. We make the following important observation.

LEMMA 3.7. *$f \in span\{f_I | I \in \mathcal{F}\}$ if and only if this formal identity holds:*

$$f(\mathbf{q}) = \sum_{I \in \mathcal{F}} \psi(\mathbf{q}, \overline{p}_I) f(\overline{p}_I). \tag{3}$$

Proof. Assume that $f = \sum_{I' \in \mathcal{F}} \lambda_{I'} f_{I'}$, where $\lambda_{I'} \in \mathbb{F}$. Then

$$\begin{aligned}
\sum_{I \in \mathcal{F}} \psi(\mathbf{q}, \overline{p}_I) f(\overline{p}_I) &= \sum_{I \in \mathcal{F}} \psi(\mathbf{q}, \overline{p}_I) \sum_{I' \in \mathcal{F}} \lambda_{I'} f_{I'}(\overline{p}_I) \\
&= \sum_{I \in \mathcal{F}} \psi(\mathbf{q}, \overline{p}_I) \lambda_I \qquad f_{I'}(\overline{p}_I) = a_{II'} = 0 \text{ if } I \neq I' \text{ and 1 otherwise} \\
&= \sum_{I \in \mathcal{F}} \lambda_I \sum_{I'' \in \mathcal{F}} f_{I''}(\mathbf{q}) \cdot \mathbf{x}_{I''}(\overline{p}_I) \\
&= \sum_{I \in \mathcal{F}} \lambda_I f_I(\mathbf{q}) \\
&= f(\mathbf{q}),
\end{aligned}$$

as desired. For the other direction, note that if $f(\mathbf{q}) = \sum_{I \in \mathcal{F}} \psi(\mathbf{q}, \overline{p}_I) f(\overline{p}_I)$, then we have

$$\begin{aligned}
f(\mathbf{q}) &= \sum_{I \in \mathcal{F}} f(\overline{p}_I) \sum_{I' \in \mathcal{F}} f_{I'}(\mathbf{q}) \mathbf{x}_{I'}(\overline{p}_I) \\
&= \sum_{I \in \mathcal{F}} f(\overline{p}_I) f_I(\mathbf{q}),
\end{aligned}$$

as desired. ∎

Thus, one can probabilistically check whether f belongs to $span\{f_I | I \in \mathcal{F}\}$ by checking whether (3) holds for a random $\mathbf{q} \in \mathbb{F}^m$. Note that verifying (3) would require accessing all the values $f(\overline{p}_I)$. However, we want to read only a constant number of values of f. The idea is to use the sum-check protocol used in the proof of the PCP theorem. This protocol lets us verify that the sum of values of a function over $\{0,1\}^m$ equals some target value. We need a generalization of this protocol where the sum is taken over the restricted set of points $\{\overline{p}_I | I \in \mathcal{F}\}$. An important feature of this generalized protocol is that the points $\{\overline{p}_I | I \in \mathcal{F}\}$ are low Hamming-weight points (i.e., only d nonzero coordinates). This ensures that all the sum-check polynomials are of degree $O(d)$.

Before we get into the sum-check protocol, let us introduce the notion of *splitability* of polynomials and how to construct a certificate for splitability. This construction will be a useful primitive later and will introduce some of the new ideas. Admittedly, the reader would have no clue at this point how/why it is useful, but we think the construction is also of independent interest.

4. Splitability certificate. For $1 \leq j \leq M$, let $\mathbf{x}^{(j)}$ denote a vector of m_j formal variables where the sets of variables for different j are disjoint. Let $\mathbf{x} = \cup_{j=1}^{M} \mathbf{x}^{(j)}$.

DEFINITION 4.1. *A degree d polynomial $g \in \mathbb{F}[\mathbf{x}]$ is called splitable if it can be written as*

$$g(\mathbf{x}) := \sum_{j=1}^{M} g_j(\mathbf{x}^{(j)}), \tag{4}$$

where $g_j \in \mathbb{F}[\mathbf{x}^{(j)}]$ are degree d polynomials.

In other words, splitability means that the variables in different sets $\mathbf{x}^{(j)}$ do not *interfere* with each other. In this section, we show how to *certify* that a given polynomial g is splitable. The certificate consists of another polynomial h. The certificate can be verified w.h.p. with a constant number of accesses given tables of values of g and h.

THEOREM 4.2. *Let $\mathbf{x} = \cup_{j=1}^{M} \mathbf{x}^{(j)}$ be a partition of variables and let $g \in \mathbb{F}[\mathbf{x}]$ be a polynomial of degree d. Assume that every $\mathbf{x}^{(j)}$ has at most m variables. Let $h \in \mathbb{F}[\mathbf{z}]$ be a degree $2d$ polynomial that is supposed to be a certificate that g is splitable. The number of variables in $\mathbf{z}$ is bounded by $O(m^2M + dM)$. There is a verifier with the following properties:*

1. *The verifier accesses one value of g and four values of h.*
2. *If g is splitable, then there exists a unique polynomial h such that the verifier accepts with probability 1, and for any other h, the verifier accepts with probability at most $O(\frac{d}{|\mathbb{F}|})$. If $g \equiv 0$, then the corresponding $h \equiv 0$.*
3. *If g is not splitable, then no matter what h is (of degree $2d$), the verifier accepts with probability at most $O(\frac{d}{|\mathbb{F}|})$.*

4.1. Basic idea. Let us explain the basic idea behind our construction with a small example. Let us say we have a polynomial $g(x_1, x_2, \ldots, x_m; w_1, w_2, \ldots, w_m)$ and we want to certify that g is splitable, i.e., g can be written as the sum of two polynomials, one in $x_1, \ldots, x_m$ and the other in $w_1, \ldots, w_m$. Assume for the moment that g is guaranteed to be multilinear.

Here is our first attempt. We check for every pair (i_0, j_0) that g has no monomial involving the product $x_{i_0} w_{j_0}$. This can be done as follows. Substitute random values for all variables except x_{i_0}, w_{j_0} and check that the resulting (restricted) function is linear in x_{i_0} and w_{j_0}. More precisely, pick random values for variables $x_i, w_j, i \neq i_0, j \neq j_0$. Pick two sets of random values x_{i_0}, w_{j_0} and x'_{i_0}, w'_{j_0}. Pick a random value ζ and check that

$$\begin{aligned} \zeta \cdot g(\ldots, x_{i_0}, \ldots; \ldots, w_{j_0}, \ldots) + (1-\zeta) \cdot g(\ldots, x'_{i_0}, \ldots; \ldots, w'_{j_0}, \ldots) \\ = g(\ldots, \zeta x_{i_0} + (1-\zeta) x'_{i_0}, \ldots; \ldots, \zeta w_{j_0} + (1-\zeta) w'_{j_0}, \ldots), \end{aligned}$$

where the dots indicate the same set of values for $x_i, y_j, i \neq i_0, j \neq j_0$. Note that we read only 3 values of g and do not need any additional certificate. Clearly, if g is multilinear and splitable, the test always accepts. If g has a monomial involving the product $x_{i_0} w_{j_0}$, the test accepts with probability at most $O(\frac{d}{|\mathbb{F}|})$.

Now the trouble is that we have to do this for *every* pair i_0, j_0, which we cannot afford to do (we want to keep the number of queries constant). To overcome this

trouble, we use the following trick. Make the following substitution:

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} a_{11} & \dots & a_{1m} \\ & \dots & \\ & \vdots & \\ a_{m1} & \dots & a_{mm} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix};$$

(5)

$$\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} = \begin{bmatrix} b_{11} & \dots & b_{1m} \\ & \dots & \\ & \vdots & \\ b_{m1} & \dots & b_{mm} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{bmatrix}.$$

Here a_{ij}, b_{ij}, u_i, v_j are new formal variables. After substituting values for x_i, y_j in the polynomial g, we get a new polynomial h in a_{ij}, b_{ij}, u_i, v_j. Here is a simple observation: g is splitable if and only if h has no monomial involving the product $u_1 v_1$.

The polynomial h will be the certificate. Using the checking procedure described earlier, we can check that h has no monomial involving $u_1 v_1$. Also, we can verify that g and h are consistent by checking that they agree on points which are related via the transformation (5).

This is the basic idea behind the splitability certificate. An additional twist is that g is not necessarily multilinear, but this possibility is handled easily as well.

4.2. Defining the splitability certificate. Now we prove Theorem 4.2. For $1 \le j \le M$, let m_j be the number of variables in $\mathbf{x}^{(j)}$. Let $\mathbf{y}^{(j)}$ denote a vector of $m_j - 1$ variables and let Y_j denote a single special variable. Let $\mathbf{T}^{(j)}$ denote a matrix of m_j^2 variables. The intended meaning is that $\mathbf{x}^{(j)} = \mathbf{T}^{(j)}(\mathbf{y}^{(j)} \circ Y_j)$. To be precise, we intend that

$$\mathbf{x}_i^{(j)} = \sum_{s=1}^{m_j - 1} \mathbf{T}_{is}^{(j)} \mathbf{y}_s^{(j)} + \mathbf{T}_{im_j}^{(j)} Y_j \quad \text{for } 1 \le i \le m_j. \tag{6}$$

Let $\mathbf{T} = \cup_{j=1}^{M} \mathbf{T}^{(j)}$, $\mathbf{y} = \cup_{j=1}^{M} \mathbf{y}^{(j)}$, and $\mathbf{Y} = \cup_{j=1}^{M} \{Y_j\}$. Substituting the values of $\mathbf{x}^{(j)}$ into (4), we get

$$g = \sum_{j=1}^{M} h_j(\mathbf{T}^{(j)}, \mathbf{y}^{(j)}, Y_j),$$

where each h_j has degree $2d$. We write h_j in increasing powers of Y_j so that

$$h_j := h_{j0}(\mathbf{T}^{(j)}, \mathbf{y}^{(j)}) + \sum_{l=1}^{2d} h_{jl}(\mathbf{T}^{(j)}, \mathbf{y}^{(j)}) Y_j^l.$$

Therefore

$$g = \sum_{j=1}^{M} h_{j0}(\mathbf{T}^{(j)}, \mathbf{y}^{(j)}) + \sum_{j=1}^{M} \sum_{l=1}^{2d} h_{jl}(\mathbf{T}^{(j)}, \mathbf{y}^{(j)}) Y_j^l.$$

The *splitability certificate* is obtained by replacing powers of Y_j by new variables. To be precise, for every $1 \le j \le M$, $1 \le l \le 2d$, let U_{jl} be a single formal variable and

let $\mathbf{U}$ denote the vector of all such variables. The certificate is the polynomial

$$h(\mathbf{T},\mathbf{y},\mathbf{U}) := \sum_{j=1}^{M} h_{j0}(\mathbf{T}^{(j)},\mathbf{y}^{(j)}) + \sum_{j=1}^{M}\sum_{l=1}^{2d} h_{jl}(\mathbf{T}^{(j)},\mathbf{y}^{(j)})U_{jl}.$$

4.3. Checking the splitability certificate. The checking procedure consists of two parts: first we check that the polynomial h is a linear function of variables $\mathbf{U}$. Then we check that g and h are consistent (this automatically checks that g is indeed splitable).

Checking linearity. Pick values for $\mathbf{T}^{(j)},\mathbf{y}^{(j)}$ at random. Let $\mathbf{U}$, $\mathbf{U}'$ be two sets of values for variables in $\mathbf{U}$ picked at random. Pick a scaler/field value ζ at random. Accept if and only if

$$h(\mathbf{T},\mathbf{y},\zeta\mathbf{U}+(1-\zeta)\mathbf{U}') = \zeta\cdot h(\mathbf{T},\mathbf{y},\mathbf{U}) + (1-\zeta)\cdot h(\mathbf{T},\mathbf{y},\mathbf{U}').$$

Clearly, the test reads 3 values of h. It accepts with probability 1 if h is indeed linear in $\mathbf{U}$ and accepts with negligible probability (i.e., at most $O(\frac{d}{|\mathbb{F}|})$) otherwise.

Checking consistency. Pick values for $\mathbf{T}^{(j)},\mathbf{y}^{(j)}$ and variables Y_j at random. Define $\mathbf{x}^{(j)} = \mathbf{T}^{(j)}(\mathbf{y}^{(j)}\circ Y_j)$. Let $U_{jl} = Y_j^l$. Accept if and only if

$$g(\mathbf{x}) = h(\mathbf{T},\mathbf{y},\mathbf{U}).$$

It is clear that if g is splitable, and h is defined appropriately, then this test accepts with probability 1. So the main task is proving soundness. Assume that g is not splitable. We want to show that the test accepts with negligible probability, no matter what certificate h the adversary gives. Let $\tilde{h}$ be the polynomial given by the adversary. We can assume that it is linear in $\mathbf{U}$ and thus looks like

$$\tilde{h}(\mathbf{T},\mathbf{y},\mathbf{U}) = \tilde{h}_0(\mathbf{T},\mathbf{y}) + \sum_{j=1}^{M}\sum_{l=1}^{2d}\tilde{h}_{jl}(\mathbf{T},\mathbf{y})U_{jl}.$$

For the given polynomial $g[\mathbf{x}]$, let $\tilde{g}(\mathbf{T},\mathbf{y},\mathbf{Y})$ be the polynomial obtained by substituting $\mathbf{x}^{(j)} = \mathbf{T}^{(j)}(\mathbf{y}^{(j)}\circ Y_j)$. The test accepts with negligible probability unless we have a formal identity

$$\tilde{g}(\mathbf{T},\mathbf{y},\mathbf{Y}) = \tilde{h}_0(\mathbf{T},\mathbf{y}) + \sum_{j=1}^{M}\sum_{l=1}^{2d}\tilde{h}_{jl}(\mathbf{T},\mathbf{y})Y_j^l.$$

Now the crucial point is that on the right-hand side (RHS), the powers Y_j^l do not *interfere* with each other. In other words, there is no monomial that involves the product $Y_jY_{j'}$ where $j\neq j'$. On the other hand, if $g[\mathbf{x}]$ is not splitable, then it will have a monomial that involves the product $\mathbf{x}_i^{(j)}\mathbf{x}_{i'}^{(j')}$ for $j\neq j'$. This will give a monomial involving the product $Y_jY_{j'}$ on the left-hand side (LHS), a contradiction!

5. Generalized sum-check protocol. In this section, we describe the generalized sum-check (GSC) protocol. This protocol along with the construction of the splitability certificate (in the previous section) is used to construct the subspace membership certificate in the next section.

The protocol is much more complicated than the usual sum-check protocol that allows us to verify that the sum of values of a polynomial f over the cube $\{0,1\}^m$ equals a target value. We provide the description of the usual sum-check protocol in Appendix E. We recommend reading it before reading the generalized protocol.

Our protocol needs to work for a sum over an arbitrary set of weight-d inputs (inputs in $\{0,1\}^m$ with exactly d coordinates equal to 1). The protocol could be of independent interest.

$GSC(a, \mathbb{F}, \tilde{f}, m, D, d, \mathcal{F}, \{g_{b,r} | b \in \{0,1\}, 0 \le r \le m-1\})$ consists of the following:

1. $\mathbb{F}$ is a field and $a \in \mathbb{F}$ is a *target* value.
2. $\tilde{f} \in \mathbb{F}[\mathbf{x}]$ is a degree D polynomial where $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ is a vector of m formal variables. We are given $\tilde{f}$ as a table of values.
3. $\mathcal{F} \subseteq \binom{m}{d}$, i.e., $\mathcal{F}$ is a family of size d subsets of $[m]$.
4. Every $g_{b,r}$ is a polynomial in $\mathbb{F}[\boldsymbol{\alpha}, \boldsymbol{\theta}, X]$ where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$, $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_m)$ are vectors of formal variables and X is a single formal variable. The degree of $g_{b,r}$ is $D + d$ and $g_{b,r}$ is guaranteed to depend only on $\alpha_1, \alpha_2, \ldots, \alpha_r$, $\theta_1, \ldots, \theta_r, X$. The polynomials $g_{b,r}$ are given as table of values.

The goal is to verify that

$$a = \sum_{I \in \mathcal{F}} \tilde{f}(\overline{p}_I) \tag{7}$$

by reading a constant number of values of $\tilde{f}$ and of each of the polynomials $g_{b,r}$. The *sum-check polynomials* $g_{b,r}$ are supposed to assist us in verifying that the summation is true. We have the following theorem.

THEOREM 5.1. *There is a protocol (verifier) for*

$$GSC(a, \mathbb{F}, \tilde{f}, m, D, d, \mathcal{F}, \{g_{b,r} | b \in \{0,1\}, 0 \le r \le m-1\})$$

satisfying these properties:

1. *The protocol picks $\boldsymbol{\alpha} \in \mathbb{F}^m$, $\boldsymbol{\theta} \in \mathbb{F}^m$ at random.*
2. *The protocol reads one value of $\tilde{f}$, namely, $\tilde{f}(\boldsymbol{\theta})$. For every $b \in \{0,1\}, 0 \le r \le m-1$, the protocol reads two values of $g_{b,r}$, namely, $g_{b,r}(\boldsymbol{\alpha}, \boldsymbol{\theta}, X = b)$ and $g_{b,r}(\boldsymbol{\alpha}, \boldsymbol{\theta}, X = \theta_{r+1})$.*
3. *The protocol performs a single homogeneous linear test on the target value a and the values read.*
4. *(Completeness.) If $a = \sum_{I \in \mathcal{F}} \tilde{f}(\overline{p}_I)$, then there exists a choice of sum-check polynomials $\{g_{b,r}\}$ such that the protocol accepts with probability 1.*
5. *If $a \neq \sum_{I \in \mathcal{F}} \tilde{f}(\overline{p}_I)$, then no matter what the sum-check polynomials $\{g_{b,r}\}$ are, the protocol accepts with probability at most $O(\frac{D+d}{|\mathbb{F}|})$.*

Before we describe the protocol, let us see what the intended meaning of $g_{b,r}$ is. For a set $I \in \mathcal{F}$, let I_i denote the ith coordinate of its characteristic vector.

For $\boldsymbol{\alpha} \in \mathbb{F}^m, I \in \mathcal{F}, 0 \le s \le m$, let

$$w(I, \boldsymbol{\alpha}, s) := \prod_{1 \le i \le s,\ I_i = 1} \alpha_i.$$

Note that in particular, for $s = 0$, $w(I, \boldsymbol{\alpha}, 0) = 1$. Also, since $|I| = d$, each $w(\cdot, \cdot, \cdot)$ is a monomial of degree at most d.

For $\boldsymbol{\theta} \in \mathbb{F}^m, I \in \mathcal{F}, 0 \le s \le m$, let $p(I, \boldsymbol{\theta}, s) \in \mathbb{F}^m$ be a point whose ith coordinate equals θ_i for $1 \le i \le s$ and equals I_i for $s < i \le m$. In particular, $p(I, \boldsymbol{\theta}, 0) = \overline{p}_I$ and $p(I, \boldsymbol{\theta}, m) = \boldsymbol{\theta}$.

Consider the following *partial sum* for $0 \le r \le m$:

$$S_r := \sum_{I \in \mathcal{F}} w(I, \boldsymbol{\alpha}, r) \tilde{f}(p(I, \boldsymbol{\theta}, r)).$$

In particular,

$$\begin{aligned} S_0 &= \sum_{I \in \mathcal{F}} w(I, \boldsymbol{\alpha}, 0) \tilde{f}(p(I, \boldsymbol{\theta}, 0)) \\ &= \sum_{I \in \mathcal{F}} \tilde{f}(\overline{p}_I). \end{aligned}$$

Thus the goal of the GSC protocol is to verify that $S_0 = a$, where a is the target value. Note that

$$S_m = \left(\sum_{I \in \mathcal{F}} w(I, \boldsymbol{\alpha}, m) \right) \tilde{f}(\boldsymbol{\theta}).$$

Here, the verifier can evaluate the quantity in the summation by himself and the value $\tilde{f}(\boldsymbol{\theta})$ can be read from the table of values of $\tilde{f}$. Thus the verifier *knows* the values S_0 and S_m and thus has to verify that the values $S_1, S_2, \ldots, S_{m-1}$ give a correct *interpolation* between S_0 and S_m.

For $b \in \{0,1\}$ and $1 \le s \le m$, let

$$\mathcal{F}_s^b := \{I \mid I \in \mathcal{F},\ I_s = b\}.$$

Thus $\mathcal{F}_s^0$ is the subfamily of $\mathcal{F}$ consisting of all sets that do not contain $s \in [m]$ and $\mathcal{F}_s^1$ consists of those sets that do contain s.

For $\boldsymbol{\theta} \in \mathbb{F}^m, I \in \mathcal{F}, 1 \le s \le m$, let $P(I, \boldsymbol{\theta}, s, X) \in \mathbb{F}^m$ be a point whose ith coordinate equals θ_i for $1 \le i < s$ and equals I_i for $s < i \le m$, and the sth coordinate is a formal variable X. Note that if $I_{s+1} = b$, then $P(I, \boldsymbol{\theta}, s+1, b) = p(I, \boldsymbol{\theta}, s)$, where $P(I, \boldsymbol{\theta}, s+1, b)$ denotes the point obtained from $P(I, \boldsymbol{\theta}, s+1, X)$ by substituting $X = b$. Also, when one substitutes $X = \theta_{s+1}$, we get $P(I, \boldsymbol{\theta}, s+1, \theta_{s+1}) = p(I, \boldsymbol{\theta}, s+1)$.

With so much notation, we are ready to see what $g_{b,r}$ are. Fix $b \in \{0,1\}$ and $0 \le r \le m-1$. Consider the sum

$$g_{b,r}(\boldsymbol{\alpha}, \boldsymbol{\theta}, X) := \sum_{I \in \mathcal{F}_{r+1}^b} w(I, \boldsymbol{\alpha}, r) \tilde{f}(P(I, \boldsymbol{\theta}, r+1, X)). \tag{8}$$

Note that $g_{b,r}$ is a polynomial in $\boldsymbol{\alpha}, \boldsymbol{\theta}$, and X. The degree in $\boldsymbol{\alpha}$ is at most d and the degree in $\boldsymbol{\theta}, X$ is at most D. Also, $g_{b,r}$ depends only on $\alpha_1, \alpha_2, \ldots, \alpha_r,\ \theta_1, \theta_2, \ldots, \theta_r$, and X.

Here is the link between the partial sums S_r and the polynomials $g_{b,r}$.

LEMMA 5.2.

$$\begin{aligned} &S_r = g_{0,r}(\boldsymbol{\alpha}, \boldsymbol{\theta}, 0) + g_{1,r}(\boldsymbol{\alpha}, \boldsymbol{\theta}, 1), \\ &g_{0,r}(\boldsymbol{\alpha}, \boldsymbol{\theta}, \theta_{r+1}) + \alpha_{r+1} g_{1,r}(\boldsymbol{\alpha}, \boldsymbol{\theta}, \theta_{r+1}) = S_{r+1}. \end{aligned}$$

Proof.

$$\begin{aligned}
S_r &= \sum_{I\in\mathcal{F}} w(I,\boldsymbol{\alpha},r)\tilde{f}(p(I,\boldsymbol{\theta},r)) \\
&= \sum_{I\in\mathcal{F}^0_{r+1}} w(I,\boldsymbol{\alpha},r)\tilde{f}(p(I,\boldsymbol{\theta},r)) + \sum_{I\in\mathcal{F}^1_{r+1}} w(I,\boldsymbol{\alpha},r)\tilde{f}(p(I,\boldsymbol{\theta},r)) \\
&= \sum_{I\in\mathcal{F}^0_{r+1}} w(I,\boldsymbol{\alpha},r)\tilde{f}(P(I,\boldsymbol{\theta},r+1,0)) + \sum_{I\in\mathcal{F}^1_{r+1}} w(I,\boldsymbol{\alpha},r)\tilde{f}(P(I,\boldsymbol{\theta},r+1,1)) \\
&= g_{0,r}(\boldsymbol{\alpha},\boldsymbol{\theta},0) + g_{1,r}(\boldsymbol{\alpha},\boldsymbol{\theta},1).
\end{aligned}$$

On the other hand,

$$\begin{aligned}
&g_{0,r}(\boldsymbol{\alpha},\boldsymbol{\theta},\theta_{r+1}) + \alpha_{r+1}g_{1,r}(\boldsymbol{\alpha},\boldsymbol{\theta},\theta_{r+1}) \\
&= \sum_{I\in\mathcal{F}^0_{r+1}} w(I,\boldsymbol{\alpha},r)\tilde{f}(P(I,\boldsymbol{\theta},r+1,\theta_{r+1})) + \sum_{I\in\mathcal{F}^1_{r+1}} (\alpha_{r+1}w(I,\boldsymbol{\alpha},r))\,\tilde{f}(P(I,\boldsymbol{\theta},r+1,\theta_{r+1})) \\
&= \sum_{I\in\mathcal{F}^0_{r+1}} w(I,\boldsymbol{\alpha},r+1)\tilde{f}(p(I,\boldsymbol{\theta},r+1)) + \sum_{I\in\mathcal{F}^1_{r+1}} w(I,\boldsymbol{\alpha},r+1)\tilde{f}(p(I,\boldsymbol{\theta},r+1)) \\
&= \sum_{I\in\mathcal{F}} w(I,\boldsymbol{\alpha},r+1)\tilde{f}(p(I,\boldsymbol{\theta},r+1)) \\
&= S_{r+1}. \qquad \square
\end{aligned}$$

COROLLARY 5.3.

$$g_{0,r}(\boldsymbol{\alpha},\boldsymbol{\theta},\theta_{r+1}) + \alpha_{r+1}g_{1,r}(\boldsymbol{\alpha},\boldsymbol{\theta},\theta_{r+1}) = g_{0,r+1}(\boldsymbol{\alpha},\boldsymbol{\theta},0) + g_{1,r+1}(\boldsymbol{\alpha},\boldsymbol{\theta},1).$$

5.1. The GSC protocol. The protocol works as follows. The protocol accepts if and only if all of the following checks are satisfied (we later combine these checks into a single homogeneous linear test):

- Pick $\boldsymbol{\alpha},\boldsymbol{\theta}\in\mathbb{F}^m$ at random.
- (Stage 0:) Check if

$$a = g_{0,0}(\boldsymbol{\alpha},\boldsymbol{\theta},0) + g_{1,0}(\boldsymbol{\alpha},\boldsymbol{\theta},1). \tag{9}$$

- (Stage $r+1$ for $r=0,1,\ldots,m-2$:) Check if

$$g_{0,r}(\boldsymbol{\alpha},\boldsymbol{\theta},\theta_{r+1}) + \alpha_{r+1}g_{1,r}(\boldsymbol{\alpha},\boldsymbol{\theta},\theta_{r+1}) = g_{0,r+1}(\boldsymbol{\alpha},\boldsymbol{\theta},0) + g_{1,r+1}(\boldsymbol{\alpha},\boldsymbol{\theta},1). \tag{10}$$

- (Stage m:) Check if

$$g_{0,m-1}(\boldsymbol{\alpha},\boldsymbol{\theta},\theta_m) + \alpha_m g_{1,m-1}(\boldsymbol{\alpha},\boldsymbol{\theta},\theta_m) = \left(\sum_{I\in\mathcal{F}} w(I,\boldsymbol{\alpha},m)\right)\tilde{f}(\boldsymbol{\theta}). \tag{11}$$

Note that the values on lines (9), (10), (11) are (supposed to be) S_0, S_{r+1}, S_m, respectively.

5.2. Completeness.

LEMMA 5.4. *If the polynomial $\tilde{f}$ satisfies* (7)*, then it is possible to define polynomials $g_{b,r}(\boldsymbol{\alpha},\boldsymbol{\theta},X)$ so that the protocol accepts with probability* 1.

Proof. The proof is obvious. Just define $g_{b,r}$ as they are supposed to be as in (8). $\square$

5.3. Soundness.

LEMMA 5.5. *If the polynomial $\tilde{f}$ does not satisfy* (7)*, then no matter what polynomials $g_{b,r}$ one takes, the protocol accepts with negligible probability, i.e., with probability at most $O(\frac{D+d}{|\mathbb{F}|})$. It is assumed that the polynomials $g_{b,r}$ have degree at most $D+d$ and $g_{b,r}$ depends only on $\alpha_1,\ldots,\alpha_r$, $\theta_1,\ldots,\theta_r$, and X.*

Proof. Let $\tilde{g}_{b,r}$ be polynomials obtained from $\tilde{f}$ using rule (8). We will argue in reverse order (i.e., $r = m-1, m-2, \ldots, 0$) that unless $g_{b,r} \equiv \tilde{g}_{b,r}$, the protocol succeeds with negligible probability. The only fact we use is that two polynomials agree with nonnegligible probability if and only if they are identical as formal polynomials.

(*Case* $r = m-1$.) Look at Stage m, equation (11). The RHS is just S_m, which, using Lemma 5.2, can be written in terms of $\tilde{g}_{0,m-1}$ and $\tilde{g}_{1,m-1}$. Thus we get

$$g_{0,m-1}(\boldsymbol{\alpha},\boldsymbol{\theta},\theta_m) + \alpha_m g_{1,m-1}(\boldsymbol{\alpha},\boldsymbol{\theta},\theta_m) = \tilde{g}_{0,m-1}(\boldsymbol{\alpha},\boldsymbol{\theta},\theta_m) + \alpha_m \tilde{g}_{1,m-1}(\boldsymbol{\alpha},\boldsymbol{\theta},\theta_m).$$

Note that $g_{0,m-1}, g_{1,m-1}, \tilde{g}_{0,m-1}, \tilde{g}_{1,m-1}$ do not depend on α_m, θ_m. Therefore, if this is a formal identity, then we must have $g_{0,m-1} = \tilde{g}_{0,m-1}$ and $g_{1,m-1} = \tilde{g}_{1,m-1}$.

(*Case* $0 \le r \le m-2$.) Assume that we have already proved that $g_{0,r+1} = \tilde{g}_{0,r+1}$ and $g_{1,r+1} = \tilde{g}_{1,r+1}$. Therefore, the RHS of (10) is

$$\begin{aligned} g_{0,r+1}(\boldsymbol{\alpha},\boldsymbol{\theta},0) + g_{1,r+1}(\boldsymbol{\alpha},\boldsymbol{\theta},1) &= \tilde{g}_{0,r+1}(\boldsymbol{\alpha},\boldsymbol{\theta},0) + \tilde{g}_{1,r+1}(\boldsymbol{\alpha},\boldsymbol{\theta},1) \\ &= \tilde{g}_{0,r}(\boldsymbol{\alpha},\boldsymbol{\theta},\theta_{r+1}) + \alpha_{r+1}\tilde{g}_{1,r}(\boldsymbol{\alpha},\boldsymbol{\theta},\theta_{r+1}), \end{aligned}$$

where we used Corollary 5.3. Now we equate this with the LHS of (10) and note that $g_{0,r}, g_{1,r}, \tilde{g}_{0,r}, \tilde{g}_{1,r}$ do not depend on $\alpha_{r+1}, \theta_{r+1}$. Hence we conclude that $g_{0,r} = \tilde{g}_{0,r}$ and $g_{1,r} = \tilde{g}_{1,r}$

Now we show that the test fails at Stage 0. This is because, $g_{0,0} = \tilde{g}_{0,0}, g_{1,0} = \tilde{g}_{1,0}$ and

$$\begin{aligned} a &\neq \sum_{I\in\mathcal{F}} \tilde{f}(\overline{p}_I) \\ &= \tilde{g}_{0,0}(\boldsymbol{\alpha},\boldsymbol{\theta},0) + \tilde{g}_{1,0}(\boldsymbol{\alpha},\boldsymbol{\theta},1) \\ &= g_{0,0}(\boldsymbol{\alpha},\boldsymbol{\theta},0) + g_{1,0}(\boldsymbol{\alpha},\boldsymbol{\theta},1). \qquad \square \end{aligned}$$

5.4. Combining all tests into a single homogeneous test. Instead of verifying (9), (10), and (11) separately, the protocol can multiply these equations with random values in $\mathbb{F}$, add them up, and verify that the single combined equation is satisfied. It is easily seen that the completeness and soundness properties are still satisfied.

6. Constructing subspace membership certificate. Now we return to the task we started with. We want polynomials g and h that certify that the polynomial f in section 3.1 satisfies (3). The equation says that $f(\mathbf{q})$ equals a sum of values of $\psi(\mathbf{q},\cdot)f(\cdot)$. We verify this by running the generalized sum-check protocol on $\psi(\mathbf{q},\cdot)f(\cdot)$. The polynomial g will be obtained by *clubbing together* all the sum-check polynomials. The polynomial h will certify that g is splitable, a property that makes sure that the sum-check protocol works (as explained below).

We want to verify (3) for a random $\mathbf{q}$. Let $\tilde{f}[\mathbf{q},\mathbf{x}] = \psi(\mathbf{q},\mathbf{x})f(\mathbf{x})$ and $a = f(\mathbf{q})$. Thus we can run the protocol

$$GSC(a = f(\mathbf{q}), \mathbb{F}, \tilde{f}, m, D = 3d, d, \mathcal{F}, \{g_{b,r} | b \in \{0,1\}, 0 \le r \le m-1\}).$$

Here, $\tilde{f}$ is a polynomial in $\mathbf{q}, \mathbf{x}$, but the sum-check protocol is run on the $\mathbf{x}$ part only. Usually, the sum-check polynomials $g_{b,r}$ are polynomials in $\boldsymbol{\alpha}, \boldsymbol{\theta}, X$. But now, examining rule (8), they are polynomials also in $\mathbf{q}$. Since $g_{b,r}$ is polynomial of degree $D+d$, we can write it in increasing powers of X as

$$g_{b,r}(\mathbf{q}, \boldsymbol{\alpha}, \boldsymbol{\theta}, X) := \sum_{t=0}^{D+d} g_{b,r,t}(\mathbf{q}, \boldsymbol{\alpha}, \boldsymbol{\theta}) X^t,$$

where $g_{b,r,t}$ are polynomials in $\mathbf{q}, \alpha_1, \ldots, \alpha_r, \theta_1, \ldots, \theta_r$.

The GSC protocol (section 5.1) needs values $a = f(\mathbf{q})$ and $\tilde{f}(\mathbf{q}, \boldsymbol{\theta})$. The protocol reads values $f(\mathbf{q})$ and $f(\boldsymbol{\theta})$ from the table of values of f. The value $\tilde{f}(\mathbf{q}, \boldsymbol{\theta}) = \psi(\mathbf{q}, \boldsymbol{\theta}) f(\boldsymbol{\theta})$ is computed from the value $f(\boldsymbol{\theta})$ and by using (2). In addition, the protocol needs the value of some homogeneous linear form on the values:

$$g_{b,r,t}(\mathbf{q}, \boldsymbol{\alpha}, \boldsymbol{\theta}) \;\; b \in \{0,1\},\; 0 \le r \le m-1,\; 0 \le t \le D+d.$$

However, we can afford only a constant number of queries. So reading all these values separately is ruled out. We do the following trick: we just *read off* the value of the linear form as a single query. This can be done if we have access to the Hadamard code over the polynomials $g_{b,r,t}$. This is exactly what the *certificate polynomial* g will do. As a first attempt (which does not work), we expect to have the following polynomial g^* as a certificate that enables us to read off the linear form. Let $R_{b,r,t}$ be formal variables and let $\mathbf{R}$ denote the vector of all these variables. The polynomial g^* is intended to be

$$g^*(\mathbf{q}, \boldsymbol{\alpha}, \boldsymbol{\theta}, \mathbf{R}) := \sum_{b,r,t} g_{b,r,t}(\mathbf{q}, \boldsymbol{\alpha}, \boldsymbol{\theta}) R_{b,r,t}. \tag{12}$$

We are not quite done yet. How do we ensure that the supposed polynomial g^* given by an adversary is of the form (12)? First of all, note that g^* is intended to be a linear function when treated as a function of $\mathbf{R}$. This can be checked by substituting random values of $\mathbf{q}, \boldsymbol{\alpha}, \boldsymbol{\theta}$ and then using a linearity test. Assuming now that g^* is indeed a linear function of $\mathbf{R}$, let $g^*_{b,r,t}$ denote the coefficient of $R_{b,r,t}$.

The problem is the following: how do we ensure that $g^*_{b,r,t}$ depends only on $\mathbf{q}, \alpha_1, \ldots, \alpha_r, \theta_1, \ldots, \theta_r$ and is independent of $\alpha_{r+1}, \ldots, \alpha_m$, $\theta_{r+1}, \ldots, \theta_m$? This property is necessary for the GSC protocol to work.

In order to fix this problem, we need to redefine the polynomial g^*. We give a *fresh copy* of variables to every polynomial $g_{b,r,t}$. To be precise, for any (b,r,t), let $\mathbf{q}^{(b,r,t)}$ denote a vector of m variables which is supposed to be a *copy* of $\mathbf{q}$. Also, let $\boldsymbol{\alpha}^{(b,r,t)}$ denote a copy of $\alpha_1, \ldots, \alpha_r$ and let $\boldsymbol{\theta}^{(b,r,t)}$ denote a copy of $\theta_1, \ldots, \theta_r$.

Let

$$\mathbf{q}' = \cup_{b,r,t} \mathbf{q}^{(b,r,t)}, \quad \boldsymbol{\alpha}' = \cup_{b,r,t} \boldsymbol{\alpha}^{(b,r,t)}, \quad \boldsymbol{\theta}' = \cup_{b,r,t} \boldsymbol{\theta}^{(b,r,t)}, \quad \mathbf{R} = \cup_{b,r,t} \{R_{b,r,t}\}. \tag{13}$$

Define

$$g(\mathbf{q}', \boldsymbol{\alpha}', \boldsymbol{\theta}', \mathbf{R}) := \sum_{b,r,t} g_{b,r,t}(\mathbf{q}^{(b,r,t)}, \boldsymbol{\alpha}^{(b,r,t)}, \boldsymbol{\theta}^{(b,r,t)}) R_{b,r,t}. \tag{14}$$

Note that g is splitable. The sets of variables on which $g_{b,r,t}$'s depend are disjoint for different (b,r,t)'s. Now we can attach a splitability certificate h to make sure that an adversary is forced to give a g that is splitable!

6.1. Proving Theorem 3.4. Now we prove Theorem 3.4. First we prove a version of the theorem where instead of a single polynomial f in the HomAlgCSP instance we have three polynomials f, g, h and the constraints are homogeneous constraints on $O(1)$ values of these polynomials.

Given tables of values of polynomials $f(\mathbf{q})$ and $g(\mathbf{q}', \boldsymbol{\alpha}', \boldsymbol{\theta}', \mathbf{R})$ and a splitability certificate h, we run the following verification procedure.

Checking that g is linear in $\mathbf{R}$. Pick random values for $\mathbf{q}', \boldsymbol{\alpha}', \boldsymbol{\theta}'$ and two sets of random values for $\mathbf{R}$ and $\mathbf{R}'$. Pick two random scalar/field values ζ, ζ' and check if

$$g(\mathbf{q}', \boldsymbol{\alpha}', \boldsymbol{\theta}', \zeta\mathbf{R} + \zeta'\mathbf{R}') = \zeta \cdot g(\mathbf{q}', \boldsymbol{\alpha}', \boldsymbol{\theta}', \mathbf{R}) + \zeta' \cdot g(\mathbf{q}', \boldsymbol{\alpha}', \boldsymbol{\theta}', \mathbf{R}').$$

Checking that h is a splitability certificate. This can be done as in section 4.3 by reading one value of g and four values of h.

Running the GSC protocol. Pick random values for $\mathbf{q}, \boldsymbol{\alpha}, \boldsymbol{\theta}$. The values $f(\mathbf{q}), f(\boldsymbol{\theta})$ needed by the protocol are read from the table of values for f. The protocol also needs a value of a linear form of type

$$\sum_{b,r,t} \zeta_{b,r,t}\ g_{b,r,t}(\mathbf{q}, \boldsymbol{\alpha}, \boldsymbol{\theta}), \qquad \zeta_{b,r,t} \in \mathbb{F}. \tag{15}$$

For all (b, r, t), we set $\mathbf{q}^{(b,r,t)} = \mathbf{q}$, $\boldsymbol{\alpha}^{(b,r,t)} = (\alpha_1, \alpha_2, \ldots, \alpha_r)$, $\boldsymbol{\theta}^{(b,r,t)} = (\theta_1, \theta_2, \ldots, \theta_r)$ (this makes sense because $\mathbf{q}^{(b,r,t)}, \boldsymbol{\alpha}^{(b,r,t)}, \boldsymbol{\theta}^{(b,r,t)}$ are supposed to be copies of the same $\mathbf{q}, \boldsymbol{\alpha}, \boldsymbol{\theta}$, respectively). Set $\mathbf{R}_{b,r,t} = \zeta_{b,rt}$. Define $\mathbf{q}', \boldsymbol{\alpha}', \boldsymbol{\theta}', \mathbf{R}$ as in (13).

Finally, take the value of the linear form in (15) needed by the protocol to be the value $g(\mathbf{q}', \boldsymbol{\alpha}', \boldsymbol{\theta}', \mathbf{R})$.

Checking that f is a solution to the MDC instance. Pick a random point from set $\{\overline{p}_J \mid J \in \mathcal{F}_0\}$ and check whether $f(\overline{p}_J) = 0$.

Number of queries. Note that the verification procedure reads three values of f (namely, $f(\overline{p}_J), f(\mathbf{q}), f(\boldsymbol{\theta})$). It reads five values of g (three for checking linearity in $\mathbf{R}$, one for checking splitability, one for reading linear form needed in sum-check). It reads four values of h (as in splitability check). Thus we have a total of 12 queries.

Relating the optimum of MDC with the maximum acceptance probability. It is clear that if the MDC instance $\mathbf{A}$ has a solution with value $OPT(\mathbf{A})$, then we can take this solution, encode it as a nonzero polynomial f, and construct g, h as they should be. The verification procedure accepts whenever $f(\overline{p}_J) = 0$, which happens with probability $1 - OPT(\mathbf{A})$.

On the other hand, note that the verification process accepts with negligible probability unless $f \in span\{f_I | I \in \mathcal{F}\}$. If $f \equiv 0$, then the corresponding polynomials g, h must also be identically zero (otherwise probability of acceptance is negligible). Now if f is a nonzero polynomial in the span, we can have $f(\overline{p}_J) = 0$ only for $1 - OPT(\mathbf{A})$ fraction of $J \in \mathcal{F}_0$. Hence the verification procedure can accept with probability at most $\max\{1 - OPT(\mathbf{A}), O(\frac{d}{|\mathbb{F}|})\}$.

6.2. Clubbing together f, g, h. Now we show how to combine f, g, h into a single polynomial ϕ and thus prove Theorem 3.4. Assume that $f \in \mathbb{F}[\mathbf{x}]$, $g \in \mathbb{F}[\mathbf{y}]$, $h \in \mathbb{F}[\mathbf{z}]$, where $\mathbf{x}, \mathbf{y}, \mathbf{z}$ are disjoint sets of variables. Let X, Y, Z be formal variables and let ϕ (as intended to) be

$$\phi(\mathbf{x}, \mathbf{y}, \mathbf{z}, X, Y, Z) := Xf(\mathbf{x}) + Yg(\mathbf{y}) + Zh(\mathbf{z}).$$

Again the trouble is that the adversary might give an arbitrary polynomial ϕ and not necessarily of this form. We give the following verification procedure.

Checking linearity in X, Y, Z. Pick random values for $\mathbf{x}, \mathbf{y}, \mathbf{z}$ and random values X, X', Y, Y', Z, Z'. Pick random values ζ, ζ'. Check if

$$\phi(\mathbf{x}, \mathbf{y}, \mathbf{z}, \zeta X + \zeta' X', \zeta Y + \zeta' Y', \zeta Z + \zeta' Z') = \zeta \cdot \phi(\mathbf{x}, \mathbf{y}, \mathbf{z}, X, Y, Z) + \zeta' \cdot \phi(\mathbf{x}, \mathbf{y}, \mathbf{z}, X', Y', Z').$$

Checking splitability. Thus we can assume that ϕ is linear and hence of the form $X\tilde{f} + Y\tilde{g} + Z\tilde{h}$. We need to ensure that $\tilde{f}$ depends only on $\mathbf{x}$ and not on $\mathbf{y}, \mathbf{z}$, and also that $\tilde{g}$ depends only on $\mathbf{y}$ and $\tilde{h}$ depends only on $\mathbf{z}$. This can be done by *brute force* as follows. Note that setting $X = 1, Y = 0, Z = 0$ gives us *access* to $\tilde{f}$ and similarly for $\tilde{g}, \tilde{h}$.

- Pick random values $\mathbf{x}, \mathbf{y}, \mathbf{z}$. Pick another set of random values $\mathbf{x}', \mathbf{y}', \mathbf{z}'$.
- Check that

$$\phi(\mathbf{x}, \mathbf{y}, \mathbf{z}, 1, 0, 0) = \phi(\mathbf{x}, \mathbf{y}', \mathbf{z}', 1, 0, 0),$$
$$\phi(\mathbf{x}, \mathbf{y}, \mathbf{z}, 0, 1, 0) = \phi(\mathbf{x}', \mathbf{y}, \mathbf{z}', 0, 1, 0),$$
$$\phi(\mathbf{x}, \mathbf{y}, \mathbf{z}, 0, 0, 1) = \phi(\mathbf{x}', \mathbf{y}', \mathbf{z}, 0, 0, 1).$$

Simulating the previous verification procedure. Now we can assume that ϕ is of the form $Xf(\mathbf{x}) + Yg(\mathbf{y}) + Zh(\mathbf{z})$. We run the verification procedure described in section 6.1. Values for $f[\mathbf{x}], g[\mathbf{y}], h[\mathbf{z}]$ can be accessed by reading $\phi(\mathbf{x}, \mathbf{0}, \mathbf{0}, 1, 0, 0)$, $\phi(\mathbf{0}, \mathbf{y}, \mathbf{0}, 0, 1, 0)$, and $\phi(\mathbf{0}, \mathbf{0}, \mathbf{z}, 0, 0, 1)$, respectively.

Note that the new protocol reads $3 + 6 + 12 = 21$ queries from table of values of ϕ. The maximum probability of acceptance (for any nonzero polynomial ϕ) equals $1 - OPT(\mathbf{A})$, where $\mathbf{A}$ is the MDC instance one starts with. The constraints are all homogeneous linear. This proves Theorem 3.4.

7. The LDT and outer verifier. In this section, we describe our outer verifier. Later we modify the outer verifier and build a Hadamard code–based inner verifier. This gives a quasi-random PCP.

The starting point for the outer verifier is the gap-instance of HomAlgCSP obtained by combining Theorems 1.5 and 3.4. Let $\mathcal{A}(f, k, d^*, m, \mathbb{F}, \mathcal{C})$ denote this HomAlgCSP instance. We have either

$$OPT(\mathcal{A}) \geq 1 - 1/2^K \qquad \text{or} \qquad OPT(\mathcal{A}) \leq 1/2^{2^K}.$$

Remark 7.1. It is helpful to keep in mind the quantitative parameters. If the size of the SAT instance is n, then the size of the MDC instance is $N \leq n^{CK}$, and $N \leq |\mathbb{F}| \leq N^2$, $m = N^\epsilon$, $d^* = 100/\epsilon$. In particular, the degree d^* is constant. The size of the HomAlgCSP instance is $|\mathbb{F}|^{O(m)} \leq 2^{O(N^\epsilon)}$.

The outer verifier is given the polynomial f as a table of values. He wants to pick a random constraint $C(\{\overline{p}_i\}_{i=1}^k) \in \mathcal{C}$ and check that it is satisfied. However, an adversary could cheat by giving a table which is *not* a low degree polynomial. The *LDT* allows the verifier to guard against such cheating. The LDT was first proposed by Rubinfeld and Sudan [35] and has been a crucial ingredient in the proof of the PCP theorem (see [4], [7]).

7.1. Outer verifier.

DEFINITION 7.2. *A line ℓ in $\mathbb{F}^m$ is a linear function $\ell : \mathbb{F} \mapsto \mathbb{F}^m$. In other words, for some $\overline{a}, \overline{b} \in \mathbb{F}^m$,*

$$\ell(t) = \overline{a} + \overline{b}t.$$

DEFINITION 7.3. *A curve L in $\mathbb{F}^m$ is a function $L : \mathbb{F} \mapsto \mathbb{F}^m$. In other words*

$$L(t) = \overline{a}(t) = (a_1(t), a_2(t), \ldots, a_m(t)).$$

It is a degree d curve if each of the coordinate functions $a_i(t)$ is a degree d (univariate) polynomial.

We fix a constraint $C(\{\overline{p_i}\}_{i=1}^k) \in \mathcal{C}$ for this section and the next.

Let $t_1, t_2, \ldots, t_{k+3}$ be fixed distinct field elements in $\mathbb{F}$. For $\overline{a}, \overline{b}, \overline{c} \in \mathbb{F}^m$, let $L = L_{\overline{a},\overline{b},\overline{c}}$ be the unique degree $k+2$ curve that passes through points $\{\overline{p}_i\}_{i=1}^k, \overline{a}, \overline{b}, \overline{c}$. More precisely,

$$L(t_i) = \overline{p}_i,\ 1 \le i \le k, \quad L(t_{k+1}) = \overline{a},\ L(t_{k+2}) = \overline{b},\ L(t_{k+3}) = \overline{c}. \tag{16}$$

If f is a degree d^* polynomial, then its restriction to the curve $L = L_{\overline{a},\overline{b},\overline{c}}$ is a degree $d - 1 := (k+2)d^*$ univariate polynomial. Call this polynomial $f|_L$. In particular, $f|_L$ can be *reconstructed* from *any* d values of f on the curve. The outer verifier does precisely this. He picks $d+1$ random points from the curve, say $\overline{v}_1, \overline{v}_2, \ldots, \overline{v}_d, \overline{v}$. He reconstructs $f|_L$ using values of f at first d points, and verifies that the value of $f|_L$ at point $\overline{v}$ is the same as the value $f(\overline{v})$. In addition, *knowing* $f|_L$, the verifier *calculates* the values of f at points $\{\overline{p}_i\}_{i=1}^k$ and verifies that these values indeed satisfy the constraint C.

However, we need to combine the above task with the line-point LDT. We will use the strong analysis of LDT by Arora and Safra [4]. The outer verifier becomes somewhat complicated. Note that restriction of f to a line is a degree d^* univariate polynomial. We will be generous and allow the degree to be $d - 2$.

The verifier picks a random pair of a line ℓ and a curve L intersecting at a point $\overline{v}$. The univariate polynomials $f|_L$ and $f|_\ell$ are reconstructed by reading values of f at d random points on L and $d-1$ random points on ℓ, respectively. Then the verifier checks that $f|_L$ and $f|_\ell$ both have the value $f(\overline{v})$ at point $\overline{v}$. In addition, the verifier checks that values of $f|_L$ at points $\{\overline{p}_i\}_{i=1}^k$ satisfy the constraint C.

To prove the correctness of the outer verifier, we need to be careful about how the intersecting line-curve pair is chosen. Formally, the action of the verifier is as follows (see Figure 2).

OUTER VERIFIER.

1. Pick a random line ℓ and pick random points $\overline{v}_1, \ldots, \overline{v}_{d-1}, \overline{v}_d$ on the line (assume that these points are all distinct; this happens w.h.p.). Let $\overline{v} = \overline{v_d}$.
2. Pick $t \in \mathbb{F} \setminus \{t_1, \ldots, t_{k+3}\}$ at random, pick points $\overline{a}, \overline{b}$ at random, and let L be the unique degree $k+2$ curve $L = L_{\overline{a},\overline{b},\overline{c}}$ such that

$$L(t_i) = \overline{p}_i, 1 \le i \le k, \quad L(t_{k+1}) = \overline{a}, \quad L(t_{k+2}) = \overline{b}, \quad L(t) = \overline{v}$$

   (and $\overline{c} = L(t_{k+3})$ gets defined automatically).
3. Pick random points $\overline{v}_{d+1}, \ldots, \overline{v}_{2d}$ on curve L (we assume that these points are distinct; this happens w.h.p.).
4. Read the values $f(\overline{v}_i)$ for $1 \le i \le 2d$.
5. Let $f|_\ell$ be the unique degree $d-2$ polynomial *interpolated* using the values $\{f(\overline{v}_i)\}_{i=1}^{d-1}$.
6. Let $f|_L$ be the unique degree $d-1$ polynomial *interpolated* using the values $\{f(\overline{v}_i)\}_{i=d+1}^{2d}$.
7. Check if

$$f|_L(\overline{v}) = f(\overline{v}) = f|_\ell(\overline{v}).$$

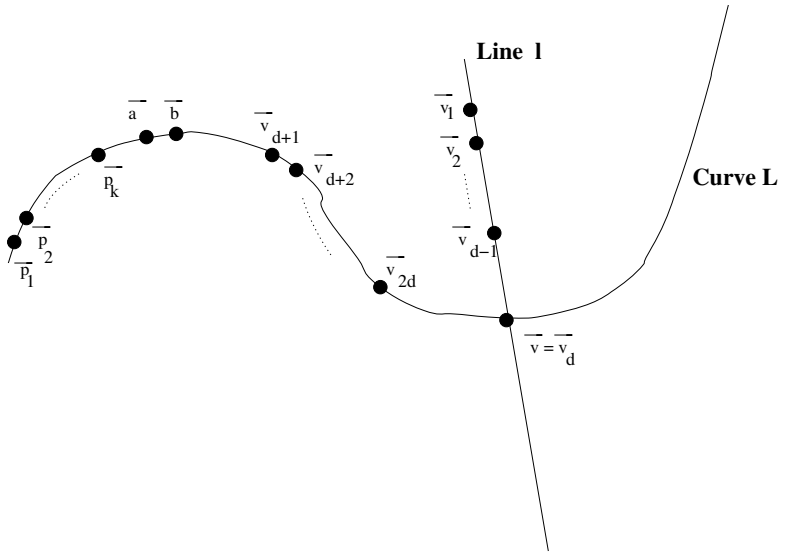


FIG. 2. *Outer verifier.*

8. Check if the values of $f|_L$ at points $\{\overline{p}_i\}_{i=1}^{k}$ satisfy the constraint C.
9. Check that not all the values read are zero, i.e., Not-All-Zero$(f(\overline{v}_i), 1 \le i \le 2d)$.

Remark 7.4. Note that the polynomials $f|_\ell$ and $f|_L$ depend on the specific choice of points $\{\overline{v}_i\}_{i=1}^{d-1}$ and $\{\overline{v}_i\}_{i=d+1}^{2d}$, respectively (i.e., these are the interpolated polynomials). We hide this fact for notational convenience. Also, a main step in the proof of Theorem 7.6 is to get rid of this dependence.

Remark 7.5. The verifier's action is straightforward except for the last step, where he checks that not all values read are zero. This is needed to make sure that the adversary does not cheat by giving a function f that is identically zero.

Given a table $f : \mathbb{F}^m \mapsto \mathbb{F}$ and an m-variate polynomial P, the *agreement* between f and P is defined to be the fraction of points $\overline{u} \in \mathbb{F}^m$ such that $f(\overline{u}) = P(\overline{u})$. Now we state the main theorem in this section. Its proof appears in the next section.

THEOREM 7.6. *There are constants c_2, c_3 such that the following holds: Let $P_1, P_2, \ldots, P_t$ be all degree d polynomials that have agreement at least $(\delta/2)^{c_2}/c_3$ with f. Then if the outer verifier accepts with probability δ, then for some $1 \le j \le t$, the polynomial P_j is a nonzero polynomial and its values at points $\{\overline{p}_i\}_{i=1}^{k}$ satisfy the constraint C. Also, $t \le 2c_3/(\delta/2)^{c_2}$.*

A crucial ingredient in the proof of Theorem 7.6 is the strong analysis of the LDT as in Arora and Sudan's paper [5] (analysis of Raz and Safra [34] could be applied as well). Let f_{lines} be a table which gives, for every line ℓ, a univariate polynomial of degree d, denoted by $f_{lines}(\ell)$.

The following theorem states the result of Arora and Sudan [5] (Theorem 3 and Proposition 2 therein). In our application, d is constant and $|\mathbb{F}|$ is a large polynomial in m (recall that $N \le |\mathbb{F}|$ and $m = N^\epsilon$, where N is the size of the MDC instance used by our reduction).

THEOREM 7.7. *There are constants c_0, c_1, c_2, c_3 such that the following holds: Let δ, d, m be parameters and let $\mathbb{F}$ be a field satisfying $|\mathbb{F}| \ge c_0(dm/\delta)^{c_1}$. Fix tables $f : \mathbb{F}^m \mapsto \mathbb{F}$ and f_{lines}. Let $\{P_1, P_2, \ldots, P_t\}$ be all degree d polynomials that have*

agreement at least δ^{c_2}/c_3 with f. Let a point $\overline{v} \in \mathbb{F}^m$ and a line ℓ passing through $\overline{v}$ be picked at random. Then

$$\Pr_{\overline{v},\ell}\left[f(\overline{v}) \notin \{P_1(\overline{v}), \ldots, P_t(\overline{v})\} \quad \textit{and} \quad f_{lines}(\ell)(\overline{v}) = f(\overline{v})\ \right] \ \leq \ \delta.$$

8. Proof of Theorem 7.6. In this section, we prove Theorem 7.6. Before applying Theorem 7.7, we need to go through a rather long sequence of arguments.

We first observe that there are two equivalent ways in which the outer verifier can make his probabilistic choices. The first way is as in steps (1)–(3). An alternate way is as follows:

- Pick $\overline{a}, \overline{b}, \overline{c}$ at random and let $L = L_{\overline{a},\overline{b},\overline{c}}$ be the curve as in (16). Pick random points $\overline{v}_{d+1}, \ldots, \overline{v}_{2d}$ on L.
- Pick a random point $\overline{v} = L(t), t \in \mathbb{F} \setminus \{t_1, \ldots, t_{k+3}\}$, on the curve. Let $\overline{v}_d = \overline{v}$.
- Pick a random line ℓ through $\overline{v}$ and pick random points $\overline{v}_1, \ldots, \overline{v}_{d-1}$ on line ℓ.

Let NZ be the event that not all values read by the verifier are zero. Let SAT^C be the event that the values of the polynomial $f|_L$ at points $\{\overline{p}_i\}_{i=1}^k$ satisfy the constraint C. The probability of acceptance of the outer verifier is

$$\Pr\left[f|_L(\overline{v}) = f(\overline{v}) = f|_\ell(\overline{v}), \quad NZ, \quad SAT^C\right]. \tag{17}$$

We will construct two new verifiers $\mathcal{V}$ and $\tilde{\mathcal{V}}$ whose acceptance probability is at least equal to the acceptance probability of the outer verifier. We finish the proof by analyzing the verifier $\tilde{\mathcal{V}}$.

8.1. Constructing verifiers $\mathcal{V}$ and $\tilde{\mathcal{V}}$. We construct a table f_{lines} that gives a single univariate polynomial of degree $d-2$ (call it $f_{lines}(\ell)$) for every line ℓ. Then we show that the probability of acceptance of the following verifier (call it $\mathcal{V}$) is at least equal to the probability of acceptance of the outer verifier: the verifier $\mathcal{V}$, instead of reconstructing $f|_\ell$ from the values $\{f(\overline{v}_i)\}_{i=1}^{d-1}$, directly uses the polynomial $f_{lines}(\ell)$. Also, instead of checking for the event NZ, it checks that not all of these are zero: the polynomial $f_{lines}(\ell)$ and the values $f(\overline{v})$, $\{f(\overline{v}_i)\}_{i=d+1}^{2d}$. Let NZ' denote this event.

It is easy to construct f_{lines}. Think of the verifier's probabilistic choices in this order: $\ell, \{\overline{v}_i\}_{i=1}^{d-1}$, $\overline{v}, L, \{\overline{v}_i\}_{i=d+1}^{2d}$. For any line ℓ, fix a choice of $\{\overline{v}_i\}_{i=1}^{d-1}$ so as to maximize the probability in (17). Define $f_{lines}(\ell)$ to be the polynomial obtained by interpolating from this *optimal* choice of $\{\overline{v}_i\}_{i=1}^{d-1}$. Thus probability of acceptance of $\mathcal{V}$ is

$$\Pr\left[f|_L(\overline{v}) = f(\overline{v}) = f_{lines}(\ell)(\overline{v}), \quad NZ', \quad SAT^C\right]. \tag{18}$$

Now we do a similar trick again. We construct a table f_{curves} that gives a single univariate polynomial of degree $d-1$ (call it $f_{curves}(L)$) for every curve L. Then we show that the probability of acceptance of the following verifier (call it $\tilde{\mathcal{V}}$) is at least equal to the probability of acceptance of $\mathcal{V}$: this verifier $\tilde{\mathcal{V}}$, instead of reconstructing $f|_L$ from the values $\{f(\overline{v}_i)\}_{i=d+1}^{2d}$, directly uses the polynomial $f_{curves}(L)$. Also, instead of checking for the event NZ', it checks that not all of these are zero: the polynomial $f_{lines}(\ell)$, the value $f(\overline{v})$, and the polynomial $f_{curves}(L)$. Let NZ'' denote this event.

It is again easy to construct f_{curves}. Think of the verifier's probabilistic choices in the alternative (but equivalent) order: $L, \{\overline{v}_i\}_{i=d+1}^{2d}, \overline{v}, \ell, \{\overline{v}_i\}_{i=1}^{d-1}$. For any curve L, fix a choice of $\{\overline{v}_i\}_{i=d+1}^{2d}$ so as to maximize the probability in (18). Define $f_{curves}(L)$

to be the polynomial obtained by interpolating from this *optimal* choice of $\{\overline{v}_i\}_{i=d+1}^{2d}$. Thus probability of acceptance of $\tilde{\mathcal{V}}$ is

$$\Pr\left[f_{curves}(L)(\overline{v}) = f(\overline{v}) = f_{lines}(\ell)(\overline{v}), \quad NZ'', \quad SAT^C\right].$$

If the outer verifier accepts with probability δ, then we have

$$\Pr\left[f_{curves}(L)(\overline{v}) = f(\overline{v}) = f_{lines}(\ell)(\overline{v}), \quad NZ'', \quad SAT^C\right] \geq \delta. \tag{19}$$

We observe that unless $f_{curves}(L) \neq 0$, the above probability is negligible. This is because if $f_{curves}(L) = 0$, one would have $0 = f(\overline{v}) = f_{lines}(\ell)(\overline{v})$ and $NZ'' \implies f_{lines}(\ell) \neq 0$. Since $\overline{v}$ is a random point on line ℓ, this can happen only with negligible probability.

Thus we can replace the condition NZ'' by the (more stringent) condition $f_{curves}(L) \neq 0$ without decreasing the above probability much. To be precise,

$$\Pr\left[f_{curves}(L)(\overline{v}) = f(\overline{v}) = f_{lines}(\ell)(\overline{v}), \quad f_{curves}(L) \neq 0, \quad SAT^C\right] \geq 3\delta/4. \tag{20}$$

8.2. Analyzing verifier $\tilde{\mathcal{V}}$. Let $P_1, P_2, \ldots, P_t$ be all degree d polynomials that agree with f at $(\delta/2)^{c_2}/c_3$ fraction of points. Let $GOOD$ be the event that $f(\overline{v}) \in \{P_1(\overline{v}), \ldots, P_t(\overline{v})\}$. Using Theorem 7.7, we see that

$$\Pr_{\overline{v},\ell}\left[\neg GOOD, \quad f_{lines}(\ell)(\overline{v}) = f(\overline{v})\ \right] \ \leq\ \delta/2. \tag{21}$$

From (20) and (21), we get

$$\Pr\left[f_{curves}(L)(\overline{v}) = f(\overline{v}) = f_{lines}(\ell)(\overline{v}),\ GOOD, \quad f_{curves}(L) \neq 0, \quad SAT^C\right] \geq \delta/4. \tag{22}$$

Now, let $GOOD_{curves}$ be the event that

$$f_{curves}(L) \in \{P_1(L), \ldots, P_t(L)\},$$

where $P_j(L)$ denotes the univariate polynomial that is the restriction of the polynomial P_j to L.

We observe that unless the event $GOOD_{curves}$ happens, the probability of acceptance in (22) is negligible. This is because $\overline{v}$ is a random point on the curve L. If it happens that $f_{curves}(L)(\overline{v}) = f(\overline{v}) \in \{P_1(\overline{v}), \ldots, P_t(\overline{v})\}$ with nonnegligible probability, then it must be the case that $f_{curves}(L)$ is identically equal to one of $P_1(L), \ldots, P_t(L)$. Hence, we conclude

$$\Pr\Big[f_{curves}(L)(\overline{v}) = f(\overline{v}) = f_{lines}(\ell)(\overline{v}), GOOD,\ GOOD_{curves}, \\ f_{curves}(L) \neq 0, \quad SAT^C\Big] \geq \delta/8.$$

Whenever the above event occurs, $f_{curves}(L) \in \{P_1(L), \ldots, P_t(L)\}$, $f_{curves}(L) \neq 0$, and the values of $f_{curves}(L)$ satisfy the constraint C. This completes the proof of Theorem 7.6.

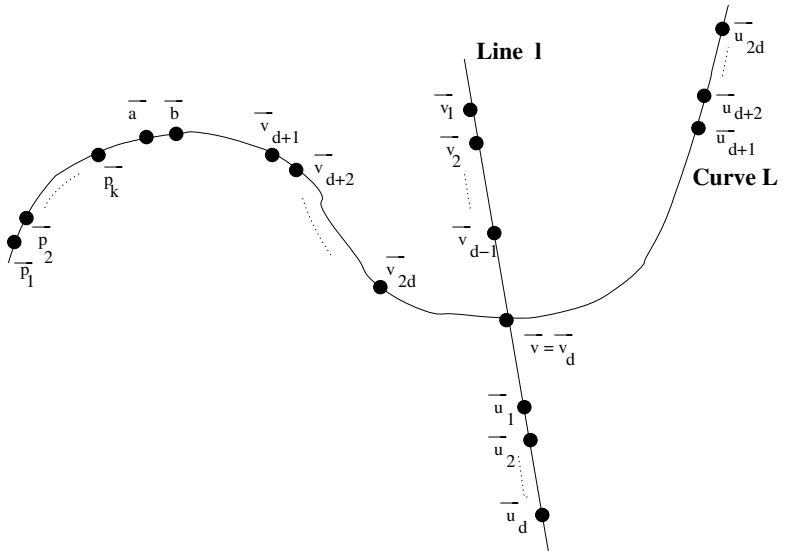


FIG. 3. *Modified outer verifier.*

9. Modified outer verifier. We use a modification of the outer verifier. This modification is necessary for the analysis of the inner verifier to go through.

Remark 9.1. Note that the outer verifier reads $2d$ values. The verifier's checks are all homogeneous linear. This is because all operations including interpolation, evaluating a polynomial at a specific value, and the constraint C are homogeneous linear.

We modify the outer verifier as follows (see Figure 3).

MODIFIED OUTER VERIFIER.

1. Follow steps (1)–(9) as in the outer verifier.
2. Pick additional random points $\overline{u}_1, \ldots, \overline{u}_d$ on line ℓ and random points $\overline{u}_{d+1}, \ldots, \overline{u}_{2d}$ on curve L.
3. Check that the degree $d-1$ univariate polynomial interpolated from values $\{f(\overline{v}_i)\}_{i=1}^{d}$ equals the polynomial interpolated from values $\{f(\overline{u}_i)\}_{i=1}^{d}$ (this actually happens to be a degree $d-2$ polynomial because of earlier steps).
4. Check that the degree $d-1$ univariate polynomial interpolated from values $\{f(\overline{v}_i)\}_{i=d+1}^{2d}$ equals the polynomial interpolated from values $\{f(\overline{u}_i)\}_{i=d+1}^{2d}$.

Remark 9.2. Note that the modified outer verifier reads $4d$ values. The verifier's checks are all homogeneous linear.

Remark 9.3. The purpose of this modification is exactly as explained in sections 1.5, 1.6, and 2.3. This modification adds *redundancy* to the outer verifier. It ensures that we get a squared Fourier coefficient when we do the Fourier analysis of the inner verifier. This is exactly the analogue of Håstad's 3-query PCP that reads 2 queries from the *larger table* so that the Fourier coefficient for the larger table appears in squared form.

10. Inner verifier.

Notation. We will let $\mathbb{F} = 2^l$ so that the field elements are represented by bit strings of length l. Note that addition over $\mathbb{F}$ and multiplication by a constant $\alpha \in \mathbb{F}$ are homogeneous linear operations on these l-bit strings. To keep the description of

the inner verifier conceptually simpler, we abstract out the modified outer verifier as follows (change of notation: redenote $2d$ by d).

ABSTRACT VIEW OF MODIFIED OUTER VERIFIER.

1. Pick an intersecting line-curve pair. Pick points $\{\overline{v}_i\}_{i=1}^{d}$ ($d/2$ of them from the line and $d/2$ from the curve, counting the intersection point as a point on the line; this is not of much relevance at the inner verifier's level).
2. Pick another set of points $\{\overline{u}_i\}_{i=1}^{d}$ (again, $d/2$ from the line and $d/2$ from the curve).
3. Let $T_{ld\times ld}$ be an invertible matrix over $\mathbb{F}_2$ chosen appropriately and let H be a subspace of $\mathbb{F}_2^{ld}$ chosen appropriately (see the remark below on how T, H are chosen).
4. Read the values $f(\overline{v}_1), \ldots, f(\overline{v}_d)$ and values $f(\overline{u}_1), \ldots, f(\overline{u}_d)$. Let
$$\begin{aligned} x &= f(\overline{v}_1) \circ f(\overline{v}_2) \circ \cdots \circ f(\overline{v}_d), \\ y &= f(\overline{u}_1) \circ f(\overline{u}_2) \circ \cdots \circ f(\overline{u}_d) \end{aligned} \tag{23}$$
($\circ$ denotes concatenation of strings).
5. Accept if and only if
$$x \neq 0, \quad x = Ty, \qquad \text{and} \qquad h \cdot x = 0 \ \forall\ h \in H \quad (\text{i.e., } x \perp H). \tag{24}$$

Remark 10.1.

1. The condition $x = Ty$ abstracts out the modified outer verifier's check that the two polynomials interpolated from values $\{f(\overline{v}_i)\}_{i=1}^{d/2}$ and $\{f(\overline{v}_i)\}_{i=d/2+1}^{d}$ are equal to the polynomials interpolated from values $\{f(\overline{u}_i)\}_{i=1}^{d/2}$ and $\{f(\overline{u}_i)\}_{i=d/2+1}^{d}$, respectively. This can be checked by multiplying one set of values by an appropriate invertible matrix and equating the result with the second set of values.
2. The condition $h \cdot x = 0\ \forall\ h \in H$ abstracts out the check that the set of values $\{f(\overline{v}_i)\}_{i=1}^{d}$ satisfy all the homogeneous constraints of the (modified) outer verifier.
3. The condition $x \neq 0$ says that Not-All-Zero$(f(\overline{v}_i), 1 \leq i \leq d)$, as in the (modified) outer verifier.

Now we are ready to describe the inner verifier. The inner verifier expects, for every point $\overline{v} \in \mathbb{F}^m$, the Hadamard code of the string $f(\overline{v}) \in \{0,1\}^l$ (see Appendix A.3 for an overview of Hadamard codes). The action of the inner verifier is as follows (remember that the following action is carried out after randomly picking a constraint $C(\{\overline{p}_i\}_{i=1}^{k})$ of the HomAlgCSP instance):

INNER VERIFIER.

1. Pick an intersecting line-curve pair, points $\{\overline{v}_i\}_{i=1}^{d}$, and points $\{\overline{u}_i\}_{i=1}^{d}$, exactly as in the modified outer verifier.
2. Let $T_{ld\times ld}$ be an invertible matrix over $\mathbb{F}_2$ chosen appropriately and let H be a subspace of $\mathbb{F}_2^{ld}$ chosen appropriately.
3. Let $A_1, A_2, \ldots, A_d$ be the (supposed) Hadamard codes of $f(\overline{v}_1), \ldots, f(\overline{v}_d)$, respectively. Let $B_1, \ldots, B_d$ be the (supposed) Hadamard codes of $f(\overline{u}_1), \ldots, f(\overline{u}_d)$, respectively.
4. Pick a random string $z \in \mathbb{F}_2^{ld}$ and a random $h \in H$. Write
$$\begin{aligned} z &= z_1 \circ z_2 \circ \cdots \circ z_d, \\ h &= h_1 \circ h_2 \circ \cdots \circ h_d, \\ zT &= w_1 \circ w_2 \circ \cdots \circ w_d. \end{aligned}$$

5. Accept if and only if

$$\oplus_{i=1}^{d} A_i(z_i \oplus h_i) = \oplus_{j=1}^{d} B_j(w_j).$$

10.1. Sanity check. Let us see why the test makes sense. Suppose each A_i is indeed a Hadamard code of $f(\overline{v}_i)$ and each B_j is indeed a Hadamard code of $f(\overline{u}_j)$. Therefore

$$A_i(z_i \oplus h_i) = (z_i \oplus h_i) \cdot f(\overline{v}_i) \quad \text{and} \quad B_j(w_j) = w_j \cdot f(\overline{u}_j).$$

Moreover assume that x and y are as in (23), and that they satisfy $x = Ty$ and $h \cdot x = 0 \ \forall \ h \in H$. Now we will show that the inner verifier's test accepts. Indeed,

$$\begin{aligned}
\oplus_{i=1}^{d} A_i(z_i \oplus h_i) &= \oplus_{i=1}^{d} (z_i \oplus h_i) \cdot f(\overline{v}_i) \\
&= (z \oplus h) \cdot (f(\overline{v}_1) \circ f(\overline{v}_2) \circ \cdots \circ f(\overline{v}_d)) \\
&= (z \oplus h) \cdot x \\
&= z \cdot x \oplus h \cdot x \\
&= z \cdot Ty \oplus 0 \\
&= (zT) \cdot y \\
&= (w_1 \circ w_2 \circ \cdots \circ w_d) \cdot (f(\overline{u}_1) \circ f(\overline{u}_2) \circ \cdots \circ f(\overline{u}_d)) \\
&= \oplus_{j=1}^{d} w_j \cdot f(\overline{u}_j) \\
&= \oplus_{j=1}^{d} B_j(w_j).
\end{aligned}$$

10.2. Proof of Theorem 1.9. Note that the inner verifier's test reads $2d$ bits in total. Let Π be the proof given to the inner verifier. Let us first prove that every query of the inner verifier is uniform over Π.

Uniformity of queries. Note that any particular query has the following distribution: pick a point $\overline{v} \in \mathbb{F}^m$ at random and then pick a random bit from the supposed Hadamard code of $f(\overline{v})$. Thus the query is uniformly distributed over proof Π.

Now the following two lemmas are sufficient to prove Theorem 1.9. These are the completeness and the soundness lemmas, respectively.

LEMMA 10.2. *Suppose that the instance $\mathcal{A}$ of HomAlgCSP has optimum $OPT(\mathcal{A})$. Let f be the polynomial that achieves this optimum and let a proof Π be constructed by taking the Hadamard code for every value of f. Let Π_0 be the set of 0-bits in the proof (this constitutes half of the bits in the whole proof). Let Q be the set of $2d$ query bits by the inner verifier. Then*

$$\Pr_Q[Q \subseteq \Pi_0] \geq OPT(\mathcal{A}) \cdot \frac{1}{2^{2d-1}}.$$

The probability is taken over the random test of the inner verifier.

LEMMA 10.3. *If Π_* is any set of half the bits from Π, and*

$$\left| \Pr_Q[Q \subseteq \Pi_*] - \frac{1}{2^{2d}} \right| \geq \delta,$$

then the instance $\mathcal{A}$ of HomAlgCSP has $OPT(\mathcal{A}) \geq \delta^{C''}$, where C'' is an absolute constant.

Let us see how these lemmas prove Theorem 1.9. Let $N = n^{CK}$ be the instance of the MDC given by Theorem 1.5. Theorem 3.4 gives an instance $\mathcal{A}$ of HomAlgCSP with parameters ϵ^*, the degree $d^* = 100/\epsilon^*$, $m = N^{\epsilon^*} = N^{100/d^*}$, and $k = 21$. We have

- $|\mathbb{F}|^m \le (N^2)^{N^{100/d^*}} \le 2^{N^{200/d^*}}$;
- $OPT(\mathcal{A}) \ge 1 - 1/2^K$ in the completeness case and $OPT(\mathcal{A}) \le 1/2^{2^K}$ in the soundness case.
- the inner verifier makes $2d$ queries, where $d = 2((k+2)d^* + 1) \le 50d^*$.

Soundness/quasi-randomness. We show that if Π_* is any set of half the bits from proof Π, then

$$\left| \Pr_Q[Q \subseteq \Pi_*] - \frac{1}{2^{2d}} \right| \le \frac{1}{2^{40d}}.$$

If on the contrary this is not true, then Lemma 10.3 implies that

$$OPT(\mathcal{A}) \ge 1/2^{40C''d} \ge 1/2^{2000C''d^*} > 1/2^{2^K},$$

where we choose $d^* = 2^K/(3000C'')$. This is a contradiction.

Now note that the size of proof Π is essentially $|\mathbb{F}|^m$ and is at most

$$|\mathbb{F}|^m \le 2^{N^{200/d^*}} \le 2^{n^{200CK \cdot 3000C''/2^K}} \le 2^{n^\epsilon}.$$

We can make ϵ arbitrarily small by choosing K large enough. Note that $\epsilon = \Omega(K/2^K)$, whereas $d = O(2^K)$. Hence $d = O(1/\epsilon \log(1/\epsilon))$.

Completeness. Completeness follows directly from Lemma 10.2 by noting that $OPT(\mathcal{A}) \ge 1 - 1/2^K = 1 - O(1/d)$.

Remark 10.4. We view the parameters $K, 1/\epsilon$, and d as constants, but "growing." The constants in the $\Omega()$ and $O()$ notation are absolute constants.

10.3. Proofs of Lemmas 10.2 and 10.3. We will prove both Lemmas 10.2 and 10.3 simultaneously. Let Π be a proof and let Π_* be a set of half the bits in the proof. Imagine setting bits in Π_* to 1 and the rest to 0. This defines the tables A_i, B_j in the proof. Let us estimate the probability that for a random test of the inner verifier, all the queries fall in Π_*, i.e.,

$$\Pr_Q[Q \subseteq \Pi_*].$$

This probability can be arithmetized as

$$E_{\substack{C,\ell,L,\overline{v_1},\ldots,\overline{v_d},\\ \overline{u_1},\ldots,\overline{u_d},z,h}} \left[\prod_{i=1}^{d} A_i(z_i \oplus h_i) \prod_{j=1}^{d} B_j(w_j) \right].$$

Here C is the choice of a random constraint of the HomAlgCSP instance. The rest of the choices are as in the inner verifier's action. Note that the bits $A_i(z_i \oplus h_i)$ and $B_j(w_j)$ are exactly the query bits. Thus we count a test only if each of the query bits falls in Π_*. We analyze this expression.

Fix everything else and consider only expectation over z, h. We drop the expectation sign for convenience. Plugging in Fourier expansions of A_i's and B_j's we get

$$\sum_{\alpha_1,\ldots,\alpha_d,\beta_1,\ldots,\beta_d} \left[\prod_{i=1}^{d} \widehat{A}_{i,\alpha_i} \prod_{j=1}^{d} \widehat{B}_{j,\beta_j} \cdot \prod_{i=1}^{d} \chi_{\alpha_i}(z_i \oplus h_i) \cdot \prod_{j=1}^{d} \chi_{\beta_j}(w_j) \right],$$

which simplifies to

$$\sum_{\substack{\alpha=\alpha_1\circ\alpha_2\ldots\circ\alpha_d,\\ \beta=\beta_1\circ\beta_2\ldots\circ\beta_d}} \left[\prod_{i=1}^{d}\widehat{A}_{i,\alpha_i}\prod_{j=1}^{d}\widehat{B}_{j,\beta_j}\cdot(-1)^{\alpha\cdot z\oplus\beta\cdot w}\cdot(-1)^{\alpha\cdot h}\right].$$

Now note that h is chosen randomly from the linear subspace H and thus the expectation over h is zero unless $\alpha\perp H$ (reason: a vector α is either orthogonal to all the vectors in H or orthogonal to exactly half the vectors in H). Moreover,

$$\begin{aligned} z\cdot\alpha\oplus w\cdot\beta &= z\cdot\alpha\oplus zT\cdot\beta\\ &= z\cdot(\alpha\oplus T\beta). \end{aligned}$$

Therefore, the expectation over z is zero unless $\alpha = T\beta$. Thus we get

$$\sum_{\substack{\alpha=\alpha_1\circ\alpha_2\ldots\circ\alpha_d,\\ \alpha\perp H,\ \beta=T^{-1}\alpha}} \left[\prod_{i=1}^{d}\widehat{A}_{i,\alpha_i}\prod_{j=1}^{d}\widehat{B}_{j,\beta_j}\right]. \tag{25}$$

10.4. Proof of Lemma 10.2. Let f be the solution for the HomAlgCSP instance with optimum $OPT(\mathcal{A})$. Thus for any $\overline{v}\in\mathbb{F}^m$, the corresponding Hadamard code $A_{\overline{v}}$ is precisely (see Fact A.7)

$$\frac{1}{2}-\frac{1}{2}\chi_{f(\overline{v})}.$$

Thus the table $A_{\overline{v}}$ has exactly two nonzero Fourier coefficients, $\widehat{A}_{\overline{v},0}=\frac{1}{2}$ and $\widehat{A}_{\overline{v},f(\overline{v})}=-\frac{1}{2}$.

Let us prove that

$$\Pr_Q[Q\subseteq\Pi_0]\geq OPT(\mathcal{A})\cdot\frac{1}{2^{2d-1}}.$$

Set $\Pi_*=\Pi_0$. As before, we imagine the bits in Π_* as set to 1 and the rest of the bits as set to 0. This corresponds to flipping the bits in the Hadamard codes. Hence the resulting tables A_i, B_j have Fourier representation (see Fact A.7)

$$A_i=\frac{1}{2}+\frac{1}{2}\chi_{f(\overline{v}_i)},\qquad B_j=\frac{1}{2}+\frac{1}{2}\chi_{f(\overline{u}_j)}.$$

Note that f satisfies $OPT(\mathcal{A})$ fraction of the constraints of the HomAlgCSP. For any satisfied constraint, we get a contribution of $\frac{1}{2^{2d}}$ in term (25) in each of these cases:

$$\begin{aligned} &\alpha_1=\alpha_2=\cdots=\alpha_d=\beta_1=\cdots=\beta_d=0,\\ &\alpha_i=f(\overline{v}_i),\ \beta_j=f(\overline{u}_j)\text{ for }i,j=1,2,\ldots,d. \end{aligned} \tag{26}$$

All other terms are nonnegative. Thus

$$\Pr_Q[Q\subseteq\Pi_0]\geq OPT(\mathcal{A})\cdot\left(\frac{1}{2^{2d}}+\frac{1}{2^{2d}}\right)=OPT(\mathcal{A})\cdot\frac{1}{2^{2d-1}}.$$

10.5. Proof of Lemma 10.3. Assume that the quantity in (25) is δ away from $\frac{1}{2^{2d}}$. First, we show that the contribution of the term with $\alpha=0$ is essentially $\frac{1}{2^{2d}}$. This implies that the contribution of terms with nonzero α is δ. Using this, we define a strategy for the outer verifier with success probability δ^2, which in turn gives a solution to the HomAlgCSP instance with value $\delta^{O(1)}$.

Contribution of the term with $\alpha = 0$. This contribution is

$$E_{\substack{\ell,L,\overline{v}_1,\ldots,\overline{v}_d,\\ \overline{u}_1,\ldots,\overline{u}_d}}\left[\prod_{i=1}^{d}\widehat{A}_{i,0}\widehat{B}_{i,0}\right],$$

where the expectation is taken over all line-curve pairs and choice of d random points each on the line and the curve.

Note that the zero Fourier coefficient equals the fraction of 1's in the table. Overall the proof contains half 1's.

We use the mixing property of curves and lines (see Appendix A.4). Therefore for a random choice of intersecting curve L and line ℓ, except with probability $O(1/|\mathbb{F}|^{1/3})$, we have

$$E_{\overline{v}\in L}[\widehat{A}_{\overline{v},0}] \approx \frac{1}{2}, \qquad E_{\overline{v}\in \ell}[\widehat{A}_{\overline{v},0}] \approx \frac{1}{2},$$

where the error in the approximation is bounded by $O(1/|\mathbb{F}|^{1/3})$. Once the curve and the line are chosen, $\overline{v}_1,\ldots,\overline{v}_d,\overline{u}_1,\ldots,\overline{u}_d$ are $2d$ random points on the curve and the line. Thus,

$$E_{\substack{\overline{v}_1,\ldots,\overline{v}_d,\\ \overline{u}_1,\ldots,\overline{u}_d}}\left[\prod_{i=1}^{d}\widehat{A}_{i,0}\widehat{B}_{i,0}\right] = \left(E_{\overline{v}\in L}[\widehat{A}_{\overline{v},0}]\right)^d \cdot \left(E_{\overline{v}\in \ell}[\widehat{A}_{\overline{v},0}]\right)^d \approx \frac{1}{2^{2d}}$$

up to error $O(1/|\mathbb{F}|^{1/3})$. This proves that the contribution of the term with $\alpha = 0$ is essentially $\frac{1}{2^{2d}}$.

Contribution of the terms with nonzero α. Using Cauchy–Schwarz, contribution of the terms with nonzero α in (25) is at most

$$\sqrt{E\left[\sum_{\alpha\neq 0,\alpha\perp H}\prod_{i=1}^{d}\widehat{A}_{i,\alpha_i}^2\right]}\sqrt{E\left[\sum_{\alpha\neq 0,\alpha\perp H,\beta=T^{-1}\alpha}\prod_{j=1}^{d}\widehat{B}_{j,\beta_j}^2\right]},$$

which is at most

$$\sqrt{E\left[\sum_{\alpha\neq 0,\alpha\perp H}\prod_{i=1}^{d}\widehat{A}_{i,\alpha_i}^2\right]}.$$

Observe that the expectation naturally gives a strategy for the outer verifier! For any point $\overline{v}\in\mathbb{F}^m$, define $f(\overline{v}) = \alpha_i$ with probability $\widehat{A}^2_{\overline{v},\alpha_i}$. This gives a (probabilistic) construction of a table of values $f:\mathbb{F}^m \mapsto \mathbb{F}$. The condition $\alpha \neq 0$ ensures that not all values read by the outer verifier are zero. The condition $\alpha\perp H$ ensures that the homogeneous linear constraint checked by the outer verifier is satisfied. These are exactly the two conditions when the outer verifier accepts. Note how the redundancy introduced by the modified outer verifier enabled us to apply Cauchy–Schwarz.

Thus, if the contribution of the terms with nonzero α is δ, then we can construct a table $f:\mathbb{F}^m \mapsto \mathbb{F}$ which the outer verifier accepts with probability δ^2. This probability is the average of the outer verifier's success probability over the choice of the HomAlgCSP constraint. Thus for at least $\delta^2/2$ fraction of the HomAlgCSP constraints, the outer verifier's success probability is $\delta^2/4$. Apply Theorem 7.6 with $\delta^2/4$ in place of δ.

Let $P_1, P_2, \ldots, P_t$ be the nonzero polynomials as in Theorem 7.6. It follows that for at least $\delta^2/4$ fraction of the HomAlgCSP constraints, at least one of the polynomials P_j satisfies this constraint. Thus there is one nonzero polynomial P_{j_0} that satisfies $(\delta^2/4)/t \geq \delta^{C''}$ fraction of the HomAlgCSP constraints. Note that $t \leq 2c_3/(\delta^2/8)^{c_2}$. This proves Lemma 10.3.

11. Conclusion. We have ruled out PTAS for three notorious problems: graph min-bisection, dense k-subgraph, and bipartite clique. Our results hold under the assumption that NP does not have subexponential time algorithms. This could be a new avenue for showing inapproximability results instead of the more common assumption that NP does not have polynomial time or quasi-polynomial time algorithms.

We have made several new contributions in terms of conceptual ideas and techniques. We introduced the notion of quasi-random PCPs that should be very useful for proving inapproximability results for graph theoretic problems, e.g., graph expansion. Some of the ideas in this paper have roots in [30]. To the best of our knowledge, it was the first paper that employed a combination of the algebraic techniques (those used to prove the PCP theorem) and more recent Fourier analysis methods. All recent results, on the other hand, use the PCP theorem as a black-box and then use parallel repetition [33] and Fourier analysis. In this paper, we make even heavier use of algebraic techniques, using generalizations and variants of the polynomial encoding method, LDT, and the sum-check protocol. This new direction in inapproximability theory looks very promising.

The connection between MDC and the graph theoretic problems is very surprising. It could be possible to prove inapproximability results assuming sufficiently strong hardness results for the shortest vector problem, the analogue of MDC for integer lattices.

Appendix A. Useful facts and tools.

A.1. Schwartz lemma. We use the following fact (called the Schwartz lemma) again and again.

FACT A.1. *Let $f(x_1, \ldots, x_m)$ be a degree d polynomial in m variables over field $\mathbb{F}$. Assume that $x_1, \ldots, x_m$ are chosen at random from $\mathbb{F}$. If*

$$\Pr[f(x_1, \ldots, x_m) = 0] \geq \frac{2d}{|\mathbb{F}|},$$

then f must identically be zero. Consequently, for two degree d polynomials f and g in m variables, if $f(x_1, \ldots, x_m) = g(x_1, \ldots, x_m)$ with probability $\frac{2d}{|\mathbb{F}|}$, then they must be identical polynomials.

A.2. Addition and multiplication over fields of characteristic 2. Here we summarize simple facts about addition and multiplication over the field $\mathbb{F} = \mathbb{F}_{2^l}$. Elements of $\mathbb{F}$ can be viewed as l-bit strings, i.e., as vectors in $\mathbb{F}_2^l$. For $x \in \mathbb{F}$, let x^* denote the corresponding vector in $\mathbb{F}_2^l$.

FACT A.2.

$$(x + y)^* = x^* \oplus y^* \ \forall\ x, y \in \mathbb{F},$$

where on the LHS we have addition in the field and on the RHS we have addition in $\mathbb{F}_2^l$.

FACT A.3. *If $\gamma \in \mathbb{F}$, then there exists an $l \times l$ matrix A_γ with entries in $\mathbb{F}_2$ such that*

$$(\gamma x)^* = A_\gamma x^* \quad \forall\ x \in \mathbb{F},$$

where on the LHS we have multiplication in the field and on the RHS we have multiplication of a matrix and a vector.

A.3. Hadamard codes and Fourier analysis.

DEFINITION A.4. *Let $x \in \mathbb{F}_2^l$ be an l-bit string. Its Hadamard code A_x is a 2^l-bit string whose coordinates are indexed by $s \in \mathbb{F}_2^l$ and*

$$A_x(s) = s \cdot x \quad \forall s \in \mathbb{F}_2^l,$$

where $s \cdot x$ denotes the inner product over $\mathbb{F}_2^l$.

DEFINITION A.5. *For $\alpha \in \mathbb{F}_2^l$, let $\chi_\alpha : \mathbb{F}_2^l \mapsto \{-1, 1\}$ be a real-valued function defined as*

$$\chi_\alpha(x) = (-1)^{\alpha \cdot x}.$$

FACT A.6. *Any real-valued function $A : \mathbb{F}_2^l \mapsto \mathbb{R}$ can be uniquely written as*

$$A = \sum_{\alpha \in \mathbb{F}_2^l} \widehat{A}_\alpha \ \chi_\alpha,$$

where $\widehat{A}_\alpha$ are real numbers called Fourier coefficients. We have Parseval's identity:

$$\sum_{\alpha \in \mathbb{F}_2^l} \widehat{A}_\alpha^2 = \|A\|_2^2 = \frac{1}{2^l} \sum_{s \in \mathbb{F}_2^l} A(s)^2.$$

FACT A.7. *Let A_x be Hadamard code of x. It can be regarded as a real-valued function $A_x : \mathbb{F}_2^l \mapsto \{0, 1\}$. Then its Fourier representation is*

$$A_x = \frac{1}{2} - \frac{1}{2}\chi_x.$$

The first term is $\frac{1}{2}\chi_0$, where $\chi_0 \equiv 1$. If A'_x is a function obtained by flipping values of A_x (i.e., 0 flipped to 1, and vice versa), then the Fourier representation of A'_x is

$$A'_x = \frac{1}{2} + \frac{1}{2}\chi_x.$$

A.4. Mixing property of lines and curves.

LEMMA A.8. *Let $\psi : \mathbb{F}^m \mapsto [0, 1]$ be any function with $E_{\overline{u} \in \mathbb{F}^m}[\psi(\overline{u})] = \frac{1}{2}$. Let ℓ be a line chosen at random. Then w.h.p. the average of ψ-values of points on ℓ is very close to $\frac{1}{2}$. To be precise,*

$$\Pr_\ell \left[\left| E_{\overline{u} \in \ell}[\psi(\overline{u})] - \frac{1}{2} \right| \geq \frac{1}{|\mathbb{F}|^{1/3}} \right] \leq \frac{1}{|\mathbb{F}|^{1/3}}.$$

Proof. A random line ℓ is chosen by picking two random points $\overline{a}, \overline{b} \in \mathbb{F}^m$ and letting

$$\ell := \{\overline{a} + t\overline{b} \mid t \in \mathbb{F}\}.$$

Note that for any $t \in \mathbb{F}$, the point $\overline{a} + t\overline{b}$ is uniformly distributed over $\mathbb{F}^m$ and, moreover, for $t_1 \neq t_2 \in \mathbb{F}$, the points $\overline{a} + t_1\overline{b}$ and $\overline{a} + t_2\overline{b}$ are pairwise independent. Hence, the random variables $\{\psi(\overline{a} + t\overline{b})| \ t \in \mathbb{F}\}$ each have expectation $\frac{1}{2}$ and are

pairwise independent. Applying Chebyshev's inequality (see below), we see that the random variable $E_t[\psi(\overline{a}+t\overline{b})]$ (which is the same as $E_{\overline{u}\in\ell}[\psi(\overline{u})]$) is sharply concentrated around $\frac{1}{2}$. □

LEMMA A.9 (Chebyshev's inequality). *Let $X_1, X_2, \ldots, X_n$ be pairwise independent random variables with range $[0,1]$, each with expectation p. Let $X = \frac{1}{n}\sum_{i=1}^{n} X_i$. Then*

$$\Pr[|X-p| \geq t] \leq \frac{var(X)}{t^2} = \frac{\frac{1}{n^2}\sum_{i=1}^{n} var(X_i)}{t^2} \leq \frac{1}{nt^2}.$$

A similar result holds for curves (the proof is similar and omitted). Let $\{\overline{p}_i\}_{i=1}^{k}$ and $\overline{v}$ be points in $\mathbb{F}^m$. Let $\{t_1, \ldots, t_{k+3}, t\}$ be distinct field elements. A random curve L passing through $\overline{v}$ is obtained as follows:

- Pick random points $\overline{a}, \overline{b} \in \mathbb{F}^m$.
- Let L be the unique degree $k+2$ curve such that

$$L(t_i) = \overline{p}_i,\ 1 \leq i \leq k, \quad L(t_{k+1}) = \overline{a},\ L(t_{k+2}) = \overline{b}, \quad L(t) = \overline{v}.$$

LEMMA A.10. *Let $\psi : \mathbb{F}^m \mapsto [0,1]$ be any function with $E_{\overline{u}\in\mathbb{F}^m}[\psi(\overline{u})] = \frac{1}{2}$. Let L be a curve chosen at random as above. Then w.h.p. the average of ψ-values of points on L is very close to $\frac{1}{2}$. To be precise,*

$$\Pr_L\left[\left|E_{\overline{u}\in L}[\psi(\overline{u})] - \frac{1}{2}\right| \geq \frac{1}{|\mathbb{F}|^{1/3}}\right] \leq \frac{1}{|\mathbb{F}|^{1/3}}.$$

Appendix B. Proof of Theorem 1.5. We start with a result of Dumer, Micciancio, and Sudan [15] that it is NP-hard to approximate the MDC within any constant factor (say 5), and the result holds on binary asymptotically good codes. Formally, we have the following.

THEOREM B.1 (see [15]). *There is an absolute constant $\zeta > 0$ and a reduction from SAT to MDC where the following hold:*

1. *The MDC instance is over $\mathbb{F}_2$, i.e., the code is binary.*
2. *(Completeness.) The YES instance of SAT is mapped to an MDC instance $\mathbf{A}$ with $OPT(\mathbf{A}) \leq \zeta$.*
3. *(Soundness.) The NO instance of SAT is mapped to an MDC instance $\mathbf{A}$ with $OPT(\mathbf{A}) \geq 5\zeta$.*
4. *The reduction runs in time n^C, where n is the size of the SAT instance and C is an absolute constant.*

The hardness of MDC can be boosted via the tensor product, giving the following theorem.

THEOREM B.2 (see [15]). *There is an absolute constant $\zeta > 0$ such that for every integer K, there is a reduction from SAT to MDC satisfying the following:*

1. *The MDC instance is over $\mathbb{F}_2$, i.e., the code is binary. It has N rows and n' columns.*
2. *(Completeness.) The YES instance of SAT is mapped to an MDC instance $\mathbf{A}$ with $OPT(\mathbf{A}) \leq \zeta^K$.*
3. *(Soundness.) The NO instance of SAT is mapped to an MDC instance $\mathbf{A}$ with $OPT(\mathbf{A}) \geq 5^K\zeta^K$.*
4. *The reduction runs in time n^{CK}, where n is the size of SAT instance and C is an absolute constant. In particular, $N \leq n^{CK}$.*

Let $\mathbf{A} = \{a_{ij}\}$ be an instance of MDC and let n, n', N, K, C, ζ be parameters as in Theorem B.2. We interpret the rows of $\mathbf{A}$ as linear forms:

$$L_i = \sum_{j=1}^{n'} a_{ij} z_j, \quad 1 \le i \le N.$$

Thus the goal of the MDC problem is to find a nonzero solution $\mathbf{z} = (z_1, z_2, \ldots, z_{n'})$ so as to minimize the number of nonzero linear forms. By taking an extension field of $\mathbb{F}_2$, we can assume that Theorem B.2 holds for codes over a field $\mathbb{F}$ of characteristic 2 with size $\approx N$.

Now the idea is the following. Create a new instance $\mathbf{A}'$ of MDC whose linear forms are linear combinations of the linear forms $\{L_i\}_{i=1}^N$. Thus a typical linear form looks like this:

$$\sum_{s=1}^{t} \alpha_s L_{i_s}, \quad \text{where } \alpha_s \in \mathbb{F},\ i_s \in [N].$$

There are $|\mathbb{F}|^t N^t$ such linear forms. However, we want to take only $O(N)$ of them, and we achieve this using walks on expanders. Choose $t = 1/(\zeta^K 2^K)$.

Let G be a degree D expander on N vertices with second largest eigenvalue λ. It is possible to explicitly (and efficiently) construct such expanders with $\lambda \le D^{9/10}$ (in fact one can get $\lambda \approx 2\sqrt{D}$ using the Ramanujan graphs [31]). Choose $D = (\frac{4}{5^K \zeta^K})^{10}$ so that $\lambda/D \le 5^K \zeta^K/4$. Let $S \subseteq \mathbb{F}$ with $|S| = 2^{2 \cdot 2^K}$.

Let the new instance $\mathbf{A}'$ of MDC consist of these linear forms:

$$\sum_{s=1}^{t} \alpha_s L_{i_s}, \quad \text{where } \alpha_s \in S, \text{ and } [i_1, i_2, \ldots, i_t] \text{ is a walk on graph } G. \tag{27}$$

The number of such linear forms is $|S|^t N \cdot D^{t-1}$, which is $O(N)$ since ζ, K are constants. Now we prove the completeness and soundness properties.

Completeness. If the SAT instance is a YES instance, we know that $OPT(\mathbf{A}) \le \zeta^K$. Thus there is a nonzero solution $\mathbf{z}$ such that at most ζ^K fraction of the linear forms $\{L_i\}_{i=1}^N$ are nonzero.

For a random walk $[i_1, i_2, \ldots, i_t]$ on G, the probability that there exists $L_{i_s} \ne 0$ is at most $t\zeta^K$. This implies that the fraction of linear forms in (27) that are nonzero is at most $t\zeta^K$. Thus $OPT(\mathbf{A}') \le t\zeta^K = \frac{1}{2^K}$.

Soundness. If the SAT instance is a NO instance, we know that $OPT(\mathbf{A}) \ge 5^K \zeta^K$. Fix any nonzero $\mathbf{z}$. We know that at least $5^K \zeta^K$ fraction of linear forms $\{L_i\}_{i=1}^N$ are nonzero. Let

$$W := \{i \mid i \in [N],\ L_i \ne 0\}$$

so that $\mu(W) = |W|/N \ge 5^K \zeta^K$. Using [32, section 15], we can bound the probability that all vertices of the random walk $[i_1, i_2, \ldots, i_t]$ fall outside W.

$$\begin{aligned}
\Pr[i_1 \in \overline{W} \wedge i_2 \in \overline{W} \ldots \wedge i_t \in \overline{W}] &\le \left(\sqrt{\mu(\overline{W})} + \frac{\lambda}{D}\right)^t \\
&\le \left(\sqrt{1 - 5^K \zeta^K} + 5^K \zeta^K/4\right)^t \\
&\le \left(1 - 5^K \zeta^K/4\right)^{1/(\zeta^K 2^K)} \\
&\le e^{-(5/2)^K/4} \le 2^{-2 \cdot 2^K}.
\end{aligned}$$

Therefore at least a fraction $1 - 2^{-2 \cdot 2^K}$ of the linear forms in (27) contain a linear form $L_{i_{s^*}} \neq 0$. For any fixed tuple $(L_{i_1}, \ldots, L_{i_t})$ that contains at least one $L_{i_{s^*}} \neq 0$, after choosing the coefficients $\alpha_1, \ldots, \alpha_t \in S$ at random, the probability that (27) is nonzero is at least $1 - \frac{1}{|S|}$. Therefore, the fraction of nonzero linear forms among (27) is at least

$$(1 - 2^{-2 \cdot 2^K}) \cdot \left(1 - \frac{1}{|S|}\right) \geq (1 - 2^{-2 \cdot 2^K}) \cdot (1 - 1/2^{2 \cdot 2^K}) \geq 1 - 1/2^{2^K}.$$

Thus $OPT(\mathbf{A}') \geq 1 - \frac{1}{2^{2^K}}$. This proves Theorem 1.5.

Appendix C. Reductions from quasi-random PCP to graph min-bisection, dense *k*-subgraph, and bipartite clique. In this section, we prove that the PCP construction given by Theorem 1.9 implies inapproximability results for graph min-bisection, dense k-subgraph, and bipartite clique. This proves Theorem 1.1. The reductions given here are similar to the reductions given by Feige [17] (correctness of his reduction, however, relies on his hypothesis about hardness of random 3SAT).

C.1. Graph min-bisection. Let N be the proof size and M the total number of tests of the PCP verifier in Theorem 1.9. Both N and M are bounded by $2^{O(n^\epsilon)}$. Let d be the integer as in that theorem.

Construct a graph G as follows: it is disjoint union of a bipartite graph and a clique. The clique has size $(1 - \frac{3}{2}\frac{1}{2^{d-1}})dM^2 - \frac{N}{2}$. The bipartite graph is constructed as follows: The LHS has N vertices corresponding to the N bits in the PCP proof. The RHS has M clusters, one for every test of the PCP verifier. Every cluster is a clique of size dM. One vertex in every cluster is designated as a *connection vertex*. The bipartite edges are only between the LHS vertices and the connection vertices on RHS. There is an edge between a vertex (= bit) on the LHS and a connection vertex (= PCP test) on the RHS if and only if the bit is accessed by the PCP test. This completes the description of the graph G. Note that every connection vertex has d neighbors on the LHS since this is a d-query PCP. Since every query is uniformly distributed, every bit in the proof appears in the same number of tests. Therefore, the LHS vertices all have the same degree, say Δ, so that $\Delta N = dM$.

Completeness. We claim that the graph G can be partitioned into two equal parts so that the number of crossing edges is at most $(\frac{1}{2} - \frac{3}{4}\frac{1}{2^{d-1}})dM$.

Theorem 1.9 gives a set Π_0 of half the bits from the proof such that at least $(1 - O(1/d))\frac{1}{2^{d-1}} \geq \frac{3}{4}\frac{1}{2^{d-1}}$ fraction of the tests access all the d queries from Π_0. Partition the graph G into two equal parts as follows: One part consists of the following:

1. $\frac{N}{2}$ vertices from the LHS corresponding to bits in Π_0. Abusing notation, denote the set of these vertices also by Π_0.
2. $\frac{3}{4}\frac{1}{2^{d-1}}M$ clusters from the RHS corresponding to the tests that access all d queries from Π_0. Call the set of these tests (or connection vertices) $\mathcal{T}_0$.
3. The clique of size $(1 - \frac{3}{2}\frac{1}{2^{d-1}})dM^2 - \frac{N}{2}$.

The second part consists of all the remaining $(1 - \frac{3}{4}\frac{1}{2^{d-1}})M$ clusters on the RHS corresponding to the tests in $\overline{\mathcal{T}}_0$. The only crossing edges are between Π_0 and $\overline{\mathcal{T}}_0$. The total number of edges incident on Π_0 is $\Delta|\Pi_0| = \Delta\frac{N}{2}$. The number of edges between Π_0 and $\mathcal{T}_0$ is $d|\mathcal{T}_0| = d(\frac{3}{4}\frac{1}{2^{d-1}}M)$ since every connection vertex in $\mathcal{T}_0$ has degree d and all its neighbors (queries) are in Π_0. Therefore, the number of crossing edges is at most

$$\Delta\frac{N}{2} - \frac{3}{4}\frac{1}{2^{d-1}}dM = \frac{dM}{2} - \frac{3}{4}\frac{1}{2^{d-1}}dM = \left(\frac{1}{2} - \frac{3}{4}\frac{1}{2^{d-1}}\right)dM.$$

Soundness. We show that for any bisection of G, the number of crossing edges is at least the expression (28). Note that the clusters and the clique have size at least dM, so any bisection that cuts either a cluster or the clique has at least $dM-1$ crossing edges. Hence we need to consider only those partitions for which every cluster and the clique are completely contained inside one of the parts.

This implies that one of the parts consists of the following:

1. $\frac{N}{2}$ vertices from the LHS. Denote the set of these vertices and the set of corresponding bits in the PCP proof by Π_*.
2. $\frac{3}{4}\frac{1}{2^{d-1}}M$ clusters from the RHS. Denote the set of corresponding connection vertices and the set of corresponding PCP tests by $\mathcal{T}_*$.
3. The clique of size $(1-\frac{3}{2}\frac{1}{2^{d-1}})dM^2-\frac{N}{2}$.

By Theorem 1.9, at most $\frac{1}{2^d}+\frac{1}{2^{20d}}$ fraction of the tests have all their queries in Π_*. Hence, the number of edges between Π_* and $\mathcal{T}_*$ is at most

$$\begin{aligned} &d\cdot\left(\frac{1}{2^d}+\frac{1}{2^{20d}}\right)M+(d-1)\cdot\left(\frac{3}{4}\frac{1}{2^{d-1}}-\left(\frac{1}{2^d}+\frac{1}{2^{20d}}\right)\right)M\\ &=dM\left(\frac{3}{4}\frac{1}{2^{d-1}}-\frac{1}{d}\left(\frac{1}{2^{d+1}}-\frac{1}{2^{20d}}\right)\right).\end{aligned}$$

The number of crossing edges is at least equal to the number of edges between Π_* and $\overline{\mathcal{T}}_*$. This is equal to the number of edges incident on Π_* (which is $\Delta\frac{N}{2}=\frac{dM}{2}$) minus the number of edges between Π_* and $\mathcal{T}_*$. Thus the number of crossing edges is at least

$$dM\left(\frac{1}{2}-\frac{3}{4}\frac{1}{2^{d-1}}+\frac{1}{d}\left(\frac{1}{2^{d+1}}-\frac{1}{2^{20d}}\right)\right). \tag{28}$$

Inapproximability factor. Clearly, the inapproximability factor is

$$\frac{\frac{1}{2}-\frac{3}{4}\frac{1}{2^{d-1}}+\frac{1}{d}(\frac{1}{2^{d+1}}-\frac{1}{2^{20d}})}{\frac{1}{2}-\frac{3}{4}\frac{1}{2^{d-1}}}.$$

This is $\approx 1+\frac{2}{d2^{d+1}}=1+\frac{1}{2^{O(1/\epsilon\log(1/\epsilon))}}$ since $d=O(1/\epsilon\log(1/\epsilon))$. Recall that the reduction runs in time $2^{O(n^\epsilon)}$ on a SAT instance of size n.

C.2. Dense *k*-subgraph. Construct a bipartite graph G as follows: The LHS consists of N vertices corresponding to the bits in the PCP proof. The RHS consists of M vertices corresponding to the tests of the PCP. Connect an LHS vertex (= bit) to an RHS vertex (= PCP test) if and only if the bit is accessed by the test. Assume that $N=\frac{M}{2^{5d}}$ (this can be achieved by duplicating vertices, since the dense subgraph must take either *all* copies of a vertex or *none*). Set the size of the subgraph to $k=\left(\frac{1}{2^{5d+1}}+\frac{3}{4}\frac{1}{2^{d-1}}\right)M$.

Completeness. By Theorem 1.9, there is a set Π_0 of $\frac{N}{2}$ bits such that at least $(1-O(1/d))\frac{1}{2^{d-1}}\geq\frac{3}{4}\frac{1}{2^{d-1}}$ fraction of the tests have all their d queries in Π_0. This gives an induced subgraph of G on

$$\frac{N}{2}+\frac{3}{4}\frac{1}{2^{d-1}}M=\left(\frac{1}{2^{5d+1}}+\frac{3}{4}\frac{1}{2^{d-1}}\right)M$$

vertices that has at least $(\frac{3}{4}\frac{1}{2^{d-1}})dM$ edges.

Soundness. Take any induced subgraph of G on $(\frac{1}{2^{5d+1}} + \frac{3}{4}\frac{1}{2^{d-1}})M$ vertices.

Case (i). If the number of LHS vertices is $\geq (\frac{1}{2} + \frac{1}{2^{5d}})N$, then the number of RHS vertices is at most

$$\left(\frac{1}{2^{5d+1}} + \frac{3}{4}\frac{1}{2^{d-1}}\right)M - \left(\frac{1}{2} + \frac{1}{2^{5d}}\right)N = \left(\frac{3}{4}\frac{1}{2^{d-1}} - \frac{1}{2^{10d}}\right)M.$$

Hence, the number of edges in the induced subgraph is at most $(\frac{3}{4}\frac{1}{2^{d-1}} - \frac{1}{2^{10d}})dM$.

Case (ii). So assume that the number of LHS vertices is $\leq (\frac{1}{2} + \frac{1}{2^{5d}})N$ (assume w.l.o.g. that the number is exactly this). Call these vertices Π'_*. The number of RHS vertices is at most $(\frac{1}{2^{5d+1}} + \frac{3}{4}\frac{1}{2^{d-1}})M$.

Now we bound the fraction of tests all of whose queries are in Π'_*. Let Π_* be any subset of Π'_* of size $\frac{N}{2}$. By Theorem 1.9, the fraction of tests with all queries in Π_* is bounded by $\frac{1}{2^d} + \frac{1}{2^{20d}}$. The fraction of tests with at least one query in $\Pi'_* \setminus \Pi_*$ is bounded by $\frac{d}{2^{5d}}$ since the queries are uniformly distributed and the probability that any single query is in $\Pi'_* \setminus \Pi_*$ equals the fractional size of this set which is at most $\frac{1}{2^{5d}}$.

Therefore, the number of tests all of whose queries are in Π'_* is at most $(\frac{1}{2^d} + \frac{1}{2^{20d}} + \frac{d}{2^{5d}})M$. Hence the number of edges in the induced subgraph is at most

$$\begin{aligned}
&d\left(\frac{1}{2^d} + \frac{1}{2^{20d}} + \frac{d}{2^{5d}}\right)M + (d-1)\left(\left(\frac{1}{2^{5d+1}} + \frac{3}{4}\frac{1}{2^{d-1}}\right) - \left(\frac{1}{2^d} + \frac{1}{2^{20d}} + \frac{d}{2^{5d}}\right)\right)M \\
&= dM\left(\left(\frac{3}{4}\frac{1}{2^{d-1}} + \frac{1}{2^{5d+1}}\right) - \frac{1}{d}\left(\frac{1}{2^{5d+1}} + \frac{3}{4}\frac{1}{2^{d-1}}\right) + \frac{1}{d}\left(\frac{1}{2^d} + \frac{1}{2^{20d}} + \frac{d}{2^{5d}}\right)\right) \\
&= dM\left(\frac{3}{4}\frac{1}{2^{d-1}} - \frac{1}{d\cdot 2^{d+1}} + (\text{low order terms})\right).
\end{aligned}$$

In either case, the number of edges in the induced subgraph is at most $(\frac{3}{4}\frac{1}{2^{d-1}} - \frac{1}{2^{10d}})dM$.

Inapproximability factor. Clearly, the inapproximability factor is

$$\frac{\frac{3}{4}\frac{1}{2^{d-1}}}{\frac{3}{4}\frac{1}{2^{d-1}} - \frac{1}{2^{10d}}},$$

which is $\approx 1 + \frac{4\cdot 2^{d-1}}{3\cdot 2^{10d}} = 1 + \frac{1}{2^{O(1/\epsilon\log(1/\epsilon))}}$ since $d = O(1/\epsilon\log(1/\epsilon))$.

C.3. Bipartite clique. Construct a bipartite graph G as follows: The LHS consists of N vertices corresponding to the bits in the PCP proof. The RHS consists of M vertices corresponding to the tests of the PCP. Connect an LHS vertex (= bit) to an RHS vertex (= PCP test) if and only if the bit is *not* accessed by the test. Assume that $\frac{N}{2} = (\frac{3}{4}\frac{1}{2^{d-1}})M$. (This can be achieved by duplicating vertices. Any maximal bipartite clique must take either *all* copies of a vertex or *none*.)

Completeness. We will show that there is a bipartite clique of size $\frac{N}{2}$. Theorem 1.9 shows that the set of 1-bits Π_1 in the proof is such that a fraction $(1 - O(1/d))\frac{1}{2^{d-1}} \geq \frac{3}{4}\frac{1}{2^{d-1}}$ of tests *does not* access any query from Π_1. Let $\mathcal{T}_1$ denote the set of all such tests with $|\mathcal{T}_1| = (\frac{3}{4}\frac{1}{2^{d-1}})M = \frac{N}{2}$. Clearly, Π_1 and $\mathcal{T}_1$ give an $\frac{N}{2}$-sized bipartite clique.

Soundness. We will show that there is no bipartite clique of size $(1 - \frac{1}{2^{5d}})\frac{N}{2}$. Assume on the contrary that there is such a clique and let Π_* be the LHS and let $\mathcal{T}_*$ be the RHS of this clique. By hypothesis,

$$|\Pi_*| = \left(1 - \frac{1}{2^{5d}}\right)\frac{N}{2} \quad \text{and} \quad |\mathcal{T}_*| = \left(1 - \frac{1}{2^{5d}}\right)\frac{N}{2} = \left(1 - \frac{1}{2^{5d}}\right)\frac{3}{4}\frac{1}{2^{d-1}}M.$$

Clearly, $|\overline{\Pi}_*| = (1 + \frac{1}{2^{5d}})\frac{N}{2}$ and $\mathcal{T}_*$ is a set of tests all of whose queries fall in $\overline{\Pi}_*$. But this is a contradiction, since by Theorem 1.9 the number of such tests is bounded by $(\frac{1}{2^d} + \frac{1}{2^{20d}} + \frac{d}{2^{5d}})M$. The calculation is exactly the same as in Case (ii) of the soundness analysis for the dense k-subgraph problem.

Inapproximability factor. Clearly, the inapproximability factor is

$$\frac{1}{1 - \frac{1}{2^{5d}}} \approx 1 + \frac{1}{2^{5d}} = 1 + \frac{1}{2^{O(1/\epsilon \log(1/\epsilon))}}.$$

Appendix D. Boosting the hardness factor for bipartite clique. In this section, we prove Theorem 1.2 showing that bipartite clique is hard to approximate within a polynomial factor. We mimic the proof of Berman and Schnitger [10], who prove a similar boosting result for the clique problem. The boosting consists of taking a random induced subgraph of a product graph.

DEFINITION D.1. *For a bipartite graph $G(V, W, E)$ with $|V| = |W| = n$, let $OPT_{BC}(G)$ denote a maximum integer t such that the complete bipartite graph $K_{t,t}$ is a subgraph of G.*

DEFINITION D.2. *For a bipartite graph $G(V, W, E)$ with $|V| = |W| = n$, and an integer k, let $G^k(V', W', E')$ denote the product graph defined as follows:*

- $V' = V^k, W' = W^k$. *Thus* $|V'| = |W'| = n^k$.
- $((v_1, v_2 \ldots, v_k), (w_1, w_2, \ldots, w_k)) \in E' \iff \forall i, j, \quad 1 \le i, j \le k, \ (v_i, w_j) \in E$.

LEMMA D.3. *Let $G(V, W, E)$ be a bipartite graph with $|V| = |W| = n$ such that $OPT_{BC}(G) \ge \alpha n$. Let $H(V_*, W_*, E_*)$ be a random induced subgraph of $G^k(V', W', E')$, where $|V_*| = |W_*| = O(n^2/\alpha^k)$. Then w.h.p. we have $OPT_{BC}(H) \ge \frac{1}{2}\alpha^k |V_*|$.*

Proof. Let $V_1 \subseteq V$, $W_1 \subseteq W$ be such that $|V_1| = |W_1| = \alpha n$ and the subgraph of G induced on V_1, W_1 is a complete bipartite graph. Clearly, the subgraph of G^k induced on V_1^k, W_1^k is a complete bipartite graph.

Now V_* is a random subset of V^k of size $O(n^2/\alpha^k)$ and $|V^k| = n^k$, $|V_1^k| = \alpha^k n^k$. We will show that w.h.p. $|V_* \cap V_1^k| \ge \frac{1}{2}\alpha^k |V_*|$. We may assume that vertices in V_* are picked randomly and independently from V^k. For every vertex in V_*, the probability that it belongs to V_1^k is α^k. Thus the expected size of $|V_* \cap V_1^k|$ is $\alpha^k |V_*|$. Using Chernoff bounds,

$$\Pr\left[|V_* \cap V_1^k| \le \frac{1}{2}\alpha^k |V_*|\right] \ \le \ 2^{-\Omega(\alpha^k |V_*|)} \le 2^{-\Omega(n^2)}.$$

Similarly w.h.p., $|W_* \cap W_1^k| \ge \frac{1}{2}\alpha^k |W_*|$. Clearly, the induced subgraph of H on $V_* \cap V_1^k, W_* \cap W_1^k$ is a complete bipartite graph and this proves the lemma. ∎

LEMMA D.4. *Let $G(V, W, E)$ be a bipartite graph with $|V| = |W| = n$ such that $OPT_{BC}(G) \le \beta n$. Let $H(V_*, W_*, E_*)$ be a random induced subgraph of $G^k(V', W', E')$, where $|V_*| = |W_*| = O(n^2/\beta^k)$. Then w.h.p. we have $OPT_{BC}(H) \le 2\beta^k |V_*|$.*

Proof. It suffices to prove that w.h.p., for *every* complete bipartite subgraph $(\tilde{V}, \tilde{W})$ of G^k, we have either $|V_* \cap \tilde{V}| \le 2\beta^k |V_*|$ or $|W_* \cap \tilde{W}| \le 2\beta^k |W_*|$.

In fact, it suffices to show this statement for every *maximal* complete bipartite subgraph of G^k. It is easily seen that a maximal complete bipartite subgraph of G^k must be of the form (V_1^k, W_1^k), where (V_1, W_1) is a complete bipartite subgraph of G. In particular, the number of maximal complete bipartite subgraphs of G^k is bounded by 2^{2n}, and for any such subgraph (V_1^k, W_1^k), either $|V_1^k| \leq \beta^k n^k$ or $|W_1^k| \leq \beta^k n^k$.

Fix any maximal complete bipartite subgraph of G^k, induced on (V_1^k, W_1^k), and assume w.l.o.g. that $|V_1^k| \leq \beta^k n^k$. We may assume that vertices in V_* are picked randomly and independently from V'. For every vertex in V_*, the probability that it belongs to V_1^k is at most β^k. Thus the expected size of $|V_* \cap V_1^k|$ is at most $\beta^k |V_*|$. Using Chernoff bounds,

$$\Pr\left[|V_* \cap V_1^k| \geq 2\beta^k |V_*|\right] \;\leq\; 2^{-\Omega(\beta^k |V_*|)} \leq 2^{-\Omega(n^2)}.$$

Taking union bound over all possible (at most 2^{2n} many) maximal complete bipartite subgraphs of G^k proves the lemma. □

D.1. Proof of Theorem 1.2. Now we prove Theorem 1.2. Fix $\epsilon > 0$ to be an arbitrarily small constant. Let $G(V, W, E)$ be the bipartite graph given by the reduction in Appendix C.3. By adding dummy vertices to the LHS, we may assume that $|V| = |W| = M$. If the size of the original SAT instance is n, then note that $M = 2^{n^\epsilon}$, and for an integer $d = O(1/\epsilon \log(1/\epsilon))$, the graph G satisfies the following:

1. If the SAT instance is a YES instance, then $OPT_{BC}(G) \geq \alpha M$, where $\alpha = \frac{3}{4}\frac{1}{2^{d-1}}$
2. If the SAT instance is a NO instance, then $OPT_{BC}(G) \leq \beta M$, where $\beta = (1 - \frac{1}{2^{5d}})\alpha$.

Let $k = 2^{5d} \log M$ and let H be a random induced subgraph of G^k of size $S = O(M^2/\beta^k)$. Applying Lemmas D.3 and D.4, we see that w.h.p.,

1. if the SAT instance is a YES instance, then $OPT_{BC}(H) \geq \frac{1}{2}\alpha^k S$;
2. if the SAT instance is a NO instance, then $OPT_{BC}(H) \leq 2\beta^k S$.

Thus the hardness factor is

$$\frac{1}{4}(\alpha/\beta)^k \geq \frac{1}{4}\left(1 + \frac{1}{2^{5d}}\right)^{2^{5d}\log M} \geq 2^{\log M} = M.$$

On the other hand,

$$S = O(M^2/\beta^k) \leq M^2 2^{dk} = M^{2+d2^{5d}} = 2^{(2+d2^{5d})n^\epsilon}.$$

Thus the hardness factor can be expressed as $S^{\frac{1}{2+d2^{5d}}}$. Hence, assuming that SAT $\notin$ BPTIME(2^{n^ϵ}), we conclude that it is hard to approximate bipartite clique on graph of size S within factor $S^{\epsilon'}$ for $\epsilon' = \frac{1}{2^{O(1/\epsilon \log(1/\epsilon))}}$.

Appendix E. The sum-check protocol. In this section, we describe the (usual) sum-check protocol. It should familiarize the reader with the ideas and techniques involved in constructing the GSC protocol.

Given a table of values of a degree D polynomial $\tilde{f} : \mathbb{F}^m \mapsto \mathbb{F}$, the goal of the protocol is to verify that the sum of values of $\tilde{f}$ over $\{0,1\}^m$ equals the given *target value*. A trivial protocol would read the values of $\tilde{f}$ on all points in $\{0,1\}^m$, whereas the sum-check protocol needs to read only $O(m)$ values. The verifier of the sum-check protocol needs access to additional auxiliary information. Specifically, he has access to polynomials $\{g_r : |\ 1 \leq r \leq m-1\}$, also given as table of values (the intended

role of these polynomials is explained below). Here is the formal description of the protocol.

$SC(a, \mathbb{F}, \tilde{f}, m, D, \{g_r | 1 \le r \le m-1\})$ consists of the following:

1. $\mathbb{F}$ is a field and $a \in \mathbb{F}$ is a *target* value.
2. $\tilde{f} \in \mathbb{F}[\mathbf{x}]$ is a degree D polynomial, where $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ is a vector of m formal variables. We are given $\tilde{f}$ as a table of values.
3. Every g_r is a polynomial in $\mathbb{F}[\boldsymbol{\theta}]$, where $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_m)$ is a vector of formal variables. The degree of g_r is D and g_r is guaranteed to depend only on $\theta_1, \ldots, \theta_r$. The polynomials g_r are given as a table of values.

The goal is to (probabilistically) verify that

$$a = \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_m \in \{0,1\}} \tilde{f}(x_1, x_2, \ldots, x_m) \tag{29}$$

by reading a constant number of values of $\tilde{f}$ and of each of the polynomials g_r.

The values of the polynomial g_r are supposed to give sums of values of the polynomial $\tilde{f}$ over *cubes* of dimension $m - r$. To be precise, g_r is intended to be

$$g_r(\theta_1, \theta_2, \ldots, \theta_r) = \sum_{x_{r+1} \in \{0,1\}} \sum_{x_{r+2} \in \{0,1\}} \cdots \sum_{x_m \in \{0,1\}} \tilde{f}(\theta_1, \theta_2, \ldots, \theta_r, x_{r+1}, x_{r+2}, \ldots, x_m). \tag{30}$$

Here is the basic idea behind the protocol: the task of verifying a summation over the m-dimensional cube $\{0,1\}^m$ is successively reduced to the task of verifying a summation over cubes of lower dimension. In the last stage, the sum over a 0-dimensional cube, i.e., value of $\tilde{f}$ at a single point, can be read off from the table of values of $\tilde{f}$. The key observation is that

$$g_r(\theta_1, \theta_2, \ldots, \theta_r) = g_{r+1}(\theta_1, \theta_2, \ldots, \theta_r, 0) + g_{r+1}(\theta_1, \theta_2, \ldots, \theta_r, 1).$$

One can design a protocol in a natural way: just verify the above equation for $0 \le r \le m-1$. Here, g_0 would denote the sum $\sum_{x_1, x_2, \ldots, x_m \in \{0,1\}} \tilde{f}(x_1, x_2, \ldots, x_m)$, which is supposed to equal the target value a. Also, $g_m(\theta_1, \ldots, \theta_m)$ would denote the value $\tilde{f}(\theta_1, \theta_2, \ldots, \theta_m)$, which can be read from the table of values of $\tilde{f}$.

The sum-check protocol. The protocol accepts if and only if all of the following checks are satisfied:

- Pick $\theta_1, \theta_2, \ldots, \theta_m \in \mathbb{F}$ at random.
- (Stage 0:) Check if
$$a = g_1(0) + g_1(1). \tag{31}$$
- (Stage r for $r = 1, \ldots, m-2$:) Check if
$$g_r(\theta_1, \ldots, \theta_r) = g_{r+1}(\theta_1, \ldots, \theta_r, 0) + g_{r+1}(\theta_1, \ldots, \theta_r, 1). \tag{32}$$
- (Stage $m-1$:) Check if
$$g_{m-1}(\theta_1, \ldots, \theta_{m-1}) = \tilde{f}(\theta_1, \ldots, \theta_{m-1}, 0) + \tilde{f}(\theta_1, \ldots, \theta_{m-1}, 1). \tag{33}$$

E.1. Completeness.

LEMMA E.1. *If the polynomial $\tilde{f}$ satisfies* (29)*, then it is possible to define polynomials $g_r(\boldsymbol{\theta})$ so that the protocol accepts with probability* 1.

Proof. The proof is obvious. Just define g_r as they are supposed to be, as in (30). ∎

E.2. Soundness.

LEMMA E.2. *If the polynomial $\tilde{f}$ does not satisfy* (29)*, then no matter what polynomials g_r one takes, the protocol accepts with negligible probability, i.e., with probability at most $O(\frac{Dm}{|\mathbb{F}|})$. It is assumed that the polynomials g_r have degree at most D and g_r depends only on $\theta_1, \ldots, \theta_r$.*

Proof. Let $\tilde{g}_r$ be polynomials obtained from $\tilde{f}$ using rule (30). We will argue in reverse order (i.e., $r = m-1, m-2, \ldots, 1$) that unless $g_r \equiv \tilde{g}_r$, the protocol succeeds with negligible probability. The only fact we use is that two polynomials agree with nonnegligible probability if and only if they are identical as formal polynomials.

(*Case* $r = m-1$.) Look at Stage $m-1$, equation (33). The RHS is equal to $\tilde{g}_{m-1}(\theta_1, \ldots, \theta_{m-1})$. Hence, we must have $g_{m-1} = \tilde{g}_{m-1}$ if the verifier is to accept with nonnegligible probability.

(*Case* $1 \le r \le m-2$.) Assume that we have already proved that $g_{r+1} = \tilde{g}_{r+1}$. The RHS of (32) is $\tilde{g}_r(\theta_1, \ldots, \theta_r)$. Hence, we must have $g_r = \tilde{g}_r$ if the verifier is to accept with nonnegligible probability.

Now we show that the test fails in Stage 0. This is because

$$\begin{aligned} a &\neq \sum_{x_1, x_2, \ldots, x_m \in \{0,1\}} \tilde{f}(x_1, x_2, \ldots, x_m) \\ &= \tilde{g}_1(0) + \tilde{g}_1(1) \\ &= g_1(0) + g_1(1). \end{aligned}$$

□

Acknowledgments. I would like to thank Guy Kindler, Muli Safra, Misha Alekhnovich, Venkatesan Guruswami, Noga Alon, Sanjeev Arora, Madhu Sudan, Avi Wigderson, Moses Charikar, and Howard Karloff for many helpful discussions. Thanks to anonymous referees for pointing out a bug in the FOCS 2004 version of Theorem 1.2.

REFERENCES

[1] M. ALEKHNOVICH, *More on average case vs. approximation complexity*, in Proceedings of the 44th IEEE Symposium on Foundations of Computer Science, 2003, pp. 298–307.

[2] S. ARORA, *Probabilistic Checking of Proofs and the Hardness of Approximation Problems*, Ph.D. thesis, University of California at Berkeley, 1994.

[3] S. ARORA AND C. LUND, *The approximability of NP-hard problems*, in Approximation Algorithms for NP-hard Problems, D. Hochbaum, ed., PWS Publishing, Boston, 1996.

[4] S. ARORA AND S. SAFRA, *Probabilistic checking of proofs: A new characterization of NP*, J. ACM, 45 (1998), pp. 70–122.

[5] S. ARORA AND M. SUDAN, *Improved low-degree testing and its applications*, Combinatorica, 23 (2003), pp. 365–426.

[6] S. ARORA, L. BABAI, J. STERN, AND E. Z. SWEEDYK, *The hardness of approximate optima in lattices, codes and systems of linear equations*, J. Comput. Systems Sci., 54 (1997), pp. 317–331.

[7] S. ARORA, C. LUND, R. MOTAWANI, M. SUDAN, AND M. SZEGEDY, *Proof verification and the hardness of approximation problems*, J. ACM, 45 (1998), pp. 501–555.

[8] M. BELLARE, O. GOLDREICH, AND M. SUDAN, *Free bits, PCPs, and nonapproximability—towards tight results*, SIAM J. Comput., 27 (1998), pp. 804–915.

[9] E. BEN-SASSON, O. GOLDREICH, P. HARSHA, M. SUDAN, AND S. VADHAN, *Robust PCPs of proximity, shorter PCPs and applications to coding*, in Proceedings of the 36th ACM Symposium on Theory of Computing, 2004, pp. 1–10.

[10] P. BERMAN AND G. SCHNITGER, *On the complexity of approximating the independent set problem*, Inform. and Comput., 96 (1992), pp. 77–94.

[11] A. BLUM, *Algorithms for Approximate Graph Coloring*, Ph.D. thesis, MIT/LCS/TR-506, MIT, Cambridge, MA, 1991.

[12] M. BLUM, M. LUBY, AND R. RUBINFELD, *Self-testing/correcting with applications to numerical problems*, J. Comput. Systems Sci., 47 (1993), pp. 549–595.

[13] I. Dinur and S. Safra, *On the importance of being biased*, in Proceedings of the 34th ACM Symposium on Theory of Computing, 2002, pp. 33–42.
[14] I. Dinur, V. Guruswami, S. Khot, and O. Regev, *A new multilayered PCP and the hardness of hypergraph vertex cover*, in Proceedings of the 35th ACM Symposium on Theory of Computing, 2003, pp. 595-601.
[15] I. Dumer, D. Micciancio, and M. Sudan, *Hardness of approximating the minimum distance of a linear code*, IEEE Trans. Inform. Theory, 49 (2003), pp. 22–37.
[16] U. Feige, *A threshold of ln n for approximating set cover*, J. ACM, 45 (1998), pp. 634–652.
[17] U. Feige, *Relations between average case complexity and approximation complexity*, in Proceedings of the 34th ACM Symposium on Theory of Computing, 2002, pp. 534–543.
[18] U. Feige and J. Kilian, *Zero knowledge and the chromatic number*, J. Comput. Systems Sci., 57 (1998), pp. 187–199.
[19] U. Feige and S. Kogan, *Hardness of Approximation of the Balanced Complete Bipartite Subgraph Problem*, Tech. Report MCS04-04, Department of Computer Science and Applied Math., Weizmann Institute of Science, Rehovot, Israel, 2004.
[20] U. Feige and R. Krauthgamer, *A polylogarithmic approximation of the minimum bisection*, SIAM J. Comput., 31 (2002), pp. 1090–1118.
[21] U. Feige, G. Kortsarz, and D. Peleg, *The dense k-subgraph problem*, Algorithmica, 29 (2001), pp. 410–421.
[22] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, *Interactive proofs and the hardness of approximating cliques*, J. ACM, 43 (1996), pp. 268–292.
[23] V. Guruswami, J. Håstad, and M. Sudan, *Hardness of approximate hypergraph coloring*, in Proceedings of the 41st IEEE Symposium on Foundations of Computer Science, 2000, pp. 149–158.
[24] P. Harsha and M. Sudan, *Small PCPs with low query complexity*, Comput. Complexity, 9 (2000), pp. 157–201.
[25] J. Håstad, *Clique is hard to approximate within* $n^{1-\epsilon}$, Acta Math., 182 (1999), pp. 105–142.
[26] J. Håstad, *Some optimal inapproximability results*, J. ACM, 48 (2001), pp. 798–859.
[27] J. Holmerin, *Vertex cover on* 4*-regular hyper-graphs is hard to approximate within* 2*-epsilon*, in Proceedings of the 34th ACM Symposium on Theory of Computing, 2002, pp. 544–552.
[28] S. Khot, *Hardness of approximating the shortest vector problem in lattices*, in Proceedings of the 45th IEEE Symposium on Foundations of Computer Science, 2004, pp. 126–135.
[29] S. Khot, *Improved inapproximability results for maxclique, chromatic number and approximate graph coloring*, in Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, 2001, pp. 600–609.
[30] S. Khot and J. Holmerin, *A new PCP outer verifier with applications to homogeneous linear equations and Max-Bisection*, in Proceedings of the 36th ACM Symposium on Theory of Computing, 2004, pp. 11–20.
[31] A. Lubotzky, R. Phillips, and P. Sarnak, *Ramanujan graphs*, Combinatorica, 8 (1988), pp. 261–277.
[32] M. Luby and A. Wigderson, *Pairwise Independence and Derandomization*, Tech. Report TR-95-035, International Computer Science Institute, Berkeley, CA, 1995.
[33] R. Raz, *A parallel repetition theorem*, SIAM J. Comput., 27 (1998), pp. 763–803.
[34] R. Raz and S. Safra, *A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP*, in Proceedings of the 29th ACM Symposium on Theory of Computing, 1997, pp. 475–484.
[35] R. Rubinfeld and M. Sudan, *Robust characterizations of polynomials with applications to program testing*, SIAM J. Comput., 25 (1996), pp. 252–271.
[36] L. Trevisan, *Inapproximability of Combinatorial Optimization Problems*, survey paper, 2004; French version appeared in Optimisation Combinatoire 2, V. Paschos, ed., Hermes, Paris, 2005.
[37] V. Vazirani. *Approximation Algorithms*, Springer-Verlag, Berlin, 2001.

SIAM J. COMPUT.
Vol. 36, No. 4, pp. 1072–1094

DETERMINISTIC EXTRACTORS FOR BIT-FIXING SOURCES BY OBTAINING AN INDEPENDENT SEED*

ARIEL GABIZON†, RAN RAZ†, AND RONEN SHALTIEL‡

Abstract. An (n,k)-bit-fixing source is a distribution X over $\{0,1\}^n$ such that there is a subset of k variables in $X_1,\ldots,X_n$ which are uniformly distributed and independent of each other, and the remaining $n-k$ variables are fixed. A deterministic bit-fixing source extractor is a function $E:\{0,1\}^n \to \{0,1\}^m$ which on an arbitrary (n,k)-bit-fixing source outputs m bits that are statistically close to uniform. Recently, Kamp and Zuckerman [*Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, 2003, pp. 92–101] gave a construction of a deterministic bit-fixing source extractor that extracts $\Omega(k^2/n)$ bits and requires $k > \sqrt{n}$.

In this paper we give constructions of deterministic bit-fixing source extractors that extract $(1-o(1))k$ bits whenever $k > (\log n)^c$ for some universal constant $c > 0$. Thus, our constructions extract almost all the randomness from bit-fixing sources and work even when k is small. For $k \gg \sqrt{n}$ the extracted bits have statistical distance $2^{-n^{\Omega(1)}}$ from uniform, and for $k \le \sqrt{n}$ the extracted bits have statistical distance $k^{-\Omega(1)}$ from uniform.

Our technique gives a general method to transform deterministic bit-fixing source extractors that extract few bits into extractors which extract almost all the bits.

Key words. bit-fixing sources, deterministic extractors, derandomization, seeded extractors, seed obtainers

AMS subject classifications. 68Q99, 68R05

DOI. 10.1137/S0097539705447049

1. Introduction.

1.1. Deterministic randomness extractors.

A "deterministic randomness extractor" is a function that "extracts" bits that are (statistically close to) uniform from "weak sources of randomness" which may be very far from uniform.

DEFINITION 1.1 (deterministic extractor). *Let $\mathcal{C}$ be a class of distributions on $\{0,1\}^n$. A function $E:\{0,1\}^n \to \{0,1\}^m$ is a* deterministic ϵ-extractor *for $\mathcal{C}$ if for every distribution X in $\mathcal{C}$ the distribution $E(X)$ (obtained by sampling x from X and computing $E(x)$) is ϵ-close to the uniform distribution on m-bit strings.*[1]

The distributions X in $\mathcal{C}$ are often referred to as "weak random sources," that is, distributions that "contain" some randomness. Given a class $\mathcal{C}$, the goal of this field is to design *explicit* (that is, efficiently computable) deterministic extractors that extract as many random bits as possible.

1.2. Some related work on randomness extraction.

Various classes $\mathcal{C}$ of distributions were studied in the literature. The first construction of deterministic extractors can be traced back to von Neumann [37], who showed how to use many

*Received by the editors February 6, 2005; accepted for publication (in revised form) August 18, 2005; published electronically December 15, 2006. A preliminary version of this paper appeared in *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, 2004.

http://www.siam.org/journals/sicomp/36-4/44704.html

†Faculty of Mathematics and Computer Science, The Weizmann Institute of Science, P.O. Box 26, Rehovot 76100, Israel (ariel.gabizon@weizmann.ac.il, ran.raz@weizmann.ac.il). The research of the first and second authors was supported by an Israel Science Foundation (ISF) grant.

‡ Department of Computer Science, Faculty of Social Sciences, University of Haifa, Haifa 31905, Israel (ronen@cs.haifa.ac.il). The research of this author was supported by the Koshland scholarship.

[1]Two distributions P and Q over $\{0,1\}^m$ are ϵ-close (denoted by $P \overset{\epsilon}{\sim} Q$) if for every event $A \subseteq \{0,1\}^m$, $|P(A)-Q(A)| \le \epsilon$.

independent tosses of a biased coin (with unknown bias) to obtain an unbiased coin. Blum [6] considered sources that are generated by a finite Markov chain. Santha and Vazirani [29], Vazirani [34, 35], Chor and Goldreich [10], Barak, Impagliazzo, and Wigderson [2], Barak et al. [3], and Raz [25] studied sources that are composed of several independent samples from various classes of distributions. Trevisan and Vadhan [31] studied sources which are "samplable" by small circuits.

A negative result was given by Santha and Vazirani that exhibits a very natural class of high-entropy sources that do not have deterministic extractors. This led to the development of a different notion of extractors called "seeded extractors." Such extractors are allowed to use a short seed of few truly random bits when extracting randomness from a source. (The notion of "seeded extractors" emerged from attempts to simulate probabilistic algorithms using weak random sources [36, 10, 12, 38, 39] and was explicitly defined by Nisan and Zuckerman [23].) Unlike deterministic extractors, seeded extractors can extract randomness from the most general class of sources: sources with high (min)-entropy. The reader is referred to [21, 22, 30, 32] for various surveys on randomness extractors.

1.3. Bit-fixing sources. In this paper we concentrate on the family of "bit-fixing sources" introduced by Chor et al. [11]. A distribution X over $\{0,1\}^n$ is a bit-fixing source if there is a subset $S \subseteq \{1,\ldots,n\}$ of "good indices" such that the bits X_i for $i \in S$ are independent fair coins and the rest of the bits are fixed.[2]

DEFINITION 1.2 (bit-fixing sources and extractors). *A distribution X over $\{0,1\}^n$ is an (n,k)-*bit-fixing source *if there exists a subset $S = \{i_1,\ldots,i_k\} \subseteq \{1,\ldots,n\}$ such that $X_{i_1}, X_{i_2}, \ldots, X_{i_k}$ is uniformly distributed over $\{0,1\}^k$ and for every $i \notin S$, X_i is constant.*

A function $E : \{0,1\}^n \to \{0,1\}^m$ is a deterministic (k,ϵ)-bit-fixing source extractor if it is a deterministic ϵ-extractor for all (n,k)-bit-fixing sources.

One of the motivations given in the literature for studying deterministic bit-fixing source extractors is that they are helpful in cryptographic scenarios in which an adversary learns (or alters) $n-k$ bits of an n-bit-long secret key [11]. Loosely speaking, one wants cryptographic protocols to remain secure even in the presence of such adversaries. Various models for such "exposure resilient cryptography" were studied [28, 7, 8, 14]. The reader is referred to [13] for a comprehensive treatment of "exposure resilient cryptography" and its relation to deterministic bit-fixing source extractors.

Every (n,k)-bit-fixing source "contains" k "bits of randomness." It follows that any deterministic (k,ϵ)-bit-fixing source extractor with $\epsilon < 1/2$ can extract at most k bits. The function $E(x) = \oplus_{1\le i\le n} x_i$ is a deterministic $(k,0)$-bit-fixing source extractor which extracts one bit for any $k \ge 1$. Chor et al. [11] concentrated on deterministic "errorless" extractors (that is, deterministic extractors in which $\epsilon = 0$). They show that such extractors cannot extract even two bits when $k < n/3$. They also give some constructions of deterministic errorless extractors for large k.

Our focus is on extractors with error $\epsilon > 0$ (which allows extracting many bits for many choices of k). A probabilistic argument shows the existence of a deterministic (k,ϵ)-bit-fixing source extractor that extracts $m = k - O(\log(n/\epsilon))$ bits for any choice of k and ϵ. Thus, it is natural to try to achieve such parameters by explicit constructions.

[2] We remark that such sources are often referred to as "oblivious bit-fixing sources" to differentiate them from other types of "nonoblivious" bit-fixing sources in which the bits outside of S may depend on the bits in S (cf. [5]). In this paper we are concerned only with the "oblivious case."

In a recent paper Kamp and Zuckerman [17] constructed explicit deterministic (k,ϵ)-bit-fixing source extractors that extract $m = \eta k^2/n$ bits for some constant $0 < \eta < 1$ with $\epsilon = 2^{-\Omega(k^2/n)}$. They pose the open problem to extract more bits from such sources. Note that the extractor of Kamp and Zuckerman is inferior to the nonexplicit extractor in two respects:

- It works only when $k > \sqrt{n}$.
- Even when $k > \sqrt{n}$ the extractor may extract only a small fraction of the randomness. For example, if $k = n^{1/2+\alpha}$ for some $0 < \alpha < 1/2$, the extractor extracts only $m = \eta n^{2\alpha}$ bits.

1.4. Our results. In this paper, we give two constructions of deterministic bit-fixing source extractors that extract $m = (1-o(1))k$ bits from (n,k)-bit-fixing sources. Our first construction is for the case of $k \gg \sqrt{n}$.

THEOREM 1.3. *For every constant $0 < \gamma < 1/2$ there exists an integer n' (depending on γ) such that for any $n > n'$ and any k, there is an explicit deterministic (k,ϵ)-bit-fixing source extractor $E : \{0,1\}^n \to \{0,1\}^m$, where $m = k - n^{1/2+\gamma}$ and $\epsilon = 2^{-\Omega(n^\gamma)}$.*

Consider $k = n^{1/2+\alpha}$ for some constant $0 < \alpha < 1/2$. We can choose any $\gamma < \alpha$ and extract $m = n^{1/2+\alpha} - n^{1/2+\gamma}$ bits whereas the construction of [17] extracts only $m = O(n^{2\alpha})$ bits. For this choice of parameters we achieve error $\epsilon = 2^{-\Omega(n^\gamma)}$ whereas [17] achieves a slightly smaller error $\epsilon = 2^{-\Omega(n^{2\alpha})}$. We remark that this comes close to the parameters achieved by the nonexplicit construction which can extract $m = n^{1/2+\alpha} - n^{1/2+\gamma}$ with error $\epsilon = 2^{-\Omega(n^{1/2+\gamma})}$.

Our second construction works for any $k > (\log n)^c$, for some universal constant c. However, the error in this construction is larger.

THEOREM 1.4. *There exist constants $c > 0$ and $0 < \mu, \nu < 1$ such that for any large enough n and any $k \geq \log^c n$, there is an explicit deterministic (k,ϵ)-bit-fixing source extractor $E : \{0,1\}^n \to \{0,1\}^m$, where $m = k - O(k^\nu)$ and $\epsilon = O(k^{-\mu})$.*

We remark that using the technique of [17] one can achieve a much smaller error ($\epsilon = 2^{-\sqrt{k}}$) at the cost of extracting very few bits ($m = \Omega(\log k)$). The precise details are given in Theorem 4.1.

1.5. Overview of techniques. We develop a general technique that transforms any deterministic bit-fixing source extractor that extracts only very few bits into one that extracts almost all of the randomness in the source. This transformation makes use of "seeded extractors."

1.5.1. Seeded randomness extractors. A seeded randomness extractor is a function which receives two inputs: In addition to a sample from a source X, a seeded extractor also receives a short "seed" Y of few uniformly distributed bits. Loosely speaking, the extractor is required to output many more random bits than the number of bits "invested" as a seed.

DEFINITION 1.5 (seeded extractors). *Let $\mathcal{C}$ be a class of distributions on $\{0,1\}^n$. A function $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a* seeded ϵ-extractor *for $\mathcal{C}$ if for every source X in $\mathcal{C}$ the distribution $E(X,Y)$ (obtained by sampling x from X and a uniform $y \in \{0,1\}^d$ and computing $E(x,y)$) is ϵ-close to the uniform distribution on m-bit strings.*

A long line of research focuses on constructing such seeded extractors with as short as possible seed length that extract as many bits as possible from the most general family of sources that allow randomness extraction: the class of sources with high min-entropy.

DEFINITION 1.6 (seeded extractors for high min-entropy sources). *The min-entropy of a distribution X over $\{0,1\}^n$ is $H_\infty(X) = min_{x\in\{0,1\}^n} \log_2(1/\Pr(x))$. A function $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a* (k,ϵ)-extractor *if it is a seeded ϵ-extractor for the class of all sources X with $H_\infty(X) \geq k$.*

There are explicit constructions of (k,ϵ)-extractors that use a seed of length $\text{polylog}(n/\epsilon)$ to extract k random bits. The reader is referred to [30] for a detailed survey on various constructions of seeded extractors.

Our goal is to construct *deterministic* bit-fixing source extractors. Nevertheless, in the next definition we introduce the concept of a *seeded* bit-fixing source extractor. We use such extractors as a component in our construction of deterministic bit-fixing source extractors.

DEFINITION 1.7 (seeded extractors for bit-fixing sources). *A function $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a* seeded (k,ϵ)-bit-fixing source extractor *if it is a seeded ϵ-extractor for the class of all (n,k)-bit-fixing sources.*

1.5.2. Seed obtainers. There is a very natural way to try to transform a deterministic bit-fixing source extractor that extracts few (say $\text{polylog}\, n$) bits into one that extracts many bits: First run the deterministic bit-fixing source extractor to extract few bits from the source, and then use these bits as a seed to a seeded extractor that extracts all the bits from the source. The obvious difficulty with this approach is that typically the output of the first extractor is *correlated* with the imperfect random source. Seeded extractors are only guaranteed to work when their seed is *independent* from the random source. To overcome this difficulty we introduce a new object which we call a *"seed obtainer."*

Loosely speaking, a seed obtainer is a function F that, given an (n,k)-bit-fixing source X, outputs two strings X' and Y with the following properties:

- X' is an (n,k')-bit-fixing source with $k' \approx k$ good bits.
- Y is a short string that is almost uniformly distributed.
- X' and Y are almost independent.

The precise definition is slightly more technical and is given in Definition 3.1. Note that a seed obtainer reduces the task of constructing *deterministic* extractors into that of constructing *seeded* extractors: Given a bit-fixing source X, one first runs the seed obtainer to obtain X' and a short Y and then uses Y as a seed to a seeded extractor that extracts all the randomness from X'. (In fact, it is even sufficient to construct seeded extractors for bit-fixing sources.)

1.5.3. Constructing seed obtainers. Note that every seed obtainer $F(X) = (X', Y)$ "contains" a deterministic bit-fixing source extractor by setting $E(X) = Y$. We show how to transform any deterministic bit-fixing source extractor into a seed obtainer. In this transformation the length of the "generated seed" Y is roughly the length of the output of the original extractor.

It is helpful to explain the intuition behind this transformation when applied to a specific deterministic bit-fixing source extractor. Consider the "xor-extractor" $E(x) = \oplus_{1\leq i\leq n} x_i$. Let X be some (n,k)-bit-fixing source, and let $Z = E(X)$. Note that the output bit Z is indeed very correlated with the input X. Nevertheless, suppose that we somehow obtain a random small subset of the indices of X. It is expected that the set contains a small fraction of the good bits. Let X' be the string that remains after "removing" the indices in the sampled set. The important observation is that X' is a bit-fixing source that is *independent* from the output Z. It turns out that the same phenomenon happens for every deterministic bit-fixing source extractor $E(X)$. However, it is not clear how to use this idea as we don't have

additional random bits to perform the aforementioned sampling of a random set. Surprisingly, we show how to use the bits extracted by the extractor E to perform this sampling.

Following this intuition, given an extractor $E(X)$ which extracts an m-bit string Z, we partition Z into two parts Y and W. We then use W as a seed to a randomness-efficient method of "sampling" a small subset T of $\{1,\ldots,n\}$. The first output of the seed obtainer X' is given by "removing" the sampled indices from X. More formally, X' is the string X restricted to the indices outside of T. The second output is Y (the other part of the output of the extractor E).

The intuition is that if T is a size n/r uniformly distributed subset of $\{1,\ldots,n\}$, then it is expected to hit approximately k/r good bits from the source. Thus, $k-k/r$ good bits remain in X'. We will require that the extractor E extracts randomness from $(n,k/r)$-bit-fixing sources. Loosely speaking, we can hope that E will extract its output from X_T (the string obtained by restricting X to the indices of T). Thus, its output will be independent from X' (the string obtained by removing X_T).

Note that the intuition above is far from being precise. The set T is sampled using random bits W that are extracted from the source X, and thus T depends on X. Whereas, the intuition corresponds to the case where T is independent from X. The precise argument appears in section 3. We remark that the analysis requires that the extractor E has error ϵ that is smaller than $2^{-|W|}$ (where $|W|$ is the number of bits used by the sampling method).

1.5.4. A deterministic extractor for large k (i.e., $k \gg \sqrt{n}$). Our first construction builds on the deterministic bit-fixing source extractor of Kamp and Zuckerman [17] that works for $k > \sqrt{n}$ and extracts at least $\Omega(k^2/n)$ bits from the source. We first transform this extractor into a seed obtainer F. Next, we run the seed obtainer F on the input source to generate a bit-fixing source X' and a seed Y. Finally, we extract all the randomness in X' by running a seeded extractor on X' using Y as seed.

1.5.5. A deterministic extractor for small k (i.e., $k < \sqrt{n}$). In order to use our technique for $k < \sqrt{n}$ we need to start with some deterministic bit-fixing source extractor that works when $k < \sqrt{n}$ and extracts a small number of bits. Our first observation is that methods similar to the ones of Kamp and Zuckerman [17] can be applied when $k < \sqrt{n}$ but only give deterministic bit-fixing source extractors that extract very few bits (i.e., $\Omega(\log k)$ bits).[3]

Deterministic extractors that extract $\Omega(\log k)$ bits. Kamp and Zuckerman [17] consider the distribution obtained by using a bit-fixing source $X = (X_1,\ldots,X_n)$ to perform a random walk on a d-regular graph. (They consider a more general model of bit-fixing sources in which every symbol X_i ranges over an alphabet of size d.) The walk starts from some fixed vertex in the graph and at step i, one uses X_i to select a neighbor of the current vertex. They show that the distribution over the vertices converges to the uniform distribution at a rate which depends on k and the "spectral gap" of the graph. It is known that 2-regular graphs cannot have small "spectral gap." Indeed, this is why Kamp and Zuckerman consider alphabet size $d > 2$ which allows using d-regular expander graphs that have a small spectral gap. Nevertheless, using their technique while choosing the graph to be a short cycle of length $k^{1/4}$ produces an extractor construction which extracts $\log(k^{1/4}) = \Omega(\log k)$ bits.[4]

[3] This was observed independently by Lipton and Vishnoi [18].

[4] In fact, a similar idea is used in [17] in order to reduce the case of large d to the case of $d = 2$.

A seeded extractor for bit-fixing sources with seed length $O(\log\log n)$. Converting the deterministic bit-fixing source extractor above into a seed obtainer, we "obtain" an $\Omega(\log k)$-bit seed. This allows us to use a seeded extractor with seed length $d = \Omega(\log k)$. However, $d < \log n$ and by a lower bound of [23, 24] the class of high min-entropy sources does not have seeded extractors with seed $d < \log n$. To bypass this problem we construct a *seeded* extractor for *bit-fixing sources* with seed length $O(\log\log n)$. Note that the aforementioned deterministic extractor extracts this many bits as long as $k > \log^c n$ for some constant c (when $\Omega(\log k) \geq O(\log\log n)$).

The seeded extractor uses its seed to randomly partition the indices $\{1, \ldots, n\}$ into r sets $T_1, \ldots, T_r$ (for r equals, say, $\log^4 n$), with the property that with high probability each one of these sets contains at least one good bit. We elaborate on this partitioning method later. We then output r bits, where the ith bit is given by $\oplus_{j \in T_i} x_j$.

By combining the seed obtainer with the seeded bit-fixing source extractor we obtain a deterministic bit-fixing source extractor which extracts $r = \log^4 n$ bits. To extract more bits, we convert this deterministic extractor into a seed obtainer. At this point we obtain a seed of length $\log^4 n$ and can afford to use a seeded extractor which extracts all the remaining randomness.

Sampling and partitioning with only $O(\log\log n)$ random bits. We now explain how to use $O(\log\log n)$ random bits to partition the indices $\{1, \ldots, n\}$ into $r = \text{polylog} n$ sets $T_1, \ldots, T_r$ such that for any set $S \subseteq \{1, \ldots, n\}$ of size k, with high probability (probability at least $1 - O(1/\log n)$) all sets $T_1, \ldots, T_r$ contain approximately k/r indices from S.

Suppose we could afford using many random bits. A natural solution is to choose n random variables $V_1, \ldots, V_n \in \{1, \ldots, r\}$ and have T_j be the set of indices i such that $V_i = j$. We expect k/r bits to fall in each T_j, and by a union bound one can show that with high probability all sets $T_1, \ldots, T_r$ have a number of indices from S that is close to the expected value.

To reduce the number of random bits we derandomize the construction above and use random variables V_i which are ϵ-close to being pairwise independent (for $\epsilon = 1/\log^a n$ for some sufficiently large constant a). Such variables can be constructed using only $O(\log\log n)$ random bits [20, 1, 15] and suffice to guarantee the required properties.

The same technique also gives us a method for sampling a set T of indices in $\{1, \ldots, n\}$ (which we require in our construction of seed obtainers). We simply take the first set T_1. This sampling method uses only $O(\log\log n)$ random bits, and thus we can afford it when transforming our deterministic extractor into a seed obtainer. (Recall that our transformation uses part of the output of the deterministic extractor for sampling a subset of the indices.) We remark that this sampling technique was used previously by Reingold, Shaltiel, and Wigderson [27] as a component in a construction of seeded extractors.

1.6. Outline. In section 2 we define the notations used in this paper. In section 3 we introduce the concept of seed obtainers and show how to construct them from deterministic bit-fixing source extractors and "averaging samplers." In section 4 we observe that the technique of [17] can be used to extract few bits even when k is small. In section 5 we give constructions for averaging samplers. In section 6 we give a construction of a seeded bit-fixing source extractor that makes use of the sampling techniques of section 5. In section 7 we plug all the components together and prove our main theorems. Finally, in section 8 we give some open problems.

2. Preliminaries.

Notation. We use $[n]$ to denote the set $\{1, \ldots, n\}$. We use $P(S)$ to denote the set of subsets of a given set S. We use U_n to denote the uniform distribution over n bits. Given a distribution A we use $w \leftarrow A$ to denote the experiment in which w is chosen randomly according to A. Given a string $x \in \{0,1\}^n$ and a set $S \subseteq [n]$, we use x_S to denote the string obtained by restricting x to the indices in S. We denote the length of a string x by $|x|$. Logarithms will always be taken with base 2.

Asymptotic notation. As this paper has many parameters we now explain exactly what we mean when using $O(\cdot)$ and $\Omega(\cdot)$ in a statement involving many parameters. We use the Ω and O signs only to denote absolute constants (i.e., not depending on any parameters even if these parameters are considered constants). Furthermore, when writing, for example, $f(n) = O(g(n))$, we always explicitly mention the conditions on n (and maybe other parameters) for which the statement holds.

2.1. Averaging samplers.

A sampler is a procedure which, given a short seed, generates a subset $T \subseteq [n]$ such that for every set $S \subseteq [n]$, $|S \cap T|$ is with high probability "close to the expected size."

DEFINITION 2.1. *An $(n, k, k_{min}, k_{max}, \delta)$-sampler $Samp : \{0,1\}^t \to P([n])$ is a function such that for any $S \subseteq [n]$ such that $|S| = k$,*

$$\Pr_{w \leftarrow U_t}(k_{min} \leq |Samp(w) \cap S| \leq k_{max}) \geq 1 - \delta.$$

The definition above is nonstandard in several respects. In the more common definition (cf. [16]), a sampler is required to work for sets of arbitrary size. In the definition above (which is similar in spirit to the one in [33]), the sampler is only required to work against sets of size k and the bounds k_{min}, k_{max} are allowed to depend on k. Furthermore, we require that the sampler has "distinct samples" as we do not allow T to be a multiset.[5]

We will use samplers to "partition" bit-fixing sources. Note that in the case of an (n, k)-bit-fixing source, *Samp* returns a subset of indices such that, with high probability, the number of good bits in the subset is between k_{min} and k_{max}.

2.2. Probability distributions.

Some of the proofs in this paper require careful manipulations of probability distributions. We use the following notation. We use U_m to denote the uniform distribution on m-bit strings. We denote the probability of an event B under a probability distribution P by $\Pr_P[B]$. A random variable R that takes values in U is a function $R : \Omega \to U$ (where Ω is a probability space). We sometimes refer to R as a probability distribution over U (the distribution of the output of R). For example, given a random variable R and a distribution P, we sometimes write "$R = P$" and this means that the distribution of the output of R is equal to P. Given two random variables R_1, R_2 over the same probability space Ω, we use (R_1, R_2) to denote the random variable induced by the function $(R_1, R_2)(\omega) = (R_1(\omega), R_2(\omega))$. Given two probability distributions P_1, P_2 over domains Ω_1, Ω_2, we define $P_1 \otimes P_2$ to be the *product distribution* of P_1 and P_1 which is defined over the domain $\Omega_1 \times \Omega_2$.

[5] We remark that some of the "standard techniques" for constructing averaging samplers (such as taking a walk on an expander graph or using a randomness extractor) perform poorly in this setup and do not work when $k < \sqrt{n}$ (even if T is allowed to be a multiset). This happens because in order to even hit a set S of size k these techniques require sampling a (multi)set T of size larger than $(n/k)^2$ which is larger than n for $k < \sqrt{n}$. In contrast, note that a completely random set of size roughly n/k will hit a fixed set S of small size with good probability.

DEFINITION 2.2 (conditioning distributions and random variables). *Given a probability distribution P over some domain U and an event $A \subseteq U$ such that $\Pr_P[A] > 0$, we define a distribution $(P|A)$ over U as follows: Given an event $B \subseteq U$, $\Pr_{(P|A)}(B) = \Pr_P[B|A] = \frac{\Pr_P[A \cap B]}{\Pr_P[A]}$.*

We extend this definition to random variables $R : \Omega \to U$. Given an event $A \subseteq \Omega$, we define $(R|A)$ to be the probability distribution over U given by $\Pr_{(R|A)}[B] = \Pr_R[R \in B|A]$.

We also need the notion of convex combination of distributions.

DEFINITION 2.3 (convex combination of distributions). *Given distributions $P_1, \ldots, P_t$ over U and coefficients $\alpha_1, \ldots, \alpha_t \geq 0$ such that $\sum_{1 \leq i \leq t} \alpha_i = 1$, we define the distribution $P = \sum_{1 \leq i \leq t} \alpha_i P_i$ as follows: Given an event $B \subseteq U$, $\Pr_P[B] = \sum_{1 \leq i \leq t} \alpha_i \Pr_{P_i}[B]$.*

We also need the following technical lemmas.

LEMMA 2.4. *Let X, Y, and V be distributions over $\{0,1\}^n$ such that X is ϵ-close to U_n and $Y = \delta \cdot V + (1-\delta) \cdot X$. Then Y is $(2\delta + \epsilon)$-close to U_n.*

Proof. Let $B \subseteq \{0,1\}^n$ be some event such that

$$\begin{aligned} |\Pr_Y(B) - \Pr_{U_n}(B)| &= |\delta \Pr_V(B) + (1-\delta)\Pr_X(B) - \Pr_{U_n}(B)| \\ &\leq 2\delta + |\Pr_X(B) - \Pr_{U_n}(B)| \leq 2\delta + \epsilon. \qquad \square \end{aligned}$$

LEMMA 2.5. *Let (A, B) be a random variable that takes values in $\{0,1\}^u \times \{0,1\}^v$ and suppose that there exists some distribution P over $\{0,1\}^v$ such that for every $a \in \{0,1\}^u$ with $\Pr[A = a] > 0$ the distribution $(B|A = a)$ is ϵ-close to P. Then (A, B) is ϵ-close to $(A \otimes P)$.*

Proof.

$$\frac{1}{2} \cdot \sum_{a,b} |\Pr[(A,B) = (a,b)] - \Pr_{A \otimes P}[a,b]|$$

$$= \frac{1}{2} \cdot \sum_{a,b} |\Pr[A = a]\Pr[B = b|A = a] - \Pr[A = a]\Pr_P[b]|$$

$$\leq \frac{1}{2} \cdot \sum_a \Pr[A = a] \sum_b |\Pr[B = b|A = a] - \Pr_P[b]| \leq \epsilon/2. \qquad \square$$

LEMMA 2.6. *Let (A, B) be a random variable that takes values in $\{0,1\}^u \times \{0,1\}^v$ which is ϵ-close to $(A' \otimes U_v)$; then for every $b \in \{0,1\}^v$ the distribution $(A|B = b)$ is $(\epsilon \cdot 2^{v+1})$-close to A'.*

Proof. Assume for the purpose of contradiction that there exists some $b^* \in \{0,1\}^v$ such that the distribution $(A|B = b^*)$ is not α-close to A' for $\alpha = \epsilon \cdot 2^{v+1}$. Then there is an event D such that

$$|\Pr_{(A|B=b^*)}[D] - \Pr_{A'}[D]| > \alpha.$$

By complementing D if necessary we can without loss of generality (w.l.o.g.) remove the absolute value from the inequality above. We define an event D' over $\{0,1\}^u \times \{0,1\}^v$. The event $D' = \{(a,b)|b = b^*,\ a \in D\}$. We have that

$$\Pr_{(A', U_v)}[D'] = \Pr_{A'}[D] \cdot 2^{-v}.$$

And similarly,

$$\Pr_{(A,B)}[D'] = \Pr_{(A|B=b^*)}[D] \Pr_B[B = b^*].$$

We know that B is ϵ-close to U_v and therefore $\Pr_B[B = b^*] \geq 2^{-v} - \epsilon$. Thus,

$$\Pr_{(A,B)}[D'] - \Pr_{(A',U_v)}[D'] = \Pr_{(A|B=b^*)}[D] \Pr_B[B = b^*] - \Pr_{A'}[D] \cdot 2^{-v}$$

$$\geq \Pr_{(A|B=b^*)}[D](2^{-v} - \epsilon) - \Pr_{A'}[D] \cdot 2^{-v} \geq 2^{-v}[\Pr_{(A|B=b^*)}[D] - \Pr_{A'}[D]] - \epsilon.$$

By our assumption the expression in square brackets is at least α and thus

$$> 2^{-v}\alpha - \epsilon = \epsilon.$$

Thus, we get a contradiction. ∎

3. Obtaining an independent seed.

3.1. Seed obtainers and their application. One of the natural ways to try and extract *many* bits from imperfect random sources is to first run a "weak extractor" which extracts few bits from the input distribution and then use these few bits as a seed to a second extractor which extracts more bits. The obvious difficulty with this approach is that typically the output of the first extractor is *correlated* with the imperfect random source and it is not clear how to use it. (Seeded extractors are only guaranteed to work when the seed is *independent* from the random source.) In the next definition we introduce the concept of a *"seed obtainer"* that overcomes this difficulty. Loosely speaking, a seed obtainer is a deterministic function which, given a bit-fixing source X, outputs a new bit-fixing source X' (with roughly the same randomness) together with a short random seed Y which is *independent* from X'. Thus, the seed Y can later be used to extract randomness from X' using a seeded extractor.

DEFINITION 3.1 (seed obtainer). *A function $F : \{0,1\}^n \to \{0,1\}^n \times \{0,1\}^d$ is a (k, k', ρ)-seed obtainer if for every (n,k)-bit-fixing source X, the distribution $R = F(X)$ can be expressed as a convex combination of distributions $R = \eta Q + \sum_a \alpha_a R_a$ (here the coefficients η and α_a are nonnegative and $\eta + \sum_a \alpha_a = 1$) such that $\eta \leq \rho$ and for every a there exists an (n,k')-bit-fixing source Z_a such that R_a is ρ-close to $Z_a \otimes U_d$.*

It follows that given a seed obtainer one can use a *seeded extractor* for bit-fixing sources to construct a *deterministic* (i.e., seedless) extractor for bit-fixing sources.

THEOREM 3.2. *Let $F : \{0,1\}^n \to \{0,1\}^n \times \{0,1\}^d$ be a (k, k', ρ)-seed obtainer. Let $E_1 : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be a seeded (k', ϵ)-bit-fixing source extractor. Then $E : \{0,1\}^n \to \{0,1\}^m$ defined by $E(x) = E_1(F(x))$ is a deterministic $(k, \epsilon + 3\rho)$-bit-fixing source extractor*

Proof. By the definition of a seed obtainer we have that $E(X) = \eta E_1(Q) + \sum_a \alpha_a E_1(R_a)$ for some $\eta \leq \rho$. For each a we have that $E_1(R_a)$ is $(\epsilon + \rho)$-close to U_m. It follows that $E(X)$ is $(\epsilon + \rho)$-close to $\eta E_1(Q) + (1 - \eta)U_m$ and therefore by Lemma 2.4 we have that $E(X)$ is $(2\eta + \epsilon + \rho)$-close to uniform. The lemma follows because $2\eta + \epsilon + \rho \leq \epsilon + 3\rho$. ∎

3.2. Constructing seed obtainers. Note that every seed obtainer "contains" a deterministic extractor for bit-fixing sources. More precisely, given a seed obtainer $F(x) = (x', y)$, the function $E(x) = y$ is a deterministic extractor for bit-fixing sources. We now show how to convert any deterministic bit-fixing source extractor with sufficiently small error into a seed obtainer.

Ingredients:
- An $(n, k, k_{min}, k_{max}, \delta)$-sampler $Samp : \{0,1\}^t \to P([n])$.
- A deterministic (k_{min}, ϵ)-bit-fixing source extractor $E : \{0,1\}^n \to \{0,1\}^m$ with $m > t$.

Result: A (k, k', ρ)-seed obtainer $F : \{0,1\}^n \to \{0,1\}^n \times \{0,1\}^{m-t}$ with $k' = k - k_{max}$ and $\rho = \max(\epsilon + \delta, \epsilon \cdot 2^{t+1})$.

The construction of F:
- Given $x \in \{0,1\}^n$ compute $E(x)$ and let $E_1(x)$ denote the first t bits of $E(x)$ and $E_2(x)$ denote the remaining $m - t$ bits.
- Let $T = Samp(E_1(x))$.
- Let $x' = x_{[n]\setminus T}$. If $|x'| < n$, we pad it with zeros to get an n-bit long string.
- Let $y = E_2(x)$, Output x', y.

FIG. 1. *A seed obtainer for (n, k)-bit-fixing sources.*

Our construction appears in Figure 1. In words, given x, the seed obtainer first computes $E(x)$. It uses a part of $E(x)$ as the second output y and another part to sample a substring of x. It obtains the first output x' by erasing the sampled substring from x. We now state the main theorem of this section.

THEOREM 3.3 (construction of seed obtainers). *For every n and $k < n$, let Samp and E be as in Figure 1 (that is, $Samp : \{0,1\}^t \to P([n])$ is an $(n, k, k_{min}, k_{max}, \delta)$-sampler and $E : \{0,1\}^n \to \{0,1\}^m$ is a deterministic (k_{min}, ϵ)-bit-fixing source extractor). Then, $F : \{0,1\}^n \to \{0,1\}^n \times \{0,1\}^d$ defined in Figure 1 is a (k, k', ρ)-seed obtainer for $d = m - t$, $k' = k - k_{max}$ and $\rho = \max(\epsilon + \delta, \epsilon \cdot 2^{t+1})$.*

Proof of Theorem 3.3. In this section we prove Theorem 3.3. Let E be a bit-fixing source extractor and $Samp$ be a sampler which satisfy the requirements in Theorem 3.3. Let X be some (n, k)-bit-fixing source and let $S \subseteq [n]$ be the set of k good indices for X. We will use capital letters to denote the random variables which come up in the construction. We split $E(X)$ into two parts $(E_1(X), E_2(X)) \in \{0,1\}^t \times \{0,1\}^{m-t}$. For a string $a \in \{0,1\}^t$ we use T_a to denote $Samp(a)$ and T'_a to denote $[n] \setminus Samp(a)$. Given a string $x \in \{0,1\}^n$, we use x_a to denote x_{T_a} and x'_a to denote the n-bit string obtained by padding $x_{T'_a}$ to length n. Let $X' = X'_{E_1(X)}$ and $Y = E_2(X)$. Our goal is to show that the pair (X', Y) is close to a convex combination of pairs of distributions where the first component is a bit-fixing source and the second is independent and uniformly distributed.

DEFINITION 3.4. *We say that a string $a \in \{0,1\}^t$* correctly splits *X if $k_{min} \leq |S \cap T_a| \leq k_{max}$.*

Note that by the properties of the sampler, almost all strings a correctly split X. We start by showing that for every *fixed* a which correctly splits X the variables X'_a and $E(X)$ are essentially independent. Loosely speaking, this happens because we can argue that there are enough good bits in X_a and therefore the extractor can

extract randomness from X_a which is independent of the randomness in X'_a.

LEMMA 3.5. *For every fixed $a \in \{0,1\}^t$ which correctly splits X the pair of random variables $(X'_a, E(X))$ is ϵ-close to the pair $(X'_a \otimes U_m)$.*

Proof. Let $\ell = |Samp(a)|$. Given a string $\sigma \in \{0,1\}^\ell$ and a string $\sigma' \in \{0,1\}^{n-\ell}$, we define $[\sigma;\sigma']$ to be the n-bit string obtained by placing σ in the indices of T_a and σ' in the indices of T'_a. More formally, we denote the ℓ indices of T_a by $i_1 < i_2 < \cdots < i_\ell$ and the $n-\ell$ indices of T'_a by $i'_1 < i'_2 < \cdots < i'_{n-\ell}$. Given an $i \in T_a$, we define $index(i)$ to be the index j such that $i_j = i$, and equivalently, given $i \in T'_a$, we define $index'(i)$ to be the index j such that $i'_j = i$. The string $[\sigma;\sigma'] \in \{0,1\}^n$ is defined as follows:

$$[\sigma;\sigma']_i = \begin{cases} \sigma_{index(i)} & i \in T_a \\ \sigma'_{index'(i)} & i \in T'_a \end{cases}$$

Note that in this notation $X = [X_a; X'_a]$. We are interested in the distribution of the random variable $(X'_a, E(X)) = (X'_a, E([X_a; X'_a]))$. For every $b \in \{0,1\}^{n-\ell}$ we consider the event $\{X'_a = b\}$. Fix some $b \in \{0,1\}^{n-\ell}$ such that $\Pr[X'_a = b] > 0$. The distribution

$$(E(X)|X'_a = b) = (E([X_a; X'_a])|X'_a = b) = E([X_a; b]),$$

where the last equality follows because X_a and X'_a are independent and therefore X_a is not affected by fixing X'_a. Note that as a correctly splits X, the distribution $[X_a; b]$ is a bit-fixing source with at least k_{min} "good" bits. We conclude that for every $b \in \{0,1\}^{n-\ell}$ such that $\Pr[X'_a = b] > 0$ the distribution $(E(X)|X'_a = b)$ is ϵ-close to uniform. We now apply Lemma 2.5 with $A = X'_a$ and $B = E(X)$ and conclude that the pair $(X'_a, E(X))$ is ϵ-close to $(X'_a \otimes U_m)$. ∎

We now argue that if ϵ is small enough, then the pair $(X'_a, E_2(X))$ is essentially independent even when conditioning the probability space on the event $\{E_1(X) = a\}$.

LEMMA 3.6. *For every fixed $a \in \{0,1\}^t$ that correctly splits X, the distribution $((X'_a, E_2(X))|E_1(X) = a)$ is $\epsilon \cdot 2^{t+1}$-close to $(X'_a \otimes U_{m-t})$.*

Proof. First noting that the statement is meaningless unless $\epsilon < 2^{-t}$, we will assume w.l.o.g. that this is the case and then for every fixed $a \in \{0,1\}^t$ the event $\{E_1(X) = a\}$ occurs with nonzero probability as $E_1(X)$ is ϵ-close to uniform over $\{0,1\}^t$. The lemma will follow as a straightforward application of Lemma 2.6. We set $A = (X'_a, E_2(X))$, $B = E_1(X)$, and $A' = (X'_a, U_{m-t})$. We indeed have that (A, B) is ϵ-close to (A', U_t) and the lemma follows. ∎

We are now ready to prove Theorem 3.3.

Proof of Theorem 3.3. By the properties of the extractor we have that $E_1(X)$ is ϵ-close to uniform. It follows (by the properties of the sampler) that the probability that $E_1(X)$ correctly splits X is $1 - \eta$ for some η which satisfies $\eta \leq \epsilon + \delta$. We now consider the output random variable $R = (X', E_2(X))$. We need to express this random variable as a convex combination of independent distributions and a small error term. We set Q to be the distribution $(R|\text{"}E_1(X) \text{ doesn't correctly split } X\text{"})$. For every correctly splitting a we set R_a to be the distribution $(R|E_1(X) = a)$ and $\alpha_a = \Pr[E_1(X) = a]$. By our definition we have that indeed $R = \eta Q + \sum_a \alpha_a R_a$. For every a that correctly splits X we have that $R_a = ((X', E_2(X))|E_1(X) = a) = ((X'_{E_1(X)}, E_2(X))|E_1(X) = a) = ((X'_a, E_2(X))|E_1(X) = a)$. By Lemma 3.6 we have that R_a is $\epsilon \cdot 2^{t+1}$-close to $(X'_a \otimes U_{m-t})$. As a correctly splits X, we have that X'_a is an $(n, k - k_{max})$-bit-fixing source as required. Thus, we have shown that the distribution R_a is close to a convex combination of pairs of essentially independent distributions where the first is a bit-fixing source and the second is uniform. ∎

4. Extracting few bits for any k. The deterministic bit-fixing source extractor of Kamp and Zuckerman [17] works only for $k > \sqrt{n}$. However, their technique easily gives a deterministic bit-fixing source extractor that extracts very few bits ($\Omega(\log k)$ bits) from a bit-fixing source with arbitrarily small k. We will later use this extractor to construct a seed obtainer that will enable us to extract many more bits.

THEOREM 4.1. *For every $n > k \geq 100$ there is an explicit deterministic $(k, 2^{-\sqrt{k}})$-bit-fixing source extractor $E : \{0,1\}^n \to \{0,1\}^{(\log k)/4}$.*

For the proof, we need the following result, which is a very special case of Lemma 3.3 in [17].

LEMMA 4.2 (see [17, Lemma 3.3] for $\epsilon = 0$ and $d = 2$). *Let the graph G be an odd cycle with M vertices and second eigenvalue λ. Suppose we take a walk on G for n steps, starting from some fixed vertex v with the steps taken according to the symbols from an (n,k)-bit-fixing source X. Let Z be the distribution on the vertices at the end of the walk; then Z is $\left(\frac{1}{2}\lambda^k\sqrt{M}\right)$-close to the uniform distribution on $[M]$.*

To extract few bits from a bit-fixing source X, we will use the bits of X to conduct a random walk on a small cycle.

Proof of Theorem 4.1. We use the source string to take a walk on a cycle of size $\sqrt[4]{k}$ from a fixed vertex. The second eigenvalue of a d-cycle is $\cos(\frac{\pi}{d})$ [19, Ex. 11.1]. Using Lemma 4.2, we reach distance $\left(\cos\left(\frac{\pi}{\sqrt[4]{k}}\right)\right)^k k^{1/8}$ from uniform. By the Taylor expansion of cos, for $0 < x < 1$

$$\cos(x) < 1 - \frac{x^2}{2} + \frac{x^4}{24} < 1 - \frac{x^2}{4}.$$

Therefore

$$\left(\cos\left(\frac{\pi}{\sqrt[4]{k}}\right)\right)^k < \left(1 - \frac{\pi^2}{4\sqrt{k}}\right)^k$$

$$< \left(e^{-\frac{\pi^2}{4}}\right)^{\sqrt{k}} < 4^{-\sqrt{k}},$$

where the second-to-last inequality holds because $(1 - x) < e^{-x}$ for $0 < x < 1$. Therefore, we reach distance $4^{-\sqrt{k}}k^{1/8} \leq 2^{-\sqrt{k}}$. By outputting the final vertex's name we get $\frac{\log(k)}{4}$ bits with the same distance from uniform. ∎

5. Sampling and partitioning with a short seed. Let $S \subseteq [n]$ be some subset of size k. In this section we show how to use few random bits in order perform two related tasks.

Sampling: Generate a subset $T \subseteq [n]$ such that $|S \cap T|$ is in a prespecified interval $[k_{min}, k_{max}]$ (see Definition 2.1).

Partitioning: Partition $[n]$ into r distinct subsets $T_1, \ldots, T_r$ such that for every $1 \leq i \leq r$, $|S \cap T_i|$ is in a prespecified interval $[k_{min}, k_{max}]$. Needless to say, a partitioning scheme immediately implies a sampling scheme by concentrating on a single T_i.

In this section we present two constructions of such schemes. The first construction is used in our deterministic bit-fixing source extractor for $k > \sqrt{n}$. In this setup we can allow the sampler to use many random bits (say $n^{\Omega(1)}$ bits) and can have error $2^{-n^{\Omega(1)}}$.

LEMMA 5.1 (sampling with low error). *Fix any constants $0 < \gamma \leq 1/2$ and $\alpha > 0$. There exists a constant n' depending on α and γ, such that for any integers n, k satisfying $n > n'$ and $n^{1/2+\gamma} \leq k \leq n$, there exists an $(n, k, (n^{1/2+\gamma})/6, n^{1/2+\gamma}, 2^{-\Omega(\alpha \cdot n^{\gamma})})$-sampler $Samp : \{0,1\}^t \to P([n])$, where $t = \alpha \cdot n^{2\gamma}$.*

The second construction is used in our deterministic bit-fixing source extractor for small k. For that construction we require schemes that use only $\alpha \log k$ bits for some small constant $\alpha > 0$. The construction of Lemma 5.1 requires at least $\log n > \log k$ bits, which is too much. Instead, we use a different construction which has much larger error (e.g., $k^{-\Omega(1)}$).

LEMMA 5.2 (sampling with $O(\log k)$ bits). *Fix any constant $0 < \alpha < 1$. There exist constants $c > 0, 0 < b < 1$ and $1/2 < e < 1$ (all depending on α) such that for any $n \geq 16$ and $k \geq \log^c n$, we obtain an explicit $(n, k, k^e/2, 3 \cdot k^e, O(k^{-b}))$-sampler $Samp : \{0,1\}^t \to P([n])$, where $t = \alpha \cdot \log k$.*

LEMMA 5.3 (partitioning with $O(\log k)$ bits). *Fix any constant $0 < \alpha < 1$. There exist constants $c > 0, 0 < b < 1$ and $1/2 < e < 1$ (all depending on α) such that for any $n \geq 16$ and $k \geq \log^c n$, we can use $\alpha \cdot \log k$ random bits to explicitly partition $[n]$ into $m = \Omega(k^b)$ sets $T_1, \ldots, T_m$ such that for any $S \subseteq [n]$, where $|S| = k$,*

$$\Pr(\forall i, \quad k^e/2 \leq |T_i \cap S| \leq 3 \cdot k^e) \geq 1 - O(k^{-b}).$$

The first construction is based on "ℓ-wise independence," and the second is based on "almost 2-wise dependence" [20, 1, 15]. Sampling techniques based on ℓ-wise independence were first suggested by Bellare and Rompel [4]. However, this technique is not good enough in our setting and we use a different approach (which was also used in [27] with slightly different parameters). In Appendix A we explain the approach in detail, compare it to the approach of [4], and give full proofs of the lemmas above.

6. A seeded bit-fixing source extractor with a short seed. In this section we give a construction of a seeded bit-fixing source extractor that uses seed length $O(\log k)$ to extract $k^{\Omega(1)}$ bits as long as k is not too small. This seeded extractor is used as a component in our construction of deterministic extractors for bit-fixing sources.

THEOREM 6.1. *Fix any constant $0 < \alpha < 1$. There exist constants $c > 0$ and $0 < b < 1$ (both depending on α) such that for any $n \geq 16$ and $k \geq \log^c n$, there exists an explicit seeded (k, ϵ)-bit-fixing source extractor $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $d = \alpha \cdot \log k$, $m = \Omega(k^b)$, and $\epsilon = O(k^{-b})$.*

Proof. Let X be an (n, k)-bit-fixing source. Let $x = x_1, \ldots, x_n$ be a string sampled by X. The extractor E works as follows: We use the extractor seed y to construct a partition of the bits of x into m sets. Then we output the xor of the bits in each set. With high probability, each set will contain a good bit and therefore, with high probability, the output will be uniformly distributed.

More formally, let b and c be the constants from Lemma 5.3 when using the lemma with the parameter α.

E(x,y):

- We use the seed y to obtain a partition of $[n]$ into $m = \Omega(k^b)$ sets $T_1, \ldots, T_m$ using Lemma 5.3 with the parameter α.
- For $1 \leq i \leq m$, compute $z_i = \oplus_{j \in T_i} x_j$.
- Output $z = z_1, \ldots, z_m$.

We give the following detailed correctness proof although it is very straightforward.

Let $S \subseteq [n]$ be the set of good indices and let Z be the distribution of the output string z. We need to prove that Z is close to uniform. Let A be the event

$\{\forall i\ T_i \cap S \neq \emptyset\}$. That is, A is the "good" event in which all sets contain a random bit (and therefore in this case the output is uniform). Let A^c be the complement event; i.e., A^c is the event $\{\exists i\ T_i \cap S = \emptyset\}$. We decompose Z according to A and A^c:

$$Z = \Pr(A^c) \cdot (Z|A^c) + \Pr(A) \cdot (Z|A).$$

$(Z|A)$ is uniformly distributed. From Lemma 5.3, when $k \geq \log^c n$, $\Pr(A) \geq 1 - O(k^{-b})$. Therefore, by Lemma 2.4

$$Z \overset{O(k^{-b})}{\sim} U_m. \qquad \square$$

7. Deterministic extractors for bit-fixing sources. In this section, we compose the ingredients from previous sections to prove Theorems 1.3 and 1.4. Namely, given choices for a deterministic bit-fixing source extractor, sampler, and seeded bit-fixing source extractor, we use Theorems 3.2 and 3.3 to get a new deterministic bit-fixing source extractor. This works as follows: We "plug in" a deterministic extractor that extracts little randomness and a sampler into Theorem 3.3 to get a seed obtainer. We then "plug in" this seed obtainer and a seeded extractor into Theorem 3.2 to get a new deterministic extractor which extracts almost all of the randomness. It is convenient to express this composition as in the following theorem.

THEOREM 7.1. *Assume we have the following ingredients:*

- *an $(n, k, k_{min}, k_{max}, \delta)$-sampler $Samp : \{0,1\}^t \to P([n])$;*
- *a deterministic (k_{min}, ϵ^*)-bit-fixing source extractor $E^* : \{0,1\}^n \to \{0,1\}^{m'}$;*
- *a seeded $(k - k_{max}, \epsilon_1)$-bit-fixing source extractor $E_1 : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$,*

where $m' \geq d + t$. Then we construct a deterministic (k, ϵ)-bit-fixing source extractor $E : \{0,1\}^n \to \{0,1\}^m$, where $\epsilon = \epsilon_1 + 3 \cdot \max(\epsilon^* + \delta, \epsilon^* \cdot 2^{t+1})$.

Proof. We use *Samp* and E^* in Theorem 3.3 to get a $(k, k - k_{max}, \max(\epsilon^* + \delta, \epsilon^* \cdot 2^{t+1}))$-seed obtainer $F : \{0,1\}^n \to \{0,1\}^n \times \{0,1\}^{m'-t}$. Since $m' - t \geq d$, we can use F and E_1 in Theorem 3.2 to obtain a deterministic (k, ϵ)-bit-fixing source extractor $E : \{0,1\}^n \to \{0,1\}^m$, where $\epsilon = \epsilon_1 + 3 \cdot \max(\epsilon^* + \delta, \epsilon^* \cdot 2^{t+1})$. $\square$

We also require the following construction of a seeded extractor (which is in particular a seeded bit-fixing source extractor).

THEOREM 7.2 (see [26]). *For any n, k and $\epsilon > 0$, there exists a (k, ϵ)-extractor $Ext : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, where $m = k$ and $d = O(\log^2 n \cdot \log(1/\epsilon) \cdot \log k)$.*

7.1. An extractor for large k (proof of Theorem 1.3). To prove Theorem 1.3, we first state results about the required ingredients and then use the ingredients in Theorem 7.1.

We use the deterministic bit-fixing source extractor of Kamp and Zuckerman [17]. Loosely speaking, the following theorem states that when $k \gg \sqrt{n}$, we can deterministically extract a polynomial fraction of the randomness with an exponentially small error.

THEOREM 7.3 (see [17]). *Fix any integers n, k such that $k = b \cdot n^{1/2+\gamma}$ for some $b > 0$ and $0 < \gamma \leq 1/2$. There exists a constant $c > 0$ (*not *depending on any of the parameters) such that there exists an explicit deterministic (k, ϵ^*)-bit-fixing source extractor $E^* : \{0,1\}^n \to \{0,1\}^m$, where $m = cb^2 \cdot n^{2\gamma}$ and $\epsilon^* = 2^{-m}$.*

Using the theorem above we can obtain a seed of length $O(n^{2\gamma})$. This means that we can afford this many bits for our sampler and seeded bit-fixing source extractor. We use the sampler based on ℓ-wise independence from Lemma 5.1. We use the seeded extractor of [26] (Theorem 7.2) which we now restate in the following form.

COROLLARY 7.4. *Fix any constants $0 < \gamma \leq 1/2$ and $\alpha > 0$. There exists a constant n' depending on γ such that for any integers n, k satisfying $n > n'$ and $k \leq n$ there exists a (k, ϵ_1)-extractor $E_1 : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, where $m = k$, $d = \alpha \cdot n^{2\gamma}$, and $\epsilon_1 = 2^{-\Omega(\alpha \cdot n^\gamma)}$.*

Proof. We use the extractor of Theorem 7.2. We need $d = c_1 \cdot (\log^3 n \cdot \log(1/\epsilon_1))$ random bits for some constant $c_1 > 0$. We want to use at most $\alpha \cdot n^{2\gamma}$ random bits. We get the inequality $\alpha \cdot n^{2\gamma} \geq c_1 \cdot \log^3 n \cdot \log(1/\epsilon_1)$. Equivalently, $\epsilon_1 \geq 2^{-\frac{\alpha \cdot n^{2\gamma}}{c_1 \cdot \log^3 n}}$. So for a large enough n (depending on γ), we can take $\epsilon_1 = 2^{-\frac{\alpha \cdot n^\gamma}{c_1}} = 2^{-\Omega(\alpha \cdot n^\gamma)}$. ∎

We now compose the ingredients from Theorem 7.3, Lemma 5.1, and Corollary 7.4 to prove Theorem 1.3. The composition is a bit cumbersome in terms of the different parameters. The main issue is that when $k = n^{1/2+\gamma}$, the deterministic extractor of Kamp and Zuckerman extracts $\Omega(n^{2\gamma})$ random bits; and this is enough to use as a seed for a sampler and seeded extractor (that extracts all the randomness) with error $2^{-\Omega(n^\gamma)}$.

Proof of Theorem 1.3. Let c be the constant in Theorem 7.3. We use Theorem 7.1 with the following ingredients:

- the $(n, k, (n^{1/2+\gamma})/6, n^{1/2+\gamma}, \delta = 2^{-\Omega(n^\gamma)})$-sampler $Samp : \{0,1\}^t \to P([n])$ from Lemma 5.1, where $t = (c/72)n^{2\gamma}$;
- the deterministic $((n^{1/2+\gamma})/6, \epsilon^* = 2^{-m'})$-bit-fixing source extractor $E^* : \{0,1\}^n \to \{0,1\}^{m'}$ from Theorem 7.3, where $m' = (c/36)n^{2\gamma}$;
- the $(k - n^{1/2+\gamma}, \epsilon_1 = 2^{-\Omega(n^\gamma)})$-extractor $E_1 : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ from Corollary 7.4 with $d \leq (c/72)n^{2\gamma}$ and $m = k - n^{1/2+\gamma}$.

Note that all three objects exist for a large enough n depending only on γ (c is a universal constant). Note that $m' \geq t + d$. Therefore, applying Theorem 7.1, we get a deterministic (k, ϵ)-bit-fixing source extractor $E : \{0,1\}^n \to \{0,1\}^m$, where $m = k - n^{1/2+\gamma}$ and

$$\epsilon = \epsilon_1 + 3 \cdot \max(\epsilon^* + \delta, \epsilon^* \cdot 2^{t+1})$$

$$= 2^{-\Omega(n^\gamma)} + 3 \cdot \max\left(2^{-(c/36)n^{2\gamma}} + 2^{-\Omega(n^\gamma)}, 2^{-(c/36)n^{2\gamma}} \cdot 2^{(c/72)n^{2\gamma}+1}\right)$$

$$= 2^{-\Omega(n^\gamma)} + 3 \cdot \max\left(2^{-\Omega(n^\gamma)}, 2^{-(c/72)n^{2\gamma}+1}\right) = 2^{-\Omega(n^\gamma)}$$

(for a large enough n depending on γ). ∎

7.2. An extractor for small k (proof of Theorem 1.4). To prove Theorem 1.4 we need a deterministic bit-fixing source extractor for $k < \sqrt{n}$. We use the extractor of Theorem 4.1. We prove the theorem in two steps. First, we use Theorem 7.1 to convert the initial extractor into a deterministic bit-fixing source extractor that extracts more bits. We then apply Theorem 7.1 again to obtain a deterministic bit-fixing source extractor which extracts almost all bits.

The following lemma implements the first step and shows how to extract a polynomial fraction of the randomness with a polynomially small error, whenever $k \geq \log^c n$ for some constant c.

LEMMA 7.5. *There exist constants $c, b > 0$ such that for any $k \geq \log^c n$ and large enough n, there exists an explicit deterministic (k, k^{-b})-bit-fixing source extractor $E : \{0,1\}^n \to \{0,1\}^m$, where $m = k^{\Omega(1)}$.*

Proof. Roughly speaking, the main issue is that we can get $\Omega(\log k)$ random bits using the deterministic extractor of Theorem 4.1. We will need $c_1 \cdot \log\log n$ random bits to use the sampler of Lemma 5.2 and the seeded extractor of Theorem 6.1 (for some constant c_1). Thus, when $k \geq \log^c n$ for large enough c, we will have enough bits.

Formally, we use Theorem 7.1 with the following ingredients:

- the $(n, k, k^e/2, 3 \cdot k^e, \delta = k^{-\Omega(1)})$-sampler $Samp : \{0,1\}^t \to P([n])$ from Lemma 5.2, where $t = \log k/32$ and $e > 1/2$ is the constant from that lemma;
- the deterministic $(k^e/2, \epsilon^* = 2^{-\sqrt{k^e/2}})$-bit-fixing source extractor $E^* : \{0,1\}^n \to \{0,1\}^{m'}$ from Theorem 4.1, where $m' = \log(k^e/2)/4$;
- the seeded $(k - 3 \cdot k^e, \epsilon_1 = (k - 3 \cdot k^e)^{-\Omega(1)})$-bit-fixing source extractor $E_1 : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ from Theorem 6.1 with $d = \log k/32$ and $m = (k - 3 \cdot k^e)^{\Omega(1)}$.

Note that all three objects exist for $k \geq \log^c n$ for some constant c and large enough n. Assume that n is large enough so that $k \geq \log^c n \geq 2$. To use Theorem 7.1 we need to check that $m' \geq t + d$: Indeed, $m' = \log(k^e/2)/4 \geq \log k/16 = t + d$ (where we used $e > 1/2$, as stated in Lemma 5.2). Applying Theorem 7.1, we get a deterministic (k, ϵ)-bit-fixing source extractor $E : \{0,1\}^n \to \{0,1\}^m$. Notice that for large enough n: $\epsilon_1 = k^{-\Omega(1)}$, therefore

$$\epsilon = \epsilon_1 + 3 \cdot \max(\epsilon^* + \delta, \epsilon^* \cdot 2^{t+1})$$

$$= k^{-\Omega(1)} + 3 \cdot \max\left(2^{-\sqrt{k^e/2}} + k^{-\Omega(1)}, 2^{-\sqrt{k^e/2}} \cdot 2^{\log k/32+1}\right) = k^{-\Omega(1)}$$

(for a large enough n). Also, $m = (k - 3 \cdot k^e)^{\Omega(1)} = k^{\Omega(1)}$ (for a large enough n) and thus we get the required parameters. ∎

We now compose the ingredients from Lemmas 5.2 and 7.5 and Theorem 7.2 to prove Theorem 1.4. The composition is a bit cumbersome in terms of the different parameters. The main issue is that we can extract $k^{\Omega(1)}$ random bits using the deterministic extractor of Lemma 7.5. We want $\log^5 n$ random bits to use the seeded extractor of Theorem 7.2. Thus, when $k \geq \log^c n$ for large enough c, we will have enough bits.

Proof of Theorem 1.4. Let b be the constant in Lemma 7.5. We use Theorem 7.1 with the following ingredients:

- the $(n, k, k^e/2, 3 \cdot k^e, \delta = k^{-\Omega(1)})$-sampler $Samp : \{0,1\}^t \to P([n])$ from Lemma 5.2, where $t = (b/2)\log k$ and $e > 1/2$ is the constant from that lemma;
- the deterministic $(k^e/2, \epsilon^* = (k^e/2)^{-b})$-bit-fixing source extractor $E^* : \{0,1\}^n \to \{0,1\}^{m'}$ from Lemma 7.5, where $m' = (k^e/2)^{\Omega(1)}$;
- the $(k - 3 \cdot k^e, \epsilon_1 = 1/n)$-extractor $E_1 : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ from Theorem 7.2 with $d \leq \log^5 n$ and $m = (k - 3 \cdot k^e)$.

Note that all three objects exist for $k \geq \log^c n$ for some constant c and for large enough n. To use Theorem 7.1 we need to check that $m' \geq t + d$: Note that $m' = k^{\Omega(1)}$. We take c large enough so that for large enough n, $m'/2 > \log^5 n$ and $m'/2 > (b/2)/\log k$. So for such n

$$m' \geq \log^5 n + (b/2)\log k \geq d + t.$$

Applying Theorem 7.1, we get a deterministic (k, ϵ)-bit-fixing source extractor $E : \{0,1\}^n \to \{0,1\}^m$, where

$$\epsilon = \epsilon_1 + 3 \cdot \max(\epsilon^* + \delta, \epsilon^* \cdot 2^{t+1})$$

$$= 1/n + 3 \cdot \max\left((k^e/2)^{-b} + k^{-\Omega(1)}, 2 \cdot (k^e/2)^{-b} \cdot k^{b/2}\right) = k^{-\Omega(1)}$$

(for large enough n). Since $m = k - O(k^e)$, where $1/2 < e < 1$, we are done. □

8. Discussion and open problems. We give explicit constructions of deterministic bit-fixing source extractors that extract almost all the randomness. However, we achieve the rather large error $\epsilon = k^{-\Omega(1)}$ in the case that $k < \sqrt{n}$. We now explain why this happens and suggest how to reduce the error. Recall that in this case our final extractor is based on an initial extractor that extracts only $m = O(\log k)$ bits. When transforming the initial extractor into the final extractor we use the output bits of the initial extractor as a seed for an averaging sampler. The error parameter δ of an averaging sampler has to be larger than 2^{-m} and as this error is "inherited" by the final extractor we can only get an error of about $1/k$. A natural way to improve our result is to find a better construction for the initial extractor.

Some applications of deterministic bit-fixing source extractors in adaptive settings of exposure-resilient cryptography require extractors with $\epsilon \ll 2^{-m}$. We do not achieve this goal (even in our first construction that has relatively small error), unless we artificially shorten the output. Suppose one wants to extract $m = k - u$ bits (for some parameter u). It is interesting to investigate how small the error can be as a function of u. We point out that the existential nonexplicit result achieves error $\epsilon \geq 2^{-u}$ and thus cannot achieve $\epsilon < 2^{-m}$ when $m \geq k/2$. We remark that for bit-fixing sources we have examples of settings where the nonexplicit result is not optimal. For example when $m = 1$, the xor-extractor is errorless (see also [11]). Given the discussion above, we find it interesting to achieve $m = \Omega(k)$ with $\epsilon = 2^{-\Omega(k)}$ for every choice of k.

Appendix A. Sampling and partitioning. In this section we give constructions of samplers and prove Lemmas 5.1, 5.2, and 5.3.

A.1. Sampling using ℓ-wise independence. Bellare and Rompel [4] gave a sampler construction based on ℓ-wise independent variables. We use a twist on their method: Suppose we are aiming to hit k/r bits when given a subset S of size k. We generate ℓ-wise independent variables $Z_1, \ldots, Z_n \in [r]$ and define $T = \{i | Z_i = 1\}$. It follows that with high probability $S \cap T$ is of size approximately k/r. This is stated formally in the following lemma. (We explain the difference between this method and that of [4] in Remark A.3.)

LEMMA A.1. *For every integer n, k, r, t such that $r \leq k \leq n$ and $6 \log n \leq t \leq \frac{k \log n}{20r}$ there is an explicit $(n, k, \frac{1}{2} \cdot \frac{k}{r}, 3 \cdot \frac{k}{r}, 2^{-\Omega(t/\log n)})$-sampler which uses a seed of t random bits.*

Before proving this lemma we show that Lemma 5.1 is a special case.

Proof of Lemma 5.1. We use Lemma A.1 with the parameters n, k, and $r = \frac{3k}{n^{1/2+\gamma}}$, $t = \alpha \cdot n^{2\gamma}$. We need to check that $6 \log n \leq t \leq \frac{k \log n}{20r}$. Clearly, $t \geq 6 \log n$ (for a large enough n depending on α and γ). On the other hand,

$$\frac{k \log n}{20r} = \frac{n^{1/2+\gamma} \log n}{60} \geq \alpha \cdot n^{2\gamma} = t$$

(for a large enough n depending on α and γ). Thus, applying Lemma A.1, we get an $(n, k, k/2r, 3k/r, \delta)$-sampler $Samp : \{0,1\}^t \to P([n])$, where

$$\delta = 2^{-\Omega(t/\log n)} = 2^{-\Omega(\alpha \cdot n^{2\gamma}/\log n)} = 2^{-\Omega(\alpha \cdot n^{\gamma})}$$

(for a large enough n depending on α and γ). □

We need the following tail-inequality for ℓ-wise independent variables due to Bellare and Rompel [4].

THEOREM A.2 (see [4]). *Let $\ell \geq 6$ be an even integer. Suppose that $X_1, \ldots, X_n$ are ℓ-wise independent random variables taking values in $[0,1]$. Let $X = \sum_{1 \leq i \leq n} X_i$ and $\mu = E(X)$, and let $A > 0$. Then*

$$\Pr[|X - \mu| \geq A] \leq 8 \left(\frac{\ell\mu + \ell^2}{A^2} \right)^{\ell/2}.$$

We now prove Lemma A.1.

Proof of Lemma A.1. Let ℓ be the largest even integer such that $\ell \log n \leq t$, and let $q = \lfloor \log r \rfloor \leq \log n$. There are constructions which use $\ell \log n \leq t$ random bits to generate n random variables $Z_1 \ldots, Z_n \in \{0,1\}^q$ that are ℓ-wise independent [9]. The sampler generates such random variables. Let $a \in \{0,1\}^q$ be some fixed value. We define a random variable $T = \{i | Z_i = a\}$. Let $S \subseteq [n]$ be some subset of size k. For $1 \leq i \leq n$ we define a boolean random variable X_i such that $X_i = 1$ if $Z_i = a$. Let $X = |S \cap T| = \sum_{i \in S} X_i$. Note that $\mu = E(X) = k/2^q$ and that the random variables $X_1, \ldots, X_n$ are ℓ-wise independent. Applying Theorem A.2 with $A = k/2r$ we get that

$$\Pr[|X - \mu| \geq A] \leq 8 \left(\frac{\ell k/2^q + \ell^2}{A^2} \right)^{\ell/2}.$$

Note that

$$\{|X - \mu| < A\} \subseteq \left\{ \frac{k}{2^q} - A < X < \frac{k}{2^q} + A \right\} \subseteq \left\{ \frac{k}{r} - A < X < \frac{2k}{r} + A \right\}$$

$$\subseteq \{k_{min} \leq X \leq k_{max}\}$$

for $k_{min} = k/2r$ and $k_{max} = 3k/r$. Note that $\ell \leq \frac{t}{\log n} \leq \frac{k}{20r}$. We conclude that

$$\Pr[k_{min} \leq |S \cap T| \leq k_{max}] \geq 1 - 8 \left(\frac{\ell \frac{k}{2^q} + \ell^2}{(\frac{k}{2r})^2} \right)^{\ell/2} \geq 1 - 8 \left(\frac{4r^2(\frac{2\ell k}{r} + \frac{\ell k}{20r})}{k^2} \right)^{\ell/2}$$

$$\geq 1 - 8 \left(\frac{10\ell r}{k} \right)^{\ell/2} \geq 1 - 2^{-(\ell/2+3)} \geq 1 - 2^{-\Omega(t/\log n)}. \quad \square$$

Remark A.3. We remark that this construction is different from the common way of using ℓ-wise independence for sampling [4]. The more common way is to take n/r random variables $V_1, \ldots, V_{n/r} \in [n]$ which are ℓ-wise independent and sample the multiset $T = \{V_1, \ldots, V_{n/r}\}$. The expected size of the multiset $|S \cap T|$ is k/r, and one gets the same probability of success $\delta = 2^{-\Omega(\ell)}$ by the tail-inequality of [4]. The two methods require roughly the same number of random bits. Nevertheless, the method of Lemma A.1 has the following advantages:

- It can also be used for partitioning.
- The method used in Lemma A.1 guarantees that T is a set whereas the standard method may produce a multiset.

- The method used in Lemma A.1 can be derandomized and use many fewer bits (at least for small r and large δ). More precisely, suppose that $r \leq \log n$ and say $\ell = 2$. In this range of parameters, one can use $O(\log \log n)$ random bits to generate n variables $Z_1, \ldots, Z_n \in \{0,1\}^{\log r}$ which are $(1/\log n)$-close to being pairwise independent. Thus, the same technique can be used to construct more randomness-efficient samplers (at the cost of having larger error parameter δ). We use this idea in section A.2. We remark that in the case of the standard method no savings can be made as it requires variables Z_i over $\{0,1\}^{\log n}$ and even sampling one such variable requires $\log n$ random bits.

A.2. Sampling and partitioning using fewer bits. We now derandomize the construction of Lemma A.1 to give schemes which use only $O(\log k)$ bits and prove Lemmas 5.2 and 5.3. These two lemmas follow from the following more general lemma.

LEMMA A.4. *Fix any integer $n \geq 16$. Let k be an integer such that $k \leq n$. Let r satisfy $r \leq k$. Let r' be the power of 2 that satisfies $(1/2)r < r' \leq r$. Let $\epsilon > 0$ satisfy $1/kr \leq \epsilon \leq 1/8r$. We can use $7 \log r + 3(\log \log n + \log(1/\epsilon))$ random bits to explicitly partition $[n]$ into r' sets $T_1, \ldots, T_{r'}$ such that for any $S \subseteq [n]$ where $|S| = k$*

$$\Pr(\forall i, \quad k/2r \leq |T_i \cap S| \leq 3k/r) \geq 1 - O(\epsilon \cdot r^3).$$

We prove Lemma A.4 in the next section. We now explain how the two lemmas follow from Lemma A.4.

Proof of Lemma 5.3. Set $b = \alpha/38$. Use Lemma A.4 with the parameters $r = k^b$ and $\epsilon = k^{-4b}$ to obtain a partition $T_1, \ldots, T_{r'}$ of $[n]$, where $(1/2)r < r' \leq r$ is a power of 2.

To use Lemma A.4 with these parameters we need $7 \log r + 3(\log \log n + \log(1/\epsilon)) = 7 \log k^b + 3(\log \log n + \log k^{4b})$ random bits.

We want to use at most $\alpha \cdot \log k$ bits.

Set $c = 6/\alpha$. Since we assume that $k \geq \log^c n$,

$$(\alpha/2) \log k \geq (\alpha/2)(6/\alpha) \log \log n = 3 \log \log n.$$

So now we need

$$(\alpha/2) \log k \geq 7 \log k^b + 3 \log k^{4b} = b(7 + 12) \log k$$

or, equivalently,

$$b \leq \alpha/38.$$

Set $e = 1 - b$. So $k/2r = k^e/2$ and $3k/r = 3 \cdot k^e$. Note that $e > 1/2$ as required. Using Lemma A.4,

$$\Pr(\forall i, \quad k^e/2 \leq |T_i \cap S| \leq 3 \cdot k^e) \geq 1 - O(\epsilon \cdot r^3) = 1 - O(k^{-b}). \qquad \square$$

Lemma 5.2 easily follows from Lemma 5.3.

Proof of Lemma 5.2. Use Lemma 5.3 with the parameters n, k, and α to obtain a partition of $[n]$ $T_1, \ldots, T_m$ and take T_1 as the sample. It is immediate that the required parameters are achieved. $\square$

Proof of Lemma A.4. The sampler construction in Lemma A.1 relied on random variables $Z_1, \ldots, Z_n \in [r]$ which are ℓ-wise independent. We now show that we can derandomize this construction and get a (weaker) sampler by using $Z_1, \ldots, Z_n$ which are only *pairwise ϵ-dependent*. Naor and Naor [20] (and later Alon et al. [1]) gave constructions of such variables using very few random bits. This allows us to reduce the number of random bits required for sampling and partitioning.

The following definition formalizes a notion of limited independence, slightly more general than the one discussed above.

DEFINITION A.5 (ℓ-wise ϵ-dependent variables). *Let D be a distribution. We say that the random variables $Z_1, \ldots, Z_n$ are ℓ-wise ϵ-dependent according to D if for every $M \subseteq [n]$ such that $|M| \leq \ell$ the distribution Z_M (that is, the joint distribution of the Z_i's such that $i \in M$) is ϵ-close to the distribution $D^{\otimes |M|}$, i.e., the distribution of $|M|$ independent random variables chosen according to D. We sometimes omit D when it is the uniform distribution.*

Random bit variables $B_1, \ldots, B_n$ are ℓ-wise ϵ-dependent with mean p if they are ℓ-wise ϵ-dependent according to the distribution $D = (1-p, p)$ on $\{0, 1\}$.

We need two properties about ℓ-wise ϵ-dependent variables: that they can be generated using very few random bits and that their sum is concentrated around the expectation. The first property is proved in Lemma A.7 and the second in Lemma A.8.

The following theorem states that ℓ-wise ϵ-dependent bit variables can be generated by very few random bits.

THEOREM A.6 (see [1]).[6] *For any $n \geq 16$, $\ell \geq 1$, and $0 < \epsilon < 1/2$, ℓ-wise ϵ-dependent bits $B_1, \ldots, B_n$ can be generated using $3(\ell + \log\log n + \log(1/\epsilon))$ truly random bits.*

We can generate pairwise ϵ-dependent variables in larger domains using ℓ-wise ϵ-dependent bit variables.[7]

LEMMA A.7. *Let $r < n$ be a power of 2. For any $n \geq 16$ and $0 < \epsilon < 1/2$, we can generate pairwise ϵ-dependent variables $Z_1, \ldots, Z_n \in [r]$ using $7 \log r + 3(\log\log n + \log(1/\epsilon))$ truly random bits.*

Proof. Using Theorem A.6, we generate $2 \log r$-wise ϵ-dependent bit variables $B_1, \ldots, B_{n \log r}$ using $3(2 \log r + \log\log(n \log r) + \log(1/\epsilon)) \leq 7 \log r + 3(\log\log n + \log(1/\epsilon))$ bits. We partition the B_i's into n blocks of size $\log r$ and interpret the ith block as a value $Z_i \in [r]$.

The joint distribution of the bits in any block or two blocks is ϵ-close to uniform. Therefore, the Z_i's are pairwise ϵ-dependent. ∎

In the following lemma, we use Chebyshev's inequality to show that the sum of pairwise ϵ-dependent bit variables is close to its expectation with high probability.

LEMMA A.8. *Let p satisfy $0 < p < 1$. Let $\epsilon > 0$ satisfy $p/k \leq \epsilon \leq p/4$. Let $B_1, \ldots, B_k$ be pairwise ϵ-dependent bit variables with mean p. Let $B = \sum_{i=1}^{k} B_i$.*

Then

$$\Pr(|B - pk| > pk/2) = O(\epsilon/p^2).$$

Proof. Using linearity of expectation we get $|E(B) - pk| \leq \epsilon k$.

[6] The theorem is stated a bit differently and only for odd ℓ in [1], but this form is easily deduced from Theorem 3 in that paper, observing that $(\ell+1)$-wise ϵ-dependence implies ℓ-wise ϵ-dependence.

[7] Actually, a construction of such (and more general types of) variables already appears in [15].

Therefore

$$\Pr(|B - pk| > pk/2) \leq \Pr(|B - E(B)| > pk/2 - \epsilon k).$$

Thus it is enough to bound

$$\Pr(|B - E(B)| > pk/2 - \epsilon k).$$

Fix any $i, j \in [k]$ where $i \neq j$. The covariance of B_i and B_j will be small since they are almost independent:

$$\operatorname{cov}(B_i, B_j) = E(B_i \cdot B_j) - E(B_i)E(B_j)$$

$$= \Pr(B_i = 1; B_j = 1) - \Pr(B_i = 1)\Pr(B_j = 1)$$

$$\leq (p^2 + \epsilon) - (p - \epsilon)^2 = (1 + 2p - \epsilon)\epsilon \leq 3\epsilon$$

(where the second equality is because B_i and B_j are bit variables).

Therefore, the variance of B won't be too large:

$$\operatorname{Var}(B) = \sum_i \operatorname{Var}(B_i) + \sum_{i \neq j} \operatorname{cov}(B_i, B_j) \leq (p + \epsilon)k + 3\epsilon k^2 \leq pk + 4\epsilon k^2.$$

Therefore, by Chebyshev's inequality

$$\Pr(|B - E(B)| > pk/2 - \epsilon k) < \frac{pk + 4\epsilon k^2}{(pk/2 - \epsilon k)^2}$$

we required that $\epsilon \leq p/4$ and therefore

$$\leq \frac{pk + 4\epsilon k^2}{(pk/4)^2} = O(1/pk) + O(\epsilon/p^2) = O(\epsilon/p^2)$$

(where the last equality follows by the requirement that $\epsilon \geq p/k$). □

Now we can easily prove Lemma A.4.

Proof of Lemma A.4. Let r' be the power of 2 in the statement of the lemma. Using Lemma A.7, we generate pairwise ϵ-dependent $Z_1, \ldots, Z_n \in [r']$. For $1 \leq i \leq r'$, we define $T_i = \{j | Z_j = i\}$.

Assume, w.l.o.g., that $S = \{1, \ldots, k\}$.

Given $i \in [r']$, define the bit variables $B_1, \ldots, B_k$ by $B_j = 1 \Leftrightarrow Z_j = i$.

It is easy to see that the B_j's are pairwise 2ϵ-dependent with mean $1/r'$.

Define $C_i = \sum_{j=1}^{k} B_j$. Note that $C_i = |T_i \cap S|$.

Notice that $1/r'$ and 2ϵ satisfy the requirements in Lemma A.8.

Using Lemma A.8,

$$\Pr(|C_i - k/r'| > k/2r') = O(\epsilon \cdot (r')^2) = O(\epsilon \cdot r^2).$$

Using the union bound,

$$\Pr(\exists i \text{ s.t. } |C_i - k/r'| > k/2r') = O(\epsilon \cdot r^3).$$

Thus, we can obtain a partition $T_1, \ldots, T_{r'}$ of $[n]$ such that, with probability at least $1 - O(\epsilon \cdot r^3)$,

$$\forall i, \quad k/2r' \leq |T_i \cap S| \leq 3k/2r',$$

which implies that with at least the same probability,

$$\forall i, \quad k/2r \leq |T_i \cap S| \leq 3k/r. \qquad \square$$

Acknowledgments. The third author is grateful to David Zuckerman for very helpful discussions. We are grateful to Oded Goldreich and to the anonymous referees for helpful comments.

REFERENCES

[1] N. Alon, O. Goldreich, J. Hastad, and R. Peralta, *Simple constructions of almost k-wise independent random variables*, Random Structures Algorithms, 3 (1992), pp. 289–304.

[2] B. Barak, R. Impagliazzo, and A. Wigderson, *Extracting randomness from few independent sources*, in Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, 2004, pp. 384–393.

[3] B. Barak, G. Kindler, R. Shaltiel, B. Sudakov, and A. Wigderson, *Simulating independence: New constructions of condensers, Ramsey graphs, dispersers, and extractors*, in Proceedings of the 37th Annual ACM Symposium on Theory of Computing, 2005, pp. 1–10.

[4] M. Bellare and J. Rompel, *Randomness-efficient oblivious sampling*, in Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science, 1994, pp. 276–287.

[5] M. Ben-Or and N. Linial, *Collective coin flipping*, in Advances in Computing Research 5, JAI Press, Greenwich, CT, 1989, pp. 91–116.

[6] M. Blum, *Independent unbiased coin flips from a correlated biased source: A finite state Markov chain*, in Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science, 1984, pp. 425–433.

[7] V. Boyko, *On the security properties of OAEP as an all-or-nothing transform*, in Proceedings of the 19th International Advances in Cryptology Conference (CRYPTO '99), 1999, pp. 503–518.

[8] R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, and A. Sahai, *Exposure-resilient functions and all-or-nothing transforms*, in Advances in Cryptology—EUROCRYPT 2000, Lecture Notes in Comput. Sci. 1807, Springer-Verlag, Berlin, 2000, pp. 453–469.

[9] I. L. Carter and M. N. Wegman, *Universal classes of hash functions*, in Proceedings of the 9th Annual ACM Symposium on Theory of Computing, 1977, pp. 106–112.

[10] B. Chor and O. Goldreich, *Unbiased bits from sources of weak randomness and probabilistic communication complexity*, SIAM J. Comput., 17 (1988), pp. 230–261.

[11] B. Chor, O. Goldreich, J. Hastad, J. Friedman, S. Rudich, and R. Smolensky, *The bit extraction problem or t-resilient functions*, in Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science, 1985, pp. 396–407.

[12] A. Cohen and A. Wigderson, *Dispersers, deterministic amplification, and weak random sources*, in Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, 1989, pp. 14–25.

[13] Y. Dodis, *Exposure-Resilient Cryptography*, Ph.D. thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, 2000.

[14] Y. Dodis, A. Sahai, and A. Smith, *On perfect and adaptive security in exposure-resilient cryptography*, in Advances in Cryptyology—EUROCRYPT 2001, Lecture Notes in Comput. Sci. 2045, Springer-Verlag, Berlin, 2001, pp. 299–322.

[15] S. Even, O. Goldreich, M. Luby, N. Nisan, and B. Velickovic, *Efficient approximation of product distributions*, Random Structures Algorithms, 13 (1998), pp. 1–16.

[16] O. Goldreich, *A Sample of Samplers—A Computational Perspective on Sampling*, Electronic Colloquium on Computational Complexity (ECCC), 1997.

[17] J. Kamp and D. Zuckerman, *Deterministic extractors for bit-fixing sources and exposure-resilient cryptography*, in Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, 2003, pp. 92–101.

[18] R. Lipton and N. Vishnoi, *manuscript*, 2004.

[19] L. Lovasz, *Combinatorial Problems and Exercises*, North–Holland, Amsterdam, 1979.

[20] J. Naor and M. Naor, *Small-bias probability spaces: Efficient constructions and applications*, SIAM J. Comput., 22 (1993), pp. 838–856.

[21] N. Nisan, *Extracting randomness: How and why—A survey*, in Proceedings of the 11th Annual IEEE Conference on Computational Complexity, 1996, pp. 44–58.

[22] N. Nisan and A. Ta-Shma, *Extracting randomness: A survey and new constructions*, J. Comput. System Sci., 58 (1999), pp. 148–173.

[23] N. Nisan and D. Zuckerman, *Randomness is linear in space*, J. Comput. System Sci., 52 (1996), pp. 43–52.

[24] J. Radhakrishnan and A. Ta-Shma, *Bounds for dispersers, extractors, and depth-two superconcentrators*, SIAM J. Discrete Math., 13 (2000), pp. 2–24.

[25] R. Raz, *Extractors with weak random seeds*, in Proceedings of the 37th Annual ACM Symposium on Theory of Computing, 2005, pp. 11–20.
[26] R. Raz, O. Reingold, and S. Vadhan, *Extracting all the randomness and reducing the error in Trevisan's extractors*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing, 1999, pp. 149–158.
[27] O. Reingold, R. Shaltiel, and A. Wigderson, *Extracting randomness via repeated condensing*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, 2000, pp. 149–158.
[28] R. Rivest, *All-or-nothing encryption and the package transform*, in Fast Software Encryption, 4th International Workshop, FSE'97, Lecture Notes in Comput. Sci. 1267, Springer-Verlag, Berlin, 1997, pp. 210–218.
[29] M. Santha and U. V. Vazirani, *Generating quasi-random sequences from semi-random sources*, J. Comput. System Sci., 33 (1986), pp. 75–87.
[30] R. Shaltiel, *Recent developments in explicit constructions of extractors*, Bull. Eur. Assoc. Theor. Comput. Sci. 77 (2002), pp. 67–95.
[31] L. Trevisan and S. Vadhan, *Extracting randomness from samplable distributions*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, 2000, pp. 32–42.
[32] S. Vadhan, *Randomness extractors and their many guises*, in Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002, pp. 9–12.
[33] S. Vadhan, *Constructing locally computable extractors and cryptosystems in the bounded-storage model*, J. Cryptology, 17 (2004), pp. 43–77.
[34] U. Vazirani, *Efficient considerations in using semi-random sources*, in Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, pp. 160–168.
[35] U. Vazirani, *Strong communication complexity or generating quasi-random sequences from two communicating semi-random sources*, Combinatorica, 7 (1987), pp. 375–392.
[36] U. V. Vazirani and V. V. Vazirani, *Random polynomial time is equal to semi-random polynomial time*, in Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science, 1985, pp. 417–428.
[37] J. von Neumann, *Various techniques used in connection with random digits*, NBS Appl. Math Ser., 12 (1951), pp. 36–38.
[38] D. Zuckerman, *General weak random sources*, in Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science, 1990, pp. 534–543.
[39] D. Zuckerman, *Simulating BPP using a general weak random source*, Algorithmica, 16 (1996), pp. 367–391.

SIAM J. COMPUT.
Vol. 36, No. 4, pp. 1095–1118

EXTRACTING RANDOMNESS USING FEW INDEPENDENT SOURCES*

BOAZ BARAK†, RUSSELL IMPAGLIAZZO‡, AND AVI WIGDERSON§

Abstract. In this work we give the first deterministic extractors from a constant number of weak sources whose entropy rate is less than 1/2. Specifically, for every $\delta > 0$ we give an explicit construction for extracting randomness from a constant (depending polynomially on $1/\delta$) number of distributions over $\{0,1\}^n$, each having min-entropy δn. These extractors output n bits that are 2^{-n} close to uniform. This construction uses several results from additive number theory, and in particular a recent result of Bourgain et al. We also consider the related problem of constructing randomness dispersers. For any constant output length m, our dispersers use a constant number of *identical* distributions, each with min-entropy $\Omega(\log n)$, and outputs *every* possible m-bit string with positive probability. The main tool we use is a variant of the "stepping-up lemma" of Erdős and Hajnal used in establishing a lower bound on the Ramsey number for hypergraphs.

Key words. randomness extractors, Ramsey graphs, sum-product theorem

AMS subject classifications. 68Q10, 05B20, 11T23

DOI. 10.1137/S0097539705447141

1. Introduction.

1.1. Background. Randomness is prevalent in computer science and is widely used in algorithms, distributed computing, and cryptography. Perhaps the main motivation and justification for the use of randomness in computation is that randomness exists in nature, and thus it is possible to sample natural phenomena (such as the tossing of coins) in order to make random choices in computation. However, there is a discrepancy between the type of random input that we expect when designing randomized algorithms and protocols, and the type of random data that can be found in nature. While randomized algorithms and protocols expect a stream of independent uniformly distributed random bits, this is too much to hope for from samples of natural phenomena.

Thus, a natural and widely studied problem has been that of constructing *randomness extractors.*[1] Loosely speaking, a *randomness extractor* is a (deterministic polynomial-time computable) function $\mathsf{Ext} : \{0,1\}^n \to \{0,1\}^m$ such that whenever $\mathcal{X}$ is a "good" random variable (where the definition of "good" will be discussed shortly), then $\mathsf{Ext}(\mathcal{X})$ is statistically close to the uniform distribution. The random variable $\mathcal{X}$

*Received by the editors March 16, 2005; accepted for publication (in revised form) April 4, 2006; published electronically December 15, 2006.
http://www.siam.org/journals/sicomp/36-4/44714.html

†Department of Computer Science, Princeton University, Princeton, NJ 08540 (boaz@cs.princeton.edu). The work of this author was done while he was a postdoctoral member at the Institute for Advanced in Princeton, NJ and was supported by NSF grants DMS-0111298 and CCR-0324906.

‡Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093 (russell@cs.ucsd.edu). Much of this author's research was performed at the Institute for Advanced Study in Princeton, NJ and was supported by the State of New Jersey. This research was also partially supported by NSF Award CCR-0098197.

§Institute for Advanced Study, Princeton, NJ 08540 (avi@ias.edu). The research of this author was partially supported by NSF grant CCR-0324906.

[1] We note that while this has been the original motivation for studying randomness extractors, such constructs have found numerous other applications in computer science and are fundamental and interesting objects in their own right.

is intended to model the natural data, while the output of the extractor will be used to make random choices in a probabilistic algorithm or protocol.

Intuitively, the "good" distributions should be the distributions that contain more than m bits of entropy. Otherwise, information-theoretic considerations show that such an extractor cannot exist, even if it is only required to extract randomness from a fixed source that is known to the designer. Actually, past works have converged to the measure of *min-entropy* [10, 49], which is a stronger notion than standard (Shannon) entropy. (The min-entropy of a random variable $\mathcal{X}$, denoted by $H^\infty(\mathcal{X})$, is equal to $\min_{x \in \mathsf{Supp}(\mathcal{X})}(-\log \Pr[\mathcal{X} = x])$.) It can be easily seen that the min-entropy of a random variable $\mathcal{X}$ is always smaller than the (Shannon) entropy of X and that if $\mathsf{Ext}(\mathcal{X})$ is statistically close to being uniform over $\{0,1\}^m$, then the distribution $\mathcal{X}$ must be statistically close to having min-entropy at least m. Thus, possessing high min-entropy is indeed a minimal requirement of the input distribution.

We note that it is known that if a random variable $\mathcal{X}$ has min-entropy k, then it is a convex combination of random variables $\mathcal{X}_1, \ldots, \mathcal{X}_t$, where each $\mathcal{X}_i$ is the uniform distribution over some subset of size 2^k. Such random variables are called *flat*. Because of this fact, when constructing randomness extractors typically we can assume without loss of generality that the input distribution is flat. This is convenient for several reasons, one of which is that the min-entropy of flat distributions is equal to their Shannon entropy.

1.2. Seeded and seedless extractors. An unfortunate and easily seen fact is that there is no *single* function Ext that will produce a (close to) uniform output on *every* input distribution having high min-entropy. Previous works have dealt with this problem in two ways: The first way is to add a short, truly random *seed* as a secondary input to the extractor, and the second way is to use no seed, but make further assumptions on the structure of the weak sources (in addition to the minimal assumption of it containing sufficient min-entropy).

Seeded extractors. As mentioned above, one approach to tackle the above problem has been to allow the extractor to be *probabilistic*. That is, in addition to its input $\mathcal{X}$, one allows the function Ext to have an additional input $\mathcal{Y}$ that is uniformly distributed. The goal is to have the input $\mathcal{Y}$ be (much) *shorter* than the output length m. We call the additional input $\mathcal{Y}$ the *seed* of the extractor, and thus we call such constructions *seeded* extractors.[2] Seeded extractors have been studied extensively in the past two decades (see the survey [42] and references therein) with a series of exciting results, techniques, and applications. The current state of the art is a construction of extractors that are nearly optimal in the sense that they use a seed $\mathcal{Y}$ of length $O(\log n)$ bits, extracting essentially all the min-entropy of the source [27]. This in particular means that by using such extractors, together with enumeration over all possible seed values, it is possible to simulate any probabilistic algorithm with polynomial overhead, using only a high min-entropy source.

Seedless extractors. There are many reasons to try constructing *seedless* extractors. One is that the polynomial overhead implied by the seed enumeration above may sometimes be too expensive. Another is that certain applications in computer science, such as in cryptography and distributed computing, intrinsically prohibit such enumeration. For example, when using a weak source of randomness to choose an encryption key via a seeded extractor, it will certainly be *insecure* to enumerate all

[2] We remark that in most of the literature, the name *randomness extractor* (without any qualifiers) refers to what we call here a *seeded* randomness extractor.

secret keys produced using all seeds and then send the encryptions of a secret message using each of these keys. More generally, it seems that typically we cannot directly use seeded extractors in cryptography (see [28, 15, 5] for different cryptographic models under weak sources of randomness).

There have been many constructions of such seedless extractors that work for specific families of high min-entropy sources. The first to consider this problem was von Neumann, who gave a seedless extractor from a stream of *biased* but independent bits [47] (see also [32]). Other works, such as [6, 41, 10, 12, 11, 29, 25, 44], constructed seedless extractors for more general families of sources.[3]

1.3. Seedless extractors from few independent sources. When seedless extraction from one source is impossible, it is natural to consider doing so from several independent sources of the same quality. After all, assuming we have one such source in nature does not seem much weaker than assuming we have several.

The first to consider this problem were Santha and Vazirani [41], who showed how to use $O(\log n)$ independent "semirandom"[4] sources of length n and min-entropy δn for every constant $\delta > 0$.

Chor and Goldreich [10] were the first to consider general min-entropy sources, and they proved that if $\delta > n/2$, then two sources suffice: indeed the Hadamard–Sylvester matrix $H : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ defined by $H(x,y) = \langle x, y\rangle$ (with the inner product in GF(2)) is such an extractor. (We note that for this construction the $n/2$ entropy bound is tight—there are two sources of entropy exactly $n/2$ on which H is constant.) Vazirani [45] extended this to show that one can use a similar function to output from two independent sources of entropy $(\frac{1}{2} + \epsilon)n$ a linear number of bits that are exponentially close to the uniform distribution.[5]

Improving [10] seems hard. Even its disperser version, namely having a non-constant output for any two sources of entropy rate below 1/2, is the notorious "bipartite Ramsey" problem, which is still open. A slight improvement for this problem was made recently by Pudlak and Rödl [35], who lowered the min-entropy requirement for such a disperser to $n/2 - \sqrt{n}$, but getting a constant $\delta < 1/2$ remains a barrier for two sources. (Very recently, significant progress was made on this problem that builds on the results of the current work; see section 5.)

An alternative 2-source function is the Paley matrix, $P : GF(p) \times GF(p) \to \{\pm 1\}$, defined by $P(x,y) = \chi_2(x+y)$ (where p is an n-bit prime and $\chi_2 : \mathrm{GF}(p) \to \{\pm 1\}$ is the quadratic character). P too is an extractor with exponentially small error for entropy $> n/2$ and is conjectured to have the same property for entropy δn for all $\delta > 0$. While this conjecture is generally believed, proving it seems beyond current techniques.[6] Assuming even more, Zuckerman [48] showed that if this conjecture holds for *all* multiplicative characters (not just χ_2), then a constant (actually $poly(1/\delta)$) number of sources suffices for extraction of linearly many bits with exponential error.

[3] The results of Trevisan and Vadhan [44] differ from the literature, as well as from our work, in that they work in the computational setting; i.e., the restriction on the family of sources is computational, the sources are efficiently sampleable, and the extractors work while assuming an unproven computational assumption.

[4] We will not formally define these semirandom sources (also known as Santha–Vazirani sources) but will only note that they are weaker than high min-entropy sources. Nevertheless one cannot extract seedlessly from only one such source.

[5] Vazirani [45] states his result for the semirandom sources of [41] but it extends to general min-entropy sources.

[6] We note that it is known (using Weil's estimate of character sums) that P is an extractor for two sources, where one of them has entropy $> n/2$ and the other only entropy $> \log n$.

The extractor we use here is *exactly* the same as Zuckerman's (but our analysis uses no unproven assumptions).

1.4. Our results.

1.4.1. Multiple-sample extractors. In this work we will be interested in extracting randomness from a *constant* (larger than 2) number of samples from a high min-entropy source.[7] Loosely speaking, our main result is an efficient construction for extracting (almost) uniformly distributed bits from a constant number of samples from independent distributions over $\{0,1\}^n$, each sample having min-entropy at least δn, for an arbitrarily small constant $\delta > 0$. The statistical distance of our output from the uniform distribution is exponentially low (i.e., $2^{-\Omega(n)}$).[8] More formally, our main theorem is the following.

THEOREM 1.1 (multiple-sample extractor). *For every constant $\delta > 0$ there exists a constant $\ell = (1/\delta)^{O(1)}$ and a polynomial-time computable function $\mathsf{Ext} : \{0,1\}^{n \cdot \ell} \to \{0,1\}^n$ such that for every independent random variable $\mathcal{X}_1, \ldots, \mathcal{X}_\ell$ over $\{0,1\}^n$ satisfying $H^\infty(\mathcal{X}_i) \geq \delta n$ for $i = 1, \ldots, \ell$, it holds that*

$$\mathsf{dist}\Big(\mathsf{Ext}(\mathcal{X}_1, \ldots, \mathcal{X}_\ell), U_n\Big) < 2^{-n},$$

where U_n denotes the uniform distribution over the set $\{0,1\}^n$ and $\mathsf{dist}(\mathcal{X}, \mathcal{Y})$ denotes the statistical distance *of two distributions $\mathcal{X}$ and $\mathcal{Y}$. That is,*

$$\mathsf{dist}(\mathcal{X}, \mathcal{Y}) \stackrel{def}{=} \frac{1}{2} \sum_{i \in \mathsf{Supp}(\mathcal{X}) \cup \mathsf{Supp}(\mathcal{Y})} \Big| \Pr[\mathcal{X} = i] - \Pr[\mathcal{Y} = i] \Big|.$$

As mentioned in the introduction, we denote by $H^\infty(\mathcal{X})$ the min-entropy of the random variable $\mathcal{X}$ (i.e., $H^\infty(\mathcal{X}) = \min_{x \in \mathsf{Supp}(\mathcal{X})}(-\log \Pr[\mathcal{X} = x])$).

Remark 1.2. As is noted in section 3.5, we can actually reduce the statistical distance of our extractor by using more independent samples. In particular, for every $c > 1$, we can ensure that the output of our extractor will be (2^{-cn})-close to the uniform distribution by multiplying the number of samples we use by a factor of $O(c)$. Note that if a distribution $\mathcal{Y}$ over $\{0,1\}^n$ is (2^{-cn})-close to the uniform distribution, then in particular for every $y \in \{0,1\}^n$, $\Pr[\mathcal{Y} = y] \in (2^{-n} - 2^{-cn}, 2^{-n} + 2^{-cn})$ (i.e., $\mathcal{Y}$ is even close to the uniform distribution in the L_∞ norm).

1.4.2. Dispersers. We also present a construction of *dispersers* that works for much lower entropy samples of the *same* source. A *disperser* is a relaxed variant of an extractor, which does not need to output a distribution statistically close to uniform, but rather only a distribution with large support. Thus, when talking about dispersers, it is natural to consider only the support of their input, and so refer to the inputs as sets rather than random variables (where a set of size 2^k corresponds to a random variable of min-entropy k). The formal definition of dispersers is in section 2. Our main result regarding dispersers is the following theorem.

THEOREM 1.3 (multiple-sample same-source disperser). *There exist constants ℓ and d such that for every m, there is a polynomial-time computable function* Disp :

[7] We note that our results can be generalized to the case of extracting from a larger (i.e., superconstant) number of samples having sublinear entropy. However, we do not consider such extensions in this paper.

[8] Note that we get statistical distance that is much better than what is possible to obtain with seeded extractors, which cannot do better than a polynomially small distance when using a logarithmic-length seed. This low statistical distance is important for cryptographic applications.

$\{0,1\}^{n\cdot\ell} \to \{0,1\}^m$ *satisfying that for every subset* $\mathcal{X} \subseteq \{0,1\}^n$ *with* $|\mathcal{X}| \geq n^{d2^m}$,

$$\mathsf{Disp}(\mathcal{X},\ldots,\mathcal{X}) = \{0,1\}^m.$$

That is, if $|\mathcal{X}| \geq n^{d2^m}$, *then for every* $y \in \{0,1\}^m$, *there exist* $x_1,\ldots,x_\ell \in \mathcal{X}$ *such that* $\mathsf{Disp}(x_1,\ldots,x_\ell) = y$.

This is a better disperser than the multiple-source extractor of Theorem 1.1 in the sense that it works for very low min-entropy (indeed, note that if we let m be a constant, then sets of size $n^{O(1)}$ correspond to distributions of min-entropy of $O(\log n)$). However, it still has two drawbacks: one is that its output is much smaller than the input entropy (although it can still be much larger than the number of samples); another is that it requires all its input samples to come from the *same* distribution $\mathcal{X}$ (rather than from different distributions, as in our extractor). We consider the second drawback to be the more serious one. The construction of this disperser closely follows the "stepping-up" technique of Erdős and Hajnal (see Graham, Rothschild, Spencer [22, sect. 4.7]) it that it gives an (explicit) lower bound on the Ramsey number of hypergraphs. We remark that we work for worse entropy than this lower-bound[9] because we want to output a superconstant number of bits (that does not depend on the number of samples).

1.5. Techniques.

An Erdős–Szemerédi theorem for finite fields. Our main tools for Theorem 1.1 are several results from additive number theory, and in particular, a relatively recent result by Bourgain, Katz, and Tao [9]. They proved an analogue of the Erdős–Szemerédi [17] theorem for finite prime fields. Let $\mathcal{A}$ be a subset of some field $\mathbb{F}$. We define the set $\mathcal{A}+\mathcal{A}$ to equal $\{a+b \mid a,b \in \mathcal{A}\}$ and the set $\mathcal{A}\cdot\mathcal{A}$ to equal $\{a\cdot b \mid a,b \in A\}$. Note that $|\mathcal{A}| \leq |\mathcal{A}+\mathcal{A}| \leq |\mathcal{A}|^2$ (and similarly $|\mathcal{A}| \leq |\mathcal{A}\cdot\mathcal{A}| \leq |\mathcal{A}|^2$). An example for a set $\mathcal{A}$, where $\mathcal{A}+\mathcal{A}$ is small (of size about $2|\mathcal{A}|$), is an *arithmetic progression.* An example for a set $\mathcal{A}$, where $\mathcal{A}\cdot\mathcal{A}$ is small, is a *geometric progression.* The Erdős–Szemerédi theorem is that for every set $\mathcal{A} \subseteq \mathbb{N}$, either $\mathcal{A}+\mathcal{A}$ or $\mathcal{A}\cdot\mathcal{A}$ is of size at least $|\mathcal{A}|^{1+\epsilon_0}$ for some universal constant ϵ_0. In some sense, one can view this theorem as saying that a set of integers can't be simultaneously close to both an arithmetic progression and a geometric progression.

A natural question, which has many diverse applications in geometry and analysis, is whether this theorem also holds in *finite* fields. A first observation is that this theorem is *false* in a field $\mathbb{F}$ that contains a nontrivial subfield $\mathbb{F}'$. This is because if we let $\mathcal{A} = \mathbb{F}'$, then $\mathcal{A}+\mathcal{A} = \mathcal{A}\cdot\mathcal{A} = \mathcal{A}$. However, [9] showed that a variant of this theorem does hold in a finite field with no nontrivial subfields.[10] In particular it holds in the fields $\mathrm{GF}(p)$ and $\mathrm{GF}(2^p)$ for every prime p. That is, [9] proved the following.

THEOREM 1.4 (see [9]). *Let* $\delta > 0$ *be some constant and let* $\mathbb{F}$ *be a field with no subfield of size between* $|\mathbb{F}|^{\delta/2}$ *and* $|\mathbb{F}|-1$. *Let* $\mathcal{A} \subseteq \mathbb{F}$ *be a set such that* $|\mathbb{F}|^\delta < |\mathcal{A}| < |\mathbb{F}|^{1-\delta}$. *Then there exists some constant* ϵ *(depending on* δ*) such that*

$$\max\{|\mathcal{A}+\mathcal{A}|, |\mathcal{A}\cdot\mathcal{A}|\} > |\mathcal{A}|^{1+\epsilon}.$$

In [9], the dependence of the constant ϵ on δ was not specified (and examining the proof shows that it is probably of the form $\epsilon = 2^{-\Omega(1/\delta)}$). In the appendix we give

[9] Using c samples, this lower bound yields a 1-bit-output disperser for $\log^{(\Theta(c))} n$ entropy, where $\log^{(i)} n$ denotes the result of iterating $\log$ i times on n.

[10] By a *trivial* subfield in this context we mean a subfield that is either equal to the entire field or very small (e.g., of constant size).

a proof of Theorem 1.4 (using some of the lemmas of [9]) with $\epsilon = \Theta(\delta)$.[11] Konyagin [26] gave a stronger result for *prime* fields and showed that, as long as $|\mathcal{A}| < |\mathbb{F}|^{0.99}$, the value ϵ can be made a constant independent of the size of $\mathcal{A}$. That is, he proved the following theorem.

THEOREM 1.5 (see [26]). *There exists some constant ϵ_0 such that for every $\mathbb{F}$ that is a field of prime order, and every $\mathcal{A} \subseteq \mathbb{F}$ with $|\mathcal{A}| < |\mathbb{F}|^{0.99}$,*

$$\max\{|\mathcal{A}+\mathcal{A}|, |\mathcal{A}\cdot\mathcal{A}|\} > |\mathcal{A}|^{1+\epsilon_0}.$$

1.6. How is this related to extractors?

Using Theorem 1.5 *to obtain dispersers.* Theorem 1.5 actually already implies some kind of multiple-sample disperser that was not previously known. This is because it implies the following corollary.

COROLLARY 1.6. *There exists some constant ϵ_0 such that for every $\mathbb{F}$ a field of prime order, and every $\mathcal{A} \subseteq \mathbb{F}$ with $|\mathcal{A}| < |\mathbb{F}|^{0.99}$,*

$$|\mathcal{A}\cdot\mathcal{A}+\mathcal{A}| > |\mathcal{A}|^{1+\epsilon_0}.$$

Indeed, by Theorem 1.5, for every set $\mathcal{A}$ either $|\mathcal{A}\cdot\mathcal{A}|$ or $|\mathcal{A}+\mathcal{A}|$ is at least $|\mathcal{A}|^{1+\epsilon'}$ for some absolute constant ϵ'. If the former holds, then since $|\mathcal{A}\cdot\mathcal{A}+\mathcal{A}| \geq |\mathcal{A}\cdot\mathcal{A}|$ we're done. In the latter case we can use previously known number-theoretic results (see also Lemma 3.7) that imply that if $|\mathcal{A}+\mathcal{A}| \geq |\mathcal{A}|^{1+\epsilon'}$, then $|\mathcal{A}+\mathcal{B}| \geq |\mathcal{A}|^{1+\epsilon'/2}$ for every set $\mathcal{B}$ with $|\mathcal{B}| = |\mathcal{A}|$. This means that if we let s be any member of $\mathcal{A}$ and consider $\mathcal{B} = s^{-1}\mathcal{A}$, then we get that $|\mathcal{A}+\mathcal{B}| \geq |\mathcal{A}|^{1+\epsilon'/2}$ (since multiplying by a fixed element is a permutation that does not change the size of a set). For the same reason, $|\mathcal{A}+\mathcal{B}| = |\mathcal{A}+s^{-1}\mathcal{A}| = |s\mathcal{A}+\mathcal{A}|$, but the set $s\mathcal{A}+\mathcal{A}$ is a subset of $\mathcal{A}\cdot\mathcal{A}+\mathcal{A}$.

In other words, Corollary 1.6 states that there is an efficiently computable $f(\cdot)$ such that $|f(\mathcal{A},\mathcal{A},\mathcal{A})| \geq |\mathcal{A}|^{1+\epsilon_0}$ (i.e., $f(a,b,c) = a\cdot b+c$). Yet this implies that for every set $\mathcal{A}$ with $|\mathcal{A}| \geq |\mathbb{F}|^{\delta}$, if we compose f with itself $O(\log(1/\delta))$ times, we get a function $g : \mathbb{F}^{\ell} \to \mathbb{F}$ (for $\ell = 2^{O(\log(1/\delta))} = (1/\delta)^{O(1)}$) such that $g(\mathcal{A},\ldots,\mathcal{A}) = \mathbb{F}$ (where $g(\mathcal{A},\ldots,\mathcal{A})$ denotes the image of g on $\mathcal{A}^{\ell}$).[12] If we identify the field $\mathbb{F}$ with the strings $\{0,1\}^n$, then we see that this function g (which is obviously polynomial-time computable) is already some kind of a disperser.

Obtaining extractors. To obtain an *extractor* rather than a disperser, we would like to obtain a *statistical* analogue of Theorem 1.5. Consider the following notation. If $\mathcal{X}$ is a random variable, then we define $\mathcal{X}+\mathcal{X}$ to be the distribution obtained by choosing a, b independently at random from $\mathcal{X}$ and outputting $a+b$, and we define $\mathcal{X}\cdot\mathcal{X}$ in a similar way. Then, a statistical analogue of Theorem 1.5 would be that there exists some $\epsilon > 0$ such that for every random variable $\mathcal{X}$ with min-entropy at most $0.99\log|\mathbb{F}|$, either the distribution $\mathcal{X}+\mathcal{X}$ or the distribution $\mathcal{X}\cdot\mathcal{X}$ has min-entropy at least $(1+\epsilon)H^{\infty}(\mathcal{X})$. Unfortunately, this is false: For every prime field $\mathbb{F}$, there is a random variable $\mathcal{X}$ that is uniform over some set of size 2^k (for $k \ll 0.99\log|\mathbb{F}|$) and such that for both $\mathcal{X}+\mathcal{X}$ and $\mathcal{X}\cdot\mathcal{X}$, a constant fraction of the probability mass is concentrated in a set of size $O(2^k)$. Indeed, an example for such a random variable $\mathcal{X}$

[11] The proof of in our appendix also has the advantage of proving the theorem for all fields $\mathbb{F}$ simultaneously. The authors of [9] prove Theorem 1.4 for prime fields and then only mention how it can be generalized to other fields.

[12] This is because each time we apply f we grow in the set size from m to $m^{1+\epsilon_0}$. We note that one needs to analyze separately the case that $|\mathcal{A}| > |\mathbb{F}|^{0.99}$, but this can be done. This function g is described in more detail in the beginning of section 3.

is a random variable that is with probability half an arithmetic progression and with probability half a geometric progression.

Even though the statistical analogue for Theorem 1.5 does not hold, we show that a statistical analogue for Corollary 1.6 *does* hold. That is, we prove (in Lemma 3.1) that there exists some constant $\epsilon_0 > 0$ such that for every random variable $\mathcal{X}$ over a prime field $\mathbb{F}$, the distribution $\mathcal{X} \cdot \mathcal{X} + \mathcal{X}$ has (up to some negligible statistical distance) min-entropy at least $(1+\epsilon_0)H^\infty(\mathcal{X})$. (In fact, we need to (and do) prove a more general statement regarding distributions of the form $\mathcal{X} \cdot \mathcal{Y} + \mathcal{Z}$.) The proof of this lemma is the main technical step in our extractor. The proof uses Theorem 1.5, along with some other additive number-theoretic results of Ruzsa [40] and Gowers [21].

We use this lemma to show that the function g sketched above is actually not just a disperser but an *extractor*. That is, we show that for every random variable $\mathcal{X}$ of min-entropy at least $\delta \log |\mathbb{F}|$, the random variable $g(\mathcal{X}, \ldots, \mathcal{X})$ (where g is applied to ℓ independent copies of $\mathcal{X}$) not only has large support but is also in fact very close to the uniform distribution. (In fact, we need to (and do) prove a stronger version of this statement, namely, that $g(\mathcal{X}_1, \ldots, \mathcal{X}_\ell)$ is close to uniform for every independent random variable, $\mathcal{X}_1, \ldots, \mathcal{X}_\ell$ with $H^\infty(\mathcal{X}_i) \geq \delta \log |\mathbb{F}|$ for $i = 1, \ldots, \ell$.)

2. Preliminaries. In this section we establish our definitions for multiple-sample extractors and dispersers. Unfortunately, such definitions tend to have a large number of parameters.

DEFINITION 2.1 (multiple-sample extractor). *A function* $\mathsf{Ext} : \{0,1\}^{n \cdot \ell} \to \{0,1\}^m$ *is called an* ℓ-sample (k, m, ϵ)-extractor *if for every independent random variable* $\mathcal{X}_1, \ldots, \mathcal{X}_\ell$ *satisfying* $H^\infty(\mathcal{X}_i) \geq k$ *for* $i = 1, \ldots, \ell$ *it holds that*

$$\mathsf{dist}\Big(\mathsf{Ext}(\mathcal{X}_1, \ldots, \mathcal{X}_\ell), U_m\Big) < \epsilon.$$

Parameters and qualifiers:

- The parameter ℓ is called the *number of samples* of the extractor. In all our constructions we will use ℓ, which is a constant independent of the input length.
- The parameter m is called the *output length* of the extractor. Usually, we'll use $m = n$.
- The parameter k is called the min-entropy requirement by the extractor.
- The parameter ϵ is called the statistical distance of the extractor.
- One can also make a weaker definition in which the extractor is required to output a close-to-uniform value only if the variables $\mathcal{X}_1, \ldots, \mathcal{X}_\ell$ are identically distributed. We call such an extractor a *same-source* extractor. Thus, we will sometimes say that a standard (as per Definition 2.1) multiple-sample extractor is a *different-source* extractor.

We now define the weaker notion of a disperser.

DEFINITION 2.2 (multiple-sample disperser). *A function* $\mathsf{Disp} : \{0,1\}^{n \cdot \ell} \to \{0,1\}^m$ *is called an* ℓ-sample (k, m)-disperser *if for all sets* $\mathcal{X}_1, \ldots, \mathcal{X}_\ell$ *satisfying* $|\mathcal{X}_i| \geq 2^k$ *for* $i = 1, \ldots, \ell$, *it holds that*

$$\mathsf{Disp}(\mathcal{X}_1, \ldots, \mathcal{X}_\ell) = \{0,1\}^m.$$

Notes:

- We will use the same names for the parameters and qualifiers of dispersers as we used for extractors (e.g., number of samples, output length, same-source, different-source).

- In previous literature, dispersers are usually defined with a parameter ϵ analogous to the statistical distance of extractors, requiring $|\mathsf{Disp}(\mathcal{X}_1, \ldots, \mathcal{X}_\ell)| \geq (1-\epsilon)2^m$ instead of the requirement we make. Thus, one can think of our definition as setting $\epsilon = 0$. However, in our particular setting of independent samples, this is essentially without loss of generality, as we can convert a disperser satisfying the weaker definition with any constant $\epsilon < 1/2$ into a disperser satisfying the stronger definition with only a constant factor increase in the number of samples.[13]

Basic facts and observations. The following facts regarding multiple-sample extractors and dispersers are either known or easy to verify:

- There does not exist a 1-sample disperser (and hence an extractor) even with only one bit of output and even if the source is assumed to have $n-1$ bits of min-entropy.
- There is a simple construction for a 2-sample *same-source* 1-bit-output extractor whenever the source min-entropy is larger than $O(\log \frac{1}{\epsilon})$ (where ϵ is the statistical distance). This is the extractor that on input x, y outputs 1 if $x > y$.
- In contrast, by Ramsey-type arguments, a 2-sample same-source disperser (and hence an extractor) with more than one bit of output requires the input min-entropy to be at least $\log n$. The same holds for a different-source 1-bit-output disperser.

  The best known previous explicit constructions for both cases require the min-entropy to be more than $n/2$ [10, 46]. Indeed, the function $H : \{0,1\}^{2n} \to \{0,1\}$ defined as $H(x,y) = \sum x_i y_i \pmod 2$ (i.e., the adjacency function of the Hadamard graph) is known to be a 1-bit-output extractor for sources with more than $n/2$ entropy [10]. This is essentially the best known previous construction in terms of the minimum entropy requirement. There has been some improvement in obtaining variants of this extractor that have a larger output size [46, 16, 14], although these still require at least $n/2$ entropy. (Also, as mentioned above, it is known that the adjacency function of the Paley graph is a 1-bit-output extractor that works as long as one of its inputs has more than $n/2$ entropy, while the other input needs only to have more than $\log n$ entropy [23, 2]. However, in this paper we restrict ourselves to the *symmetric* case, where all sources have the same entropy requirement.)
- Using enumeration over all possible seeds, one can use a *seeded* extractor to obtain a polynomial-samples same-source extractor with the same requirement over the min-entropy. It is also possible to generalize this to work for *different sources* [38].

Explicit constructions. In the definition of extractors and dispersers we did not require these functions to be efficiently computable. However, naturally we will be interested in obtaining extractors and dispersers that are computable in polynomial time. We call such constructions *explicit*.

3. Constructing a multiple-sample extractor. In this section we prove Theorem 1.1. That is, we construct an extractor Ext (where $\mathsf{Ext} : \{0,1\}^{\ell \cdot n} \to \{0,1\}^n$)

[13] This can be done, for example, by identifying the set $\{0,1\}^n$ with elements of a prime field $\mathbb{F}$ (as we do in what follows) and using the Cauchy–Davenport theorem. This theorem says that if $\mathcal{A}$ and $\mathcal{B}$ are subsets of a prime field $\mathbb{F}$, then the set $\mathcal{A} + \mathcal{B} = \{a + b | a \in \mathcal{A}, b \in \mathcal{B}\}$ is of size at least $\min\{|\mathbb{F}|, |\mathcal{A}| + |\mathcal{B}| - 1\}$. Hence, if $\mathcal{A}_1, \ldots, \mathcal{A}_k$ are sets of size $> \epsilon|\mathbb{F}|$ with $k > 2/\epsilon$, then $\mathcal{A}_1 + \cdots + \mathcal{A}_k = \mathbb{F}$.

such that $\mathsf{Ext}(\mathcal{X}_1,\ldots,\mathcal{X}_\ell)$ is statistically close to the uniform distribution for every independent random variables $\mathcal{X}_1,\ldots,\mathcal{X}_\ell$ over $\{0,1\}^n$ with high enough min-entropy (i.e., at least δn, where $\ell = \mathrm{poly}(1/\delta)$). Our extractor will be very simple, involving only a recursive composition of the operation $(a,b,c) \mapsto a \cdot b + c$. As noted in the introduction, this is the same construction used by Zuckerman [48].

Formally, we fix a suitable field $\mathbb{F}$ (see below) and define the functions Ext^i for all $i \in \mathbb{N}$ recursively as follows:

- For every i, the function Ext^i will be a mapping from $\mathbb{F}^{3^i}$ to $\mathbb{F}$.
- $\mathsf{Ext}^0 : \mathbb{F}^{3^0} = \mathbb{F} \to \mathbb{F}$ is the identity function $\mathsf{Ext}^0(x) = x$.
- Assume Ext^i is already defined. We define $\mathsf{Ext}^{i+1} : \mathbb{F}^{3^{i+1}} = (\mathbb{F}^{3^i})^3 \to \mathbb{F}$ as follows: For every $x_1, x_2, x_3 \in \mathbb{F}^{3^i}$,

$$\mathsf{Ext}^{i+1}(x_1,x_2,x_3) \stackrel{def}{=} \mathsf{Ext}^i(x_1)\cdot\mathsf{Ext}^i(x_2) + \mathsf{Ext}^i(x_3).$$

Our extractor will be equal to Ext^i for a suitably chosen constant i. When extracting from a string in $\{0,1\}^n$, we will choose the field $\mathbb{F}$ to be of the form $\mathrm{GF}(N)$ for some prime $N \in [2^n, 2^{n+1})$ (and hence we can identify any string in $\{0,1\}^n$ with an element in the field $\mathbb{F}$). We postpone the issue of finding such a prime to section 3.4.

Theorem 1.1 will follow from the following two lemmas.

LEMMA 3.1. *There exists some constant $\epsilon > 0$ such that for every distribution $\mathcal{A},\mathcal{B},\mathcal{C}$ over a prime field $\mathbb{F}$, each with* min*-entropy at least m, the distribution $\mathcal{A}\cdot\mathcal{B}+\mathcal{C}$ is $(2^{-\epsilon m})$-close to having* min*-entropy at least $\min\{(1+\epsilon)m, 0.9\log|\mathbb{F}|\}$.*

LEMMA 3.2. *Let $\mathcal{A}_1,\ldots,\mathcal{A}_9$ be 9 independent distributions over a prime field $\mathbb{F}$ each with* min*-entropy at least $0.9\log|\mathbb{F}|$. Then, $\mathsf{Ext}^2(\mathcal{A}_1,\ldots,\mathcal{A}_9)$ is of distance at most $|\mathbb{F}|^{-0.01}$ from the uniform distribution over $\mathbb{F}$.*

Lemmas 3.1 and 3.2 imply Theorem 1.1. Indeed, Lemma 3.1 implies that for every constant $\delta > 0$, if we let $i = \log_{(1+\epsilon)}(1/\delta)$, then the output of Ext^i is close to having min-entropy $0.9\log|\mathbb{F}|$. Note that the number of samples required by Ext^i is 3^i, which is polynomial in $1/\delta$. We use Lemma 3.2 to get from min-entropy $0.9\log|\mathbb{F}|$ to a close-to-uniform output. Using the union bound, we see that the statistical distance of the extractors output from the uniform distribution on $\mathbb{F}$ will be at most $N^{-\Omega(1)}$. We defer the proof of Lemma 3.2 to section 3.6 and now turn to proving Lemma 3.1.[14]

3.1. Basic facts and notation. Before proving Lemma 3.1, we introduce some notation and some tools that will be used in the course of the proof. We identify a set $\mathcal{A}$ with the uniform distribution on elements of $\mathcal{A}$. If $\mathcal{D}$ is a distribution, then we denote by $\mathsf{cp}(\mathcal{D})$ the collision probability of $\mathcal{D}$, that is, $\mathsf{cp}(\mathcal{D}) = \Pr_{x,y\leftarrow_R\mathcal{D}}[x=y]$. Note that for a set $\mathcal{A}$, $\mathsf{cp}(\mathcal{A}) = \frac{1}{|\mathcal{A}|}$. If $\mathcal{A}$ and $\mathcal{B}$ are distributions over subsets of the field $\mathbb{F}$, then we denote by $\mathcal{A}+\mathcal{B}$ the *distribution* obtained by picking a at random from $\mathcal{A}$, picking b at random from $\mathcal{B}$, and outputting $a+b$. Note that this distribution may be far from the uniform distribution with the same support. We define the distribution $\mathcal{A}\cdot\mathcal{B}$ in a similar way. For $k \in \mathbb{Z}$ and $\mathcal{A}$ a set or a distribution, we define $k\mathcal{A}$ and $\mathcal{A}^k$ in the natural way.[15]

[14] Note that we could prove Theorem 1.1 without using Lemma 3.2 by applying the previously known 2-sample extractors, which work for entropy larger than $\frac{\log|\mathbb{F}|}{2}$. In addition, Vadhan observed that by using the fact that the function family $\{h_{b,c}\}$ (where $h_{b,c}(a) = a\cdot b+c$) is pairwise independent and the leftover hash lemma of [24], one can prove that under the conditions of Lemma 3.2 the first $0.8\log|F|$ bits of $\mathcal{A}_1\cdot\mathcal{A}_2+\mathcal{A}_3$ are within statistical distance $|F|^{-0.01}$ to the uniform distribution. This is also sufficient to prove Theorem 1.1.

[15] E.g., for $k>0$, $k\mathcal{A} = \underbrace{\mathcal{A}+\cdots+\mathcal{A}}_{k \text{ times}}$ and for $k<0$, $k\mathcal{A} = \underbrace{-\mathcal{A}-\cdots-\mathcal{A}}_{|k| \text{ times}}$.

We say that a distribution $\mathcal{X}$ is a convex combination of distributions $\mathcal{X}_1, \ldots, \mathcal{X}_m$ if there exist numbers $p_1, \ldots, p_m \in [0,1]$ such that $\sum_i p_i = 1$ and the random variable $\mathcal{X}$ (when looked at as a vector of probabilities) is equal to $\sum_i p_i \mathcal{X}_i$.

The following lemmas would be useful.

LEMMA 3.3. *Let $\mathcal{X}$ and $\mathcal{Y}$ be distributions; then* $\Pr[\mathcal{X} = \mathcal{Y}] \leq \sqrt{\mathsf{cp}(\mathcal{X})\mathsf{cp}(\mathcal{Y})} \leq \max\{\mathsf{cp}(\mathcal{X}), \mathsf{cp}(\mathcal{Y})\}$.

Proof. For every i in the union of the supports of $\mathcal{X}$ and $\mathcal{Y}$, let x_i denote the probability that $\mathcal{X} = i$ and let y_i denote the probability that $\mathcal{Y} = i$. Then

$$\Pr[\mathcal{X} = \mathcal{Y}] = \sum_i x_i y_i \leq \sqrt{\sum_i x_i^2 \sum_j y_j^2}$$

by the Cauchy-Schwarz inequality; however this is the geometric mean of $\mathsf{cp}(\mathcal{X})$ and $\mathsf{cp}(\mathcal{Y})$ (which is less than the maximum of these quantities). ∎

COROLLARY 3.4. *Let $\mathcal{A}, \mathcal{C}$ be distributions over $\mathbb{F}$. Then* $\mathsf{cp}(\mathcal{A} + \mathcal{C}) \leq \sqrt{\mathsf{cp}(\mathcal{A} - \mathcal{A})\mathsf{cp}(\mathcal{C} - \mathcal{C})}$.

Proof. Let $\mathcal{A}'$ and $\mathcal{C}'$ be two new, independent random variables distributed identically to $\mathcal{A}$ and $\mathcal{C}$, respectively. Then,

$$\mathsf{cp}(\mathcal{A} + \mathcal{C}) = Pr[\mathcal{A} + \mathcal{C} = \mathcal{A}' + \mathcal{C}'] = \Pr[\mathcal{A} - \mathcal{A}' = \mathcal{C} - \mathcal{C}'],$$

which is smaller than $\sqrt{\mathsf{cp}(\mathcal{A} - \mathcal{A})\mathsf{cp}(\mathcal{C} - \mathcal{C})}$ by Lemma 3.3. ∎

LEMMA 3.5. *Suppose that $\mathcal{X}$ is a convex combination of distributions $\mathcal{X}_1, \ldots, \mathcal{X}_m$. Then* $\mathsf{cp}(\mathcal{X}) \leq \max\{\mathsf{cp}(\mathcal{X}_1), \ldots, \mathsf{cp}(\mathcal{X}_m)\}$.

Proof. Using induction on m, this reduces to the case that $m = 2$ (since we can treat the combination of $\mathcal{X}_1, \ldots, \mathcal{X}_{m-1}$ as one distribution whose collision probability is bounded using induction). However, in this case $\mathcal{X} = \alpha\mathcal{X}_1 + (1-\alpha)\mathcal{X}_2$, and then

$$\mathsf{cp}(\mathcal{X}) = \alpha^2 \mathsf{cp}(\mathcal{X}_1) + 2\alpha(1-\alpha)\Pr[\mathcal{X}_1 = \mathcal{X}_2] + (1-\alpha)^2 \mathsf{cp}(\mathcal{X}_2).$$

However, $\Pr[\mathcal{X}_1 = \mathcal{X}_2] \leq \max\{\mathsf{cp}(\mathcal{X}_1), \mathsf{cp}(\mathcal{X}_2)\}$ by Lemma 3.3 and so we are done. ∎

LEMMA 3.6. *Let $\mathcal{X}$ be a distribution such that* $\mathsf{cp}(\mathcal{X}) \leq \frac{1}{KL}$. *Then $\mathcal{X}$ is of statistical distance* $\frac{1}{\sqrt{L}}$ *from having* min*-entropy at least* $\log K$.

Proof. We can split $\mathcal{X}$ into a convex combination $\mathcal{X} = \alpha\mathcal{X}' + (1-\alpha)\mathcal{X}''$, where $\mathcal{X}'$ contains the "heavy" elements of $\mathcal{X}$ that are obtained with probability at least $\frac{1}{K}$, and $\mathcal{X}''$ contains the rest of $\mathcal{X}$. We see that $\mathsf{cp}(\mathcal{X}) \geq \alpha^2 \mathsf{cp}(\mathcal{X}')$, and so $\frac{1}{KL} \geq \frac{\alpha^2}{K}$ and thus $\alpha \leq \frac{1}{\sqrt{L}}$. However, $H^\infty(\mathcal{X}'') \geq \log K$ and so we are done. ∎

Note that if $H^\infty(\mathcal{X}) \geq k$, then clearly $\mathsf{cp}(\mathcal{X}) \leq 2^{-k}$. Together with Lemma 3.6 this implies that, up to an arbitrarily close-to-1 multiplicative factor and an exponentially small (in the min-entropy) statistical distance, $H^\infty(X)$ is approximately equal to $\log(1/\mathsf{cp}(\mathcal{X}))$. (The quantity $\log(1/\mathsf{cp}(\mathcal{X}))$ is sometimes called the 2-*entropy* or the *Renyi entropy* of $\mathcal{X}$.)

3.2. Additive number-theoretic results. The following two lemmas hold for any Abelian group, and so their "+" operators may be replaced with "·" operators. Note that we do not state these lemmas with the most optimal choice of constants.

LEMMA 3.7 (see [40]; see also [31, 39]). *Let $\mathcal{A}, \mathcal{B}, \mathcal{C}$ be subsets of some Abelian group G. Then* $|\mathcal{A} + \mathcal{C}||\mathcal{B}| \leq |\mathcal{A} + \mathcal{B}||\mathcal{B} + \mathcal{C}|$.

In particular, $|\mathcal{A}| = |\mathcal{B}| = M$ and $\rho > 0$ is such that $|\mathcal{A} + \mathcal{B}| \leq M^{1+\rho}$. Then $|\mathcal{A} + \mathcal{A}| \leq \frac{|\mathcal{A}+\mathcal{B}|^2}{M} \leq M^{1+2\rho}$.

In other words, if $\mathcal{A}+\mathcal{B}$ is "small" for some $\mathcal{B}$, then $\mathcal{A}+\mathcal{A}$ is small. Note that Lemma 3.7 implies that we can replace $\mathcal{A}+\mathcal{A}$ and $\mathcal{A}\cdot\mathcal{A}$ in Theorem 1.4 with $\mathcal{A}+\mathcal{B}$ and $\mathcal{A}\cdot\mathcal{B}$ for every $\mathcal{B}$ satisfying $|\mathcal{B}|=|\mathcal{A}|$. Note that Lemma 3.7 also implies that if $|\mathcal{A}|=M$, $|\mathcal{B}|\geq M^{1-\epsilon}$, and $|\mathcal{A}+\mathcal{B}|\leq M^{1+\rho}$, then $|\mathcal{A}+\mathcal{A}|\leq M^{1+2\rho+\epsilon}$.

We will use the following lemma of Gowers (which is a quantitative improvement of a result by Balog and Szemerédi). We state it here using different sets $\mathcal{A}$,$\mathcal{B}$ (similarly to the way it is quoted in [9], although in [21] it is stated with $\mathcal{A}=\mathcal{B}$). See also Lemma A.5 for a generalization of this lemma obtained in [43].[16]

LEMMA 3.8 (see Proposition 12 in [21]). *Let $\mathcal{A},\mathcal{B}$ be subsets of some group G with $|\mathcal{A}|=|\mathcal{B}|=M$ and let $\rho>0$ be such that $\mathsf{cp}(\mathcal{A}+\mathcal{B})\geq\frac{1}{M^{1+\rho}}$. Then there exists $\mathcal{A}'\subseteq\mathcal{A}$ and $\mathcal{B}'\subseteq\mathcal{B}$ such that $|\mathcal{A}'|,|\mathcal{B}'|\geq M^{1-10\rho}$ and $|\mathcal{A}'+\mathcal{B}'|\leq M^{1+10\rho}$.*

We remark that it can be shown that if $\mathcal{A}=\mathcal{B}$ in the conditions of this lemma, then $\mathcal{A}'=\mathcal{B}'$ in its conclusion. We also note that we can apply the lemma even if $\mathcal{A}$ and $\mathcal{B}$ are of different sizes, as long as they are close enough. Indeed, if $|\mathcal{A}|=M$ and $|\mathcal{B}|=M^{1-\rho/10}$, then we can partition $\mathcal{A}$ to subsets $\mathcal{A}_1,\ldots,\mathcal{A}_k$ of size $|\mathcal{B}|$, and since $\mathcal{A}+\mathcal{B}$ is a convex combination of $\mathcal{A}_i+\mathcal{B}$, one of these subsets has collision probability as least as large as $\mathsf{cp}(\mathcal{A}+\mathcal{B})$, and we can apply the lemma to it.

3.3. Proof of Lemma 3.1.

Fixing ϵ. We fix ϵ small enough such that for every $M<|\mathbb{F}|^{0.99}$, Theorem 1.5 ensures us that if $\mathcal{X}$ is a set of size at least $M^{1-10^4\epsilon}$, then $\max\{|\mathcal{X}\cdot\mathcal{X}|,|\mathcal{X}+\mathcal{X}|\}$ is at least $M^{1+10^4\epsilon}$ (e.g., we can take $\epsilon=\epsilon_0/10^9$). Note that this is the only place in the proof in which we use the fact that $\mathbb{F}$ is a prime field. In particular, using Theorem 1.4 instead of Theorem 1.5, our proof yields also a variant of Lemma 3.1 for nonprime fields (see Lemma 3.14).

Statistical analogue of Theorem 1.5. Roughly speaking, Theorem 1.5 says that if $|\mathcal{A}\cdot\mathcal{A}|$ is "small," then $|\mathcal{A}+\mathcal{A}|$ is "large." By combining Theorem 1.4 with Lemma 3.7, it is possible get a "different sets" analogue of this statement and show that for every set $\mathcal{A}$ of size M, if $|\mathcal{A}\cdot\mathcal{B}|$ is small (i.e. $\leq M^{1+\epsilon}$) for some set $\mathcal{B}$ of size M, then $|\mathcal{A}+\mathcal{C}|$ is large ($\geq M^{1+\epsilon}$) for *every* set $\mathcal{C}$ of size M.

At this point we would have liked to obtain a collision probability analogue of this statement. That is, we would like to prove that if $\mathcal{A},\mathcal{B},\mathcal{C}$ are uniform distributions over sets of size M, and $\mathsf{cp}(\mathcal{A}\cdot\mathcal{B})\geq\frac{1}{M^{1+\epsilon}}$, then $\mathsf{cp}(\mathcal{A}+\mathcal{C})\leq\frac{1}{M^{1+\epsilon}}$. Unfortunately, this is false, as can be witnessed by considering the following counterexample. Let $M=N^{0.1}$, and let $\mathcal{A}=\mathcal{B}=\mathcal{C}$ be a distribution that is an arithmetic progression of size M with probability $\frac{1}{2}$ and a geometric progression of size M with probability $\frac{1}{2}$. For such a distribution, both $\mathsf{cp}(\mathcal{A}+\mathcal{A})$ and $\mathsf{cp}(\mathcal{A}\cdot\mathcal{A})$ are at least $\Omega(1/M)$.[17]

However, we are able to prove a slightly weaker statement. That is, we show that if $\mathcal{A}\cdot\mathcal{B}$ is small in *set size*, then $\mathcal{A}+\mathcal{C}$ shrinks significantly in *collision probability*. This is stated in the following lemma.

[16] We note that in some sources the lemma is stated with the condition that the distribution $\mathcal{A}+\mathcal{B}$ is of statistical distance at least $M^{-C\rho}$ from having min-entropy $(1+\rho)\log M$ (a condition that up to constant factor in the distance is equivalent to the condition that there is a subset $\mathcal{C}$ such that $|\mathcal{C}|\leq M^{1+\rho/(2C)}$ but $\Pr[\mathcal{A}+\mathcal{B}\in\mathcal{C}]\geq M^{-C\rho}$). However, this is equivalent to our statement of Lemma 3.8 by the observations above on the relation between min-entropy and collision probability. In particular, if there is such a set $\mathcal{C}$, then the collision probability of $\mathcal{A}+\mathcal{B}$ is at least $\Pr[\mathcal{A}+\mathcal{B}\in\mathcal{C}]|\mathcal{C}|^{-1}$.

[17] Let $\mathcal{A}_{\mathsf{arith}}$ denote the arithmetic progression. A random element from $\mathcal{A}+\mathcal{A}$ is in the set $\mathcal{A}_{\mathsf{arith}}+\mathcal{A}_{\mathsf{arith}}$ with probability at least $\frac{1}{4}$. However, because the set $\mathcal{A}_{\mathsf{arith}}+\mathcal{A}_{\mathsf{arith}}$ is of size at most $O(M)$, we get that $\mathsf{cp}(\mathcal{A}+\mathcal{A})\geq\frac{1}{16}\Omega(1/M)=\Omega(1/M)$. The symmetrical reasoning shows that $\mathsf{cp}(\mathcal{A}\cdot\mathcal{A})\geq\Omega(1/M)$.

LEMMA 3.9. *Let M and ϵ be as above and let $\mathcal{A} \subseteq \mathbb{F}$ be a set such that $|\mathcal{A}| \geq M^{1-20\epsilon}$, but also such that $|\mathcal{A} \cdot \mathcal{B}| \leq M^{1+20\epsilon}$, for some $\mathcal{B}$ with $|\mathcal{B}| \geq M^{1-20\epsilon}$. Then $\mathsf{cp}(\mathcal{A}+\mathcal{C})$ is smaller than $\frac{1}{M^{1+20\epsilon}}$ for* all *sets $\mathcal{C}$ of size M.*

Proof. If $\mathsf{cp}(\mathcal{A}+\mathcal{C}) \geq \frac{1}{M^{1+20\epsilon}}$, then by applying Lemma 3.8 (with $\rho = 60\epsilon$, and assuming $\epsilon < 1/10$) it holds that there exists subsets $\mathcal{A}', \mathcal{C}'$ of $\mathcal{A}$ and $\mathcal{C}$, respectively, such that $|\mathcal{A}'|, |\mathcal{C}'| \geq M^{1-600\epsilon}$, but also such that $|\mathcal{A}' + \mathcal{C}'| \leq M^{1+600\epsilon}$. This means by Lemma 3.7 that $|\mathcal{A}' + \mathcal{A}'| \leq M^{1+2000\epsilon}$. However, this means that $|\mathcal{A}' \cdot \mathcal{A}'| \geq M^{1+10^4\epsilon}$ by Theorem 1.5, which implies that $|\mathcal{A}' \cdot \mathcal{B}| \geq M^{1+10^3\epsilon}$ by Lemma 3.7. However, since $\mathcal{A}' \subseteq \mathcal{A}$ this implies that $|\mathcal{A} \cdot \mathcal{B}| \geq M^{1+10^3\epsilon}$, contradicting our initial assumption. ∎

We call a set $\mathcal{A}$ that satisfies the conclusion of Lemma 3.9 $(M, 20\epsilon)$ *plus friendly*. That is, $\mathcal{A}$ is said to be (M, ϵ') *plus friendly* if $\mathsf{cp}(\mathcal{A}+\mathcal{C}) \leq M^{-1-\epsilon'}$ for every $\mathcal{C}$ with $|\mathcal{C}| = M$. Since M and ϵ are fixed for this proof, we will sometimes drop the prefix and simply call (M, ϵ)-plus-friendly sets *plus friendly*. Reversing the roles of addition and multiplication, we obtain the following lemma.

LEMMA 3.10. *Let M and ϵ be as above and let $\mathcal{A} \subseteq \mathbb{F}$ be a set such that $|\mathcal{A}| \geq M^{1-20\epsilon}$, but also such that $|\mathcal{A} + \mathcal{B}| \leq M^{1+20\epsilon}$, for some $\mathcal{B}$ with $|\mathcal{B}| \geq M^{1-20\epsilon}$. Then $\mathsf{cp}(\mathcal{A} \cdot \mathcal{C})$ is smaller than $\frac{1}{M^{1+20\epsilon}}$ for* all *sets $\mathcal{C}$ of size M.*

Again, we call a set $\mathcal{A}$ that satisfies the conclusion of Lemma 3.10 an $(M, 20\epsilon)$-*times-friendly* set, and again in the following we will sometimes simply call such sets *times friendly*. The main step in our proof will be the following lemma.

LEMMA 3.11. *Let $\mathcal{A} \subseteq \mathbb{F}$ with $|\mathcal{A}| = M$. Then there exist two disjoint subsets $\mathcal{A}_+$ and $\mathcal{A}_\times$ such that*

- *$\mathcal{A}_+$ is either empty or (M, ϵ) plus friendly.*
- *$\mathcal{A}_\times$ is either empty or (M, ϵ) times friendly.*
- *$|\mathcal{A} \setminus (\mathcal{A}_+ \cup \mathcal{A}_\times)| < M^{1-\epsilon}$ (i.e., $\mathcal{A}_+ \cup \mathcal{A}_\times$ capture almost all of $\mathcal{A}$.)*

Note that this lemma implies that the counterexample described above (of a distribution that is an arithmetic progression with probability $\frac{1}{2}$ and a geometric progression with probability $\frac{1}{2}$) captures in some sense the worst case for the theorem.

Proof of Lemma 3.11. We prove the lemma by repeatedly applying Lemma 3.8 to construct the sets $\mathcal{A}_+, \mathcal{A}_\times$. We start with $\mathcal{A}_+ = \emptyset$ and $\mathcal{A}_\times = \mathcal{A}$. At each point, we remove some elements from $\mathcal{A}_\times$ and add them to $\mathcal{A}_+$. We always maintain the invariant that $\mathcal{A}_+$ is either empty or plus friendly.

If $|\mathcal{A}_\times| < M^{1-\epsilon}$, then we are done (since we can then let $\mathcal{A}_\times = \emptyset$ and have $|\mathcal{A}_\times \cup \mathcal{A}_+| \geq M - M^{1-\epsilon}$). If $\mathcal{A}_\times$ is (M, ϵ) times friendly, then we are done. Otherwise, there exists $\mathcal{B}''$ of size M such that $\mathsf{cp}(\mathcal{A}_\times \cdot \mathcal{B}'') \geq \frac{1}{M^{1+\epsilon}}$, and so we can apply Lemma 3.8 to obtain subsets $\mathcal{A}'$ of $\mathcal{A}_\times$ and $\mathcal{B}'$ of $\mathcal{B}''$ such that $|\mathcal{A}'|, |\mathcal{B}'| \geq M^{1-5\epsilon}$ (but also such that $|\mathcal{A}' \cdot \mathcal{B}'| \leq M^{1+5\epsilon}$). By Lemma 3.9, $\mathcal{A}'$ will be $(M, 10\epsilon)$ (and so in particular (M, ϵ)) plus friendly, and so we can remove it from $\mathcal{A}_\times$ and add it to $\mathcal{A}_1$ (i.e., let $\mathcal{A}_+ = \mathcal{A}_+ \cup \mathcal{A}'$, $\mathcal{A}_\times = \mathcal{A}_\times \setminus \mathcal{A}'$). Note that the union of disjoint-plus-friendly sets is plus friendly with the same parameters (since a convex combination of low collision-probability distributions has low collision probability by Lemma 3.5). We continue in this way until either $\mathcal{A}_\times$ is (M, ϵ) times friendly or until $|\mathcal{A}_\times| \leq M^{1-\epsilon}$. ∎

Using Lemma 3.11, we can obtain a collision-probability analogue of Corollary 1.6.

LEMMA 3.12. *Let $\mathcal{A}, \mathcal{B}, \mathcal{C} \subseteq \mathbb{F}$ with $|\mathcal{A}| = |\mathcal{B}| = |\mathcal{C}| = M$. Then $\mathcal{A} \cdot \mathcal{B} + \mathcal{C}$ is $(M^{-\epsilon})$-close to having collision probability at most $\frac{1}{M^{1+\epsilon}}$.*

Proof. We split $\mathcal{A}$ into $\mathcal{A}_+$ and $\mathcal{A}_\times$ as per Lemma 3.11. The distribution $\mathcal{A} \cdot \mathcal{B} + \mathcal{C}$ is within $M^{-\epsilon}$ statistical distance to a convex combination of the distribution $\mathcal{X}_+ =$

$\mathcal{A}_+ \cdot \mathcal{B} + \mathcal{C}$ and the distribution $\mathcal{X}_\times = \mathcal{A}_\times \cdot \mathcal{B} + \mathcal{C}$. (Unless $\mathcal{A}_+$ or $\mathcal{A}_\times$ is empty, in which case $\mathcal{A} \cdot \mathcal{B} + \mathcal{C}$ is within $M^{-\epsilon}$ statistical distance to one of these distributions.) We will finish the proof by showing that for both distributions $\mathcal{X}_+$ and $\mathcal{X}_\times$, if the corresponding set is not empty, then the collision probability is at most $\frac{1}{M^{1+\epsilon}}$.

Showing that $\mathsf{cp}(\mathcal{A}_+ \cdot \mathcal{B} + \mathcal{C}) \leq \frac{1}{M^{1+\epsilon}}$. Using Lemma 3.5, $\mathsf{cp}(\mathcal{A}_+ \cdot \mathcal{B} + \mathcal{C}) \leq \max_{b \in \mathbb{F}} \mathsf{cp}(\mathcal{A}_+ b + \mathcal{C})$. However, $\mathsf{cp}(\mathcal{A}_+ b + \mathcal{C}) = \mathsf{cp}(\mathcal{A}_+ + \mathcal{C}b^{-1})$ since the latter distribution is a permutation of the former distribution (obtained by multiplying each element by b^{-1}). Yet the fact that $\mathcal{A}_+$ is (M, ϵ) plus friendly implies that $\mathsf{cp}(\mathcal{A}_+ + \mathcal{C}b^{-1}) \leq \frac{1}{M^{1+\epsilon}}$.

Showing that $\mathsf{cp}(\mathcal{A}_\times \cdot \mathcal{B} + \mathcal{C}) \leq \frac{1}{M^{1+\epsilon}}$. This follows immediately from the fact that $\mathsf{cp}(\mathcal{A}_\times \cdot \mathcal{B}) \leq \frac{1}{M^{1+\epsilon}}$ and since $\mathsf{cp}(\mathcal{A}_\times \cdot \mathcal{B} + \mathcal{C}) \leq \mathsf{cp}(\mathcal{A}_\times \cdot \mathcal{B})$ (as $\mathcal{A}_\times \cdot \mathcal{B} + \mathcal{C}$ is a convex combination of distributions of the form $\mathcal{A}_\times \cdot \mathcal{B} + c$ for some fixed $c \in \mathcal{C}$). □

3.3.1. Finishing up. Lemma 3.12 almost directly implies Lemma 3.1. First, we use the known fact that if the distributions $\mathcal{A}, \mathcal{B}, \mathcal{C}$ have min-entropy at least m, then the joint distribution $\mathcal{A}, \mathcal{B}, \mathcal{C}$ is a convex combination on independent distributions of the form $\mathcal{A}', \mathcal{B}', \mathcal{C}'$, where each of these is a uniform distribution on a set of size at least $M = 2^m$. Thus, it is sufficient to prove Lemma 3.12 for such distributions. By Lemma 3.12, for such distributions the distribution $\mathcal{A}' \cdot \mathcal{B}' + \mathcal{C}'$ is within $M^{-\epsilon}$ distance of having collision probability at most $\frac{1}{M^{1+\epsilon}}$ for some constant $\epsilon > 0$. Now, by applying Lemma 3.6 (with $K = M^{1+\epsilon/2}$, $L = M^{\epsilon/2}$) we obtain that $\mathcal{A}' \cdot \mathcal{B}' + \mathcal{C}'$ is within statistical distance $M^{-\epsilon} + M^{-\epsilon/4}$ from having min-entropy at least $\log(M^{(1+\epsilon/2)}) = (1 + \epsilon/2)m$.

3.4. Constructing the field $\mathbb{F}$. Recall that our extractor obtains inputs in $\{0,1\}^n$ and needs to treat these inputs as elements of a field $\mathbb{F}$ of prime order. Unfortunately, there is no known deterministic polynomial-time algorithm that on input 1^n outputs an n-bit prime (without assuming a strong number-theoretic assumption about the distance between consecutive primes). Fortunately, in our setting we can still find such a prime. The reason is that we can use some of our samples from the high-entropy distribution to obtain such a prime. To do so, we will use the following result on *seeded* dispersers (which we state here with the parameters suitable for our purposes).

THEOREM 3.13 (see [49]). *For every $\delta > 0$, there exists a constant $d > 1$ and a polynomial-time computable function $\mathsf{D} : \{0,1\}^{(10/\delta)n} \times \{0,1\}^{d \log n} \to \{0,1\}^n$ such that for every set $\mathcal{A} \subseteq \{0,1\}^{(10/\delta)n}$ with $|\mathcal{A}| \geq 2^{2n}$, it holds that*

$$\left|\mathsf{D}(\mathcal{A}, \{0,1\}^{d \log n}) \cap \{0,1\}^n\right| \geq (1 - \tfrac{1}{n^2}) \cdot 2^n.$$

Let D be the function obtained from Theorem 3.13 and identify its output with a number in $[2^n]$. We say that $x \in \{0,1\}^{(10/\delta)n}$ is "bad" if $2^n + \mathsf{D}(x,y)$ is *not* a prime number for *every* $y \in \{0,1\}^{d \log n}$. Because the set of primes has more than $1/n^2$ density in the interval $[2^n, 2 \cdot 2^n]$, the set of all bad $x \in \{0,1\}^{(10/\delta)n}$ is of size at most 2^{2n}. This means that if we take $10/\delta$ samples $x_1, \ldots, x_{10/\delta}$ from $10/\delta$ independent distributions over $\{0,1\}^n$, each of min-entropy at least δn, then the probability that the concatenation x of $x_1, \ldots, x_{10/\delta}$ is bad is exponentially low (at most $2^{2n}/2^{(10/\delta)\cdot\delta n} = 2^{-8n}$). This means that with $1 - 2^{\Omega(n)}$ probability if we enumerate over all seeds $y \in \{0,1\}^{d \log n}$ (which can be done in polynomial time), then we will find some y such that $2^n + \mathsf{D}(x,y)$ is prime. Note that we can check primality in deterministic polynomial time [1].[18] Thus, we can construct the field $\mathbb{F}$

[18] It is possible to use the same idea to run also a *probabilistic* primality testing algorithm using some additional samples from the high-entropy sources.

with very high probability by using the first $(10/\delta)n$ samples and the function D to obtain a prime $P \in [2^n, 2^{n+1}]$. We then embed the set $\{0,1\}^n$ in the field $\mathbb{F} = \mathrm{GF}(P)$. Note that since $n \geq \log P - 1$, there is no significant loss in relative entropy by this embedding.

Using nonprime fields. A different approach to solving the problem of obtaining the field $\mathbb{F}$ is to use a *nonprime* field such as $\mathrm{GF}(2^p)$ (where p now is a *small* prime and hence can be easily found) and then use Theorem 1.4 (the version proved in the appendix) instead of Theorem 1.5. This would yield the following variant of Lemma 3.1.

LEMMA 3.14. *There exists an absolute constant $c > 0$ such that for every prime p, every $\delta > 0$, and every distribution $\mathcal{A}, \mathcal{B}, \mathcal{C}$ over $\mathrm{GF}(2^p)$ with $H^\infty(\mathcal{A}), H^\infty(\mathcal{B}), H^\infty(\mathcal{C}) > \delta p$, the distribution $\mathcal{A} \cdot \mathcal{B} + \mathcal{C}$ is $(2^{-\epsilon m})$-close to having* min*-entropy at least* $\min\{(1+\epsilon)m, 0.9 \log |\mathbb{F}|\}$*, where $\epsilon = c\delta$.*

3.5. Decreasing the statistical distance. We now show how we can decrease the statistical distance by repetition. We use the following variant of an XOR-lemma.

LEMMA 3.15. *Let $\mathcal{Y}_1, \ldots, \mathcal{Y}_t$ be independent distributions over $\mathbb{F}$ such that $\mathsf{dist}(\mathcal{Y}_i, U_\mathbb{F}) < \epsilon$ for every $i = 1, \ldots, t$. Then*

$$\mathsf{dist}\left(\mathcal{Y}_1 + \cdots + \mathcal{Y}_t, U_\mathbb{F}\right) \leq \epsilon^t.$$

Proof. We can represent each distribution $\mathcal{Y}_i$ as a convex combination of the following form: With probability $(1-\epsilon)$ we get U_i (where each U_i is an independent copy of the uniform distribution) and with probability ϵ an element of some distribution $\tilde{\mathcal{Y}}_i$. Thus, one can think of the distribution $\mathcal{Y}_1 + \cdots + \mathcal{Y}_t$ as a convex combination, where with probability ϵ^t we get an element of the form $\tilde{\mathcal{Y}}_1 + \cdots + \tilde{\mathcal{Y}}_t$ and with probability $1 - \epsilon^t$ get a distribution of the form $\tilde{Y} + U_\mathbb{F}$ for some distribution $\tilde{Y}$, which is independent of $U_\mathbb{F}$. In other words, with probability $1 - \epsilon^t$ we get the uniform distribution. ∎

3.6. Proof of Lemma 3.2. Before proving Lemma 3.2, we prove the following related lemma.

LEMMA 3.16. *Let $\mathbb{F}$ be any field of size N, and let $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}$ be four independent random variables over $\mathbb{F}$, where each variable has collision probability at most $1/M$ for some $M > N^{3/4}$. Then,*

$$\mathsf{dist}\left((\mathcal{A} - \mathcal{C}) \cdot (\mathcal{B} - \mathcal{D}), U_\mathbb{F}\right) < O(N^{3/2}/M^2),$$

where $U_\mathbb{F}$ denotes the uniform distribution over $\mathbb{F}$.

We will start with a variant, where we divide rather than multiply, and where $A = C$ and $B = D$. (Throughout this section, we will always use N to denote the size of the field $\mathbb{F}$.)

LEMMA 3.17. *Let $\mathcal{A}_1, \mathcal{A}_2$ be identical independent random variables over $\mathbb{F}$ with collision probability at most $1/M$ for some $M > N^{1/2}$, and let $\mathcal{B}_1, \mathcal{B}_2$ be likewise. Then*

$$\mathsf{dist}\left((\mathcal{A}_1 - \mathcal{A}_2) \cdot (\mathcal{B}_1 - \mathcal{B}_2)^{-1}, U_\mathbb{F}\right) \leq O(N/M^2).$$

(We can ignore the event that $\mathcal{B}_1 = \mathcal{B}_2$, and we need to divide by zero since it happens with probability at most $1/M$. Thus, no matter how the value of $x/0$ is defined, it will not contribute more than $1/M$ to the statistical distance.)

Proof. Let s be an arbitrary nonzero element of $\mathbb{F}$. The distribution $\mathcal{A}_1 + s\mathcal{B}_1$ is a random variable over $\mathbb{F}$ and hence has collision probability at least $\frac{1}{N}$. Hence, we

see that

$$\Pr[(\mathcal{A}_1 + s\mathcal{B}_1) = \mathcal{A}_2 + s\mathcal{B}_2)] \geq \tfrac{1}{N}.$$

There are two ways that this equality can occur. If $\mathcal{B}_1 = \mathcal{B}_2$, then this equation can hold only if $\mathcal{A}_1 = \mathcal{A}_2$. Otherwise, it holds only if $s = (\mathcal{A}_1 - \mathcal{A}_2) \cdot (\mathcal{B}_1 - \mathcal{B}_2)^{-1}$, and hence we get that

$$\Pr[s = (\mathcal{A}_1 - \mathcal{A}_2) \cdot (\mathcal{B}_1 - \mathcal{B}_2)^{-1}] + \Pr[\mathcal{A}_{(1)} = \mathcal{A}_{(2)} \wedge \mathcal{B}_{(1)} = \mathcal{B}_{(2)}] \geq \tfrac{1}{N}.$$

Thus, $\Pr[s = (\mathcal{A}_1 - \mathcal{A}_2) \cdot (\mathcal{B}_1 - \mathcal{B}_2)^{-1}] \geq 1/N - 1/M^2$ for all $s \neq 0$, so almost all field elements are at least close to the uniform probability. The distance between two distributions D_1 and D_2 can be expressed as $2\sum_{s|D_1(s)\geq D_2(s)}(D_1(s) - D_2(s))$. Taking D_1 as the uniform distribution on $\mathbb{F}$, and D_2 as the above distribution, every s except 0 contributes at most $1/M^2$, and 0 contributes at most $1/N$ to this sum. Hence, the statistical distance is at most $2N/M^2 + 2/N = O(N/M^2)$. □

LEMMA 3.18. *Let X and Y be independent random variables on a set S of size N. Then $\mathsf{cp}(X) + \mathsf{cp}(Y) + 2\Pr[X = Y] \geq 4/N$.*

Proof. Let X be distributed according to D_0 and Y according to D_1. Then let $D_{1/2} = 1/2D_0 + 1/2D_1$. Then $D_{1/2}$ is a probability distribution on S, so $\mathsf{cp}(D_{1/2}) \geq 1/N$. On the other hand, $\mathsf{cp}(D_{1/2}) = \sum_{s\in S}(1/2D_0(s) + 1/2D_1(s))^2 = 1/4(\sum_{s\in S} D_0(s)^2 + 2\sum_{s\in S} D_0(s)D_1(s) + \sum_{s\in S} D_1(s)^2) = 1/4(\mathsf{cp}(X) + 2\Pr[X = Y] + \mathsf{cp}(Y))$. □

LEMMA 3.19. *Let $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}$ be four independent variables over $\mathbb{F}$, each with collision probability at most $1/M$, where $M > N^{1/2}$. Let $\mathcal{A}_1, \mathcal{A}_2$ be two independent variables distributed in the same way as $\mathcal{A}$, and let $\mathcal{B}_1, \mathcal{B}_2, \mathcal{C}_1, \mathcal{C}_2, \mathcal{D}_1, \mathcal{D}_2$ be distributed likewise. Then for any $s \in \mathbb{F}$, $s \neq 0$, $\Pr[s = (\mathcal{A}_1 - \mathcal{A}_2)(\mathcal{B}_2 - \mathcal{B}_1)^{-1}] + \Pr[s = (\mathcal{C}_1 - \mathcal{C}_2)(\mathcal{D}_2 - \mathcal{D}_1)^{-1}] + 2\Pr[s = (\mathcal{A} - \mathcal{C})(\mathcal{D} - \mathcal{B})^{-1}] \geq 4/N - O(1/M^2)$.*

Proof. Let X be $\mathcal{A} + s\mathcal{B}$ and Y be $\mathcal{C} + s\mathcal{D}$. Note that, by a similar case analysis of the proof of Lemma 3.17, $\mathsf{cp}(X) \leq \Pr[s = (\mathcal{A}_1 - \mathcal{A}_2)(\mathcal{B}_2 - \mathcal{B}_1)^{-1}] + 1/M^2$, $\mathsf{cp}(Y) \leq \Pr[s = (\mathcal{C}_1 - \mathcal{C}_2)(\mathcal{D}_2 - \mathcal{D}_1)^{-1}] + 1/M^2$ and $\Pr[X = Y] \leq \Pr[s = (\mathcal{A} - \mathcal{C})(\mathcal{D} - \mathcal{B})^{-1}] + 1/M^2$ (using Lemma 3.3 to bound the probabilities that $\mathcal{A} = \mathcal{C}$ and $\mathcal{B} = \mathcal{D}$). The claim then follows from Lemma 3.18. □

LEMMA 3.20. *Let $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}$ be four independent random variables over $\mathbb{F}$, each with collision probability at most $1/M$, where $M > N^{1/2}$. Then*

$$\mathsf{dist}\left((\mathcal{A} - \mathcal{C}) \cdot (\mathcal{B} - \mathcal{D})^{-1}, U_{\mathbb{F}}\right) < O(N/M^2),$$

where $U_{\mathbb{F}}$ denotes the uniform distribution over $\mathbb{F}$.

Proof. The statistical distance between $(\mathcal{A}-\mathcal{C})(\mathcal{D}-\mathcal{B})^{-1}$ and $U_{\mathbb{F}}$ can be computed as the sum of $\Pr[U_{\mathbb{F}} = s] - \Pr[(\mathcal{A}-\mathcal{C})(\mathcal{D}-\mathcal{B})^{-1} = s]$ for all $s \in \mathbb{F}$, where this difference is positive. Since $\Pr[U_{\mathbb{F}} = s] = 1/N$, using Lemma 3.19, we see that for all such $s \neq 0$ (the case $s = 0$ can add at most $1/M$ to the distance) it is the case that

$$\begin{aligned}
&\frac{1}{N} - \Pr[s = (\mathcal{A} - \mathcal{C})(\mathcal{D} - \mathcal{B})^{-1}] \\
&\leq \frac{1}{2}\Pr[s = (\mathcal{A}_1 - \mathcal{A}_2)(\mathcal{B}_2 - \mathcal{B}_1)^{-1}] - \frac{1}{N} - \frac{1}{2}\Pr[s = (\mathcal{C}_1 - \mathcal{C}_2)(\mathcal{D}_2 - \mathcal{D}_1)^{-1}] + O\left(\frac{1}{M^2}\right) \\
&\leq \frac{1}{2}\left|\Pr[s = (\mathcal{A}_1 - \mathcal{A}_2)(\mathcal{B}_2 - \mathcal{B}_1)^{-1}] - \frac{1}{N}\right| + \frac{1}{2}\left|\frac{1}{2}\Pr[s = (\mathcal{C}_1 - \mathcal{C}_2)(\mathcal{D}_2 - \mathcal{D}_1)^{-1}]\right| + O\left(\frac{1}{M^2}\right).
\end{aligned}$$

Summing this over all such s, we get that

$$\begin{aligned}&\mathsf{dist}\left((\mathcal{A}-\mathcal{C})\cdot(\mathcal{B}-\mathcal{D})^{-1},U_{\mathbb{F}}\right)\\&\leq\frac{1}{2}\mathsf{dist}\left((\mathcal{A}_1-\mathcal{A}_2)\cdot(\mathcal{B}_2-\mathcal{B}_1)^{-1},U_{\mathbb{F}}\right)+\frac{1}{2}\mathsf{dist}\left((\mathcal{C}_1-\mathcal{C}_2)\cdot(\mathcal{D}_2-\mathcal{D}_1)^{-1},U_{\mathbb{F}}\right)+O\left(\frac{N}{M^2}\right)\end{aligned}$$

which is at most $O(N/M^2)$ by Lemma 3.17. □

To finish the proof we also use the following simple observation about the relation between the L_2 and L_1 norms, or in our notation, between the statistical distance of a random variable from uniform and the difference between the collision probability of this random variable and $1/N$.

LEMMA 3.21. *Let $\mathcal{Z}$ be a random variable over $\mathbb{F}$. Let $\delta=\mathsf{dist}(\mathcal{Z},U_{\mathbb{F}})$. Then $1/N+\delta^2/N\leq\mathsf{cp}(\mathcal{Z})\leq 1/N+\delta^2$.*

Proof. Let $\delta_s=|\Pr[\mathcal{Z}=s]-1/N|$. Then $\sum_{s\in\mathbb{F}}\delta_s=\delta$, and $\sum_{s\in\mathbb{F}}\delta_s^2=\sum_s(\Pr[\mathcal{Z}=s])^2-2/N\sum_s[\Pr[\mathcal{Z}=s]+N/N^2=\mathsf{cp}(\mathcal{Z})-1/N$. By convexity, $\sum\delta_s^2$ is minimized when all $\delta_s=\delta/N$ and maximized when one $\delta_s=\delta$. Thus, $\delta^2/N\leq\mathsf{cp}(\mathcal{Z})-1/N\leq\delta^2$. □

To obtain Lemma 3.16, we note that for every two distributions $\mathcal{X},\mathcal{Y}$ over $\mathbb{F}\setminus\{0\}$, $\mathsf{cp}(\mathcal{X}\cdot\mathcal{Y})=\mathsf{cp}(\frac{\mathcal{X}}{\mathcal{Y}})$. Indeed, $x\cdot y=x'\cdot y'$ if and only if $\frac{x}{y'}=\frac{x'}{y}$. By Lemma 3.20, $\mathsf{dist}((\mathcal{A}-\mathcal{C})(\mathcal{D}-\mathcal{B})^{-1},U_{\mathbb{F}})=O(N/M^2)$. Thus, from the previous lemma, $\mathsf{cp}((\mathcal{A}-\mathcal{C})(\mathcal{D}-\mathcal{B}))=\mathsf{cp}((\mathcal{A}-\mathcal{C})(\mathcal{D}-\mathcal{B})^{-1})\leq 1/N+O(N^2/M^4)$. Then it follows from the lemma above that $\mathsf{dist}((\mathcal{A}-\mathcal{C})(\mathcal{D}-\mathcal{B}),\mathcal{U})^2/N\leq O(N^2/M^4)$, so $\mathsf{dist}((\mathcal{A}-\mathcal{C})(\mathcal{D}-\mathcal{B}),\mathcal{U})\leq O(N^{3/2}/M^2)$.

Proving Lemma 3.2. We can now prove Lemma 3.2. Indeed, for every nine distributions $\mathcal{X}_1,\ldots,sX_9$, $\mathsf{cp}(\mathsf{Ext}^2(\mathcal{X}_1,\ldots,\mathcal{X}_9))\leq\mathsf{cp}(\mathsf{Ext}^1(\mathcal{X}_1,\mathcal{X}_2,\mathcal{X}_3)\cdot\mathsf{Ext}^1(\mathcal{X}_4,\mathcal{X}_5,\mathcal{X}_6))$, since adding the additional independent distribution $\mathsf{Ext}^1(\mathcal{X}_7,\mathcal{X}_8,\mathcal{X}_9)$ cannot increase the collision probability. Hence, it is enough to prove that for $\mathcal{X}_1,\ldots,\mathcal{X}_6$ of min-entropy $0.9\log N$, the distribution $(\mathcal{X}_1\cdot\mathcal{X}_2+\mathcal{X}_3)\cdot(\mathcal{X}_4\cdot\mathcal{X}_5+\mathcal{X}_6)$ is $N^{-0.1}$ close to the uniform distribution over $\mathbb{F}$. Yet this distribution is a convex combination of distributions of the form $(\mathcal{X}_1\cdot x_2+\mathcal{X}_3)\cdot(\mathcal{X}_4\cdot x_5+\mathcal{X}_6)$ for fixed (and with very high probability nonzero) $x_2,x_5\in\mathbb{F}$. Any such distribution is of the form $(\mathcal{A}-\mathcal{B})(\mathcal{C}-\mathcal{D})$ (for $\mathcal{A}=\mathcal{X}_1\cdot x_2$, $\mathcal{B}=-\mathcal{X}_3$, $\mathcal{C}=\mathcal{X}_4\cdot x_5$, $\mathcal{D}=-\mathcal{X}_6$), and hence the result is implied by Lemma 3.16.

4. A constant-samples same-source disperser for low min-entropy. In this section we prove Theorem 1.3. That is, we construct a constant-samples same-source disperser for subsets of $\{0,1\}^n$ of size $n^{O(1)}$. A central tool we will use is the *deterministic coin tossing* technique. This technique was used in several contexts in computer science (e.g., the parallel algorithm of [13]), and it was also used, in a context very similar to ours, by Erdős and Hajnal (where it was called the "stepping-up lemma"; see section 4.7 of Graham, Rothschild, and Spencer [22]) and by Fiat and Naor [18]. By "deterministic coin tossing" we mean the function $\mathsf{ct}:\{0,1\}^{2n}\to[n]$ defined as follows: For every $x,y\in\{0,1\}^n$, $\mathsf{ct}(x,y)$ is equal to the first position i such that $x_i\neq y_i$ (if $x=y$, then we let $\mathsf{ct}(x,y)=n$). The following property of this function will be useful to us.

LEMMA 4.1. *For every $A\subseteq\{0,1\}^n$, $|\mathsf{ct}(A,A)|\geq\log|A|$.*

Proof. Let $S\subseteq[n]$ denote $\mathsf{ct}(A,A)$, and suppose for the sake of contradiction that $|A|>2^{|S|}$. Then, by the pigeonhole principle, there exist two distinct strings $x,y\in A$ such that x agrees with y on all the coordinates of S. However, for such x and y, clearly $\mathsf{ct}(x,y)\notin S$. □

The main idea behind our construction of a disperser is to apply the function ct in order to map our initial domain $\{0,1\}^n$ to the much smaller domain $[n]$ and then to use brute force to find an optimal disperser (or even extractor) on this smaller domain. In fact, we will need to apply the function *twice* to reduce our domain to the domain $[\log n]$ so we can apply brute force and obtain an optimal disperser for the smaller domain. That is, we use the following simple lemma.

LEMMA 4.2. *For every constant m and every n, it is possible to construct in $2^{O(n)}$ time the table for a function $E_{\mathsf{opt}} : [n] \times [n] \to [m]$ such that for every $A \subseteq [n]$ with $|A| > \log n + 10m$, it holds that $|E_{\mathsf{opt}}(A, A)| \geq \frac{m}{2}$.*

Proof. One way to do this construction is to go over all possible functions until we find a function satisfying this condition (there will exist such a function because a random function satisfies it with high probability). However, enumerating all possible functions will take m^{n^2}-time. Note that testing this condition only requires going over all such subsets A, which are at most $\min\{2^n, n^{10m+\log n}\}$ many.

Thus, if we reduce the number of functions to test to something smaller than this number, then we can reduce the overall time to $2^{O(n)}$. This can be done by considering functions from a sample space over $[m]^{n^2}$ which is (2^{-n})-close to being $(\log n + 10m)^2$-wise independent. There are explicit constructions for such sample spaces with $2^{O(n)}$ many points [30]. ∎

Proof of Theorem 1.3. To prove Theorem 1.3, we consider the following disperser Disp:

$$\mathsf{Disp}(x_1, \ldots, x_8) = E_{\mathsf{opt}}\left(\mathsf{ct}\big(\mathsf{ct}(x_1, x_2), \mathsf{ct}(x_3, x_4)\big), \mathsf{ct}\big(\mathsf{ct}(x_5, x_6), \mathsf{ct}(x_7, x_8)\big)\right).$$

It clearly runs in polynomial time. We will prove that for every set A of size at least n^{d2^m}, $|\mathsf{Disp}(A, \ldots, A)| \geq \frac{m}{2}$. Thus, Disp is a disperser with "statistical distance" equal to $\frac{1}{2}$. Such a disperser can be modified to obtain a disperser according to our standard definition. For example, this can be done by embedding $[m]$ in some prime field $[p]$ (with $m \leq p \leq 2m$) and letting $\mathsf{Disp}'(x_1, \ldots, x_{32}) = \mathsf{Disp}(x_1, \ldots, x_8) + \mathsf{Disp}(x_9, \ldots, x_{16}) + \mathsf{Disp}(x_1, \ldots, x_{24}) + \mathsf{Disp}(x_{25}, \ldots, x_{32})$. By the Cauchy–Davenport inequalities it will hold that for every set A as above, $\mathsf{Disp}'(A, \ldots, A) = [p]$.

To prove the bound on A, we note that by Lemma 4.1, if $|A| \geq n^{d2^m}$, then $\mathsf{ct}(\mathsf{ct}(A, A), \mathsf{ct}(A, A)) \geq \log\log n + m + \log d$. Since we apply E_{opt} on a domain of size $\log n$, by Lemma 4.2, this means that for d large enough, $|E_{\mathsf{opt}}(A, A)| \geq \frac{m}{2}$, thus finishing our proof. ∎

5. Subsequent and future work. In this work we have given the first extractors for a constant number of independent sources, each of linear entropy. Unfortunately the number of sources needed is a function of the (constant) entropy rate, and the most obvious subsequent challenge was to remove this dependency. Following this work, there has been a sequence of works which, using similar additive number-theoretic techniques and additional tools, obtained significant improvements on this and other fronts.

Barak et al. [3] gave the first construction of an extractor with a fixed number of sources that works for every linear entropy. Specifically, they give explicit deterministic extractors from three independent sources of any linear entropy. This was further improved by Raz [37], whose work needs only one of the sources to have linear entropy, while the others can have only a logarithmic amount of entropy. Building on both these works, Zuckerman [50] managed to obtain a new *seeded* extractor with shorter seed length for linear entropy and used this to obtain some improved inapproximability results for the clique and chromatic number problems. Rao [36] obtained multiple

source extractors that require only $O(\frac{\log k}{\log n})$ sources of k entropy, thus achieving a constant number of sources even for *polynomially small* entropy.

On the 2-*source* front, the work of Barak et al. [3] gave a 2-source disperser for sources of entropy δn for arbitrarily small δ (and in particular $\delta < 1/2$), thus improving on the previous explicit constructions for bipartite Ramsey graphs. Pudlak [34] gave a very simple construction of such a disperser for the case $\delta = \frac{1}{2} - \epsilon$ for some small $\epsilon > 0$, and Bourgain [7] gave a two source *extractor* for this case. Barak et al. [4] gave a 2-source disperser for sources of entropy n^δ for arbitrarily small δ (and in particular $\delta < 1/2$), thus beating the Frankl–Wilson [19] construction for a (nonbipartite) Ramsey graph.

In addition, Barak et al. [3] also gave a 1-*source* disperser for *affine* sources (a uniform distribution of an affine subspace) of linear entropy. Bourgain [8] gave an *extractor* with the same parameters, and Gabizon and Raz [20] gave different affine extractors with incomparable parameters.

To summarize, after nearly 20 years of no progress, there seem to be new and exciting ideas, tools, and results on deterministic extraction from independent sources, and we expect more to soon follow.

Appendix. Proof of Theorem 1.4. In this section, we prove Theorem 1.4 with a better explicit dependence between the constants ϵ and δ (namely, $\epsilon = \Theta(\delta)$). That is, we prove the following theorem.

THEOREM A.1. *There exists an absolute constant $c_0 > 0$ such that for every field $\mathbb{F}$ and $\delta > 0$ such that $|\mathbb{F}|^{1/k}$ is* not *an integer for every integer $2 \le k \le (2/\delta)$, and every set $\mathcal{A} \subseteq \mathbb{F}$ with $|\mathbb{F}|^\delta < |\mathcal{A}| < |\mathbb{F}|^{1-\delta}$, it holds that*

$$\max\{|\mathcal{A}+\mathcal{A}|, |\mathcal{A}\cdot\mathcal{A}|\} \ge |\mathcal{A}|^{1+c_0\delta}.$$

Note that Theorem A.1 indeed implies Theorem 1.4 since a finite field $\mathbb{F}$ has a subfield of size M if and only if $M = |\mathbb{F}|^{1/k}$ for some integer k.

We prove Theorem A.1 by proving the following two claims.

CLAIM A.2. *There exists a fixed-size uniform rational expression $r(\cdot)$ such that for every δ, every field $\mathbb{F}$ satisfying the conditions of Theorem* A.1*, and every set $\mathcal{B} \subseteq \mathbb{F}$ with $|\mathcal{B}| \ge |\mathbb{F}|^\delta$,*

$$|r(\mathcal{B}, \ldots, \mathcal{B})| \ge \min\{|\mathcal{B}|^{1+\delta}, |\mathbb{F}|\}.$$

By a *rational expression* we mean an expression involving only the operations $+, -, \cdot, /$, and variable names, but no coefficients (for example, $(x_1 - x_2)/(x_2 \cdot x_3 + x_4)$).[19] By *fixed-size* we mean that the number of operations and variables in the expression does not depend on δ, $\mathbb{F}$, or $\mathcal{A}$. A rational function is obviously a division of two polynomials. We say that a polynomial is *uniform* if all its monomials have the same degree. We say that a rational function is *uniform* if it is a division of two uniform polynomials. In fact, the expression r obtained from the proof of Claim A.2 will be very simple: it will be a uniform 16-variable rational expression with both the nominator and the denominator of degree 2.

CLAIM A.3. *For every uniform rational expression $r(\cdot)$ there is a constant c (depending on $r(\cdot)$) such that if $\mathcal{A} \subseteq \mathbb{F}$ satisfies $|\mathcal{A}+\mathcal{A}|, |\mathcal{A}\cdot\mathcal{A}| \le |\mathcal{A}|^{1+\rho}$ (for sufficiently small $\rho > 0$), then there exists a set $\mathcal{B} \subseteq \mathcal{A}$ with $|\mathcal{B}| \ge |\mathcal{A}|^{1-c\rho}$, but $|r(\mathcal{B}, \ldots, \mathcal{B})| \le |\mathcal{A}|^{1+c\rho}$.*

[19] Throughout this section we define $x/0$ as 0 (it will not matter much since we will always have that the event that the denominator is zero in the expression has negligible probability).

We note that this actually holds for nonuniform rational expressions as well, but the proof is slightly easier for uniform expressions.

Proof of Theorem A.1 *using Claims* A.2 *and* A.3. Claims A.2 and A.3 together imply Theorem A.1. Indeed, let $r(\cdot)$ be the rational expression obtained from Claim A.2 and let c be the constant (depending on r) obtained from Claim A.3. If for $\delta, \mathbb{F}$ and $\mathcal{A}$, as in the conditions of the theorem, both $|\mathcal{A}+\mathcal{A}|$ and $|\mathcal{A}\cdot\mathcal{A}|$ are less than $|\mathcal{A}|^{1+\delta/(10c)}$, then there is a subset $\mathcal{B}$ of size at least $|\mathcal{A}|^{1-c\delta/(10c)} \geq |\mathbb{F}|^{\delta/2}$ such that $|r(\mathcal{B},\ldots,\mathcal{B})| \leq |\mathcal{A}|^{1+\delta/10} < |\mathcal{B}|^{1+\delta/2}$, and thus we obtain a contradiction.

We note that Claim A.3 follows almost immediately from Lemmas 2.4 and 3.1 in [9]. Nevertheless, for the sake of completeness, we do provide a proof of Claim A.3 (following lines similar to the proofs of [9]) in section A.2. We now move to the proof of Claim A.2.

A.1. Proof of Claim A.2. To prove Claim A.2 we will prove the following even simpler claim.

CLAIM A.4. *Let $\mathbb{F}$ be any field and let $\mathcal{B} \subseteq \mathbb{F}$ and $k \in \mathbb{N}$ (with $k \geq 2$) be such that $|\mathbb{F}|^{1/k} < |\mathcal{B}| \leq |\mathbb{F}|^{1/(k-1)}$. Then $|\frac{\mathcal{B}-\mathcal{B}}{\mathcal{B}-\mathcal{B}}| \geq |\mathbb{F}|^{1/(k-1)}$.*

Proof. Suppose otherwise that $|\frac{\mathcal{B}-\mathcal{B}}{\mathcal{B}-\mathcal{B}}| < |\mathbb{F}|^{1/(k-1)}$. Thus, we can find $s_1 \notin \frac{\mathcal{B}-\mathcal{B}}{\mathcal{B}-\mathcal{B}}$. Similarly, if $k > 2$, then we can find $s_2 \notin \frac{\mathcal{B}-\mathcal{B}}{\mathcal{B}-\mathcal{B}} + s_1\frac{\mathcal{B}-\mathcal{B}}{\mathcal{B}-\mathcal{B}}$ (since this set is of size at most $|\frac{\mathcal{B}-\mathcal{B}}{\mathcal{B}-\mathcal{B}}|^2 < |\mathbb{F}|$). In this way we define inductively $s_1,\ldots,s_{k-1}$ such that for $1 < i \leq k-1$,

$$s_i \notin \tfrac{\mathcal{B}-\mathcal{B}}{\mathcal{B}-\mathcal{B}} + s_1\tfrac{\mathcal{B}-\mathcal{B}}{\mathcal{B}-\mathcal{B}} + \cdots + s_{i-1}\tfrac{\mathcal{B}-\mathcal{B}}{\mathcal{B}-\mathcal{B}}.$$

Consider the function $f(x_0,\ldots,x_{k-1}) = x_0 + s_1x_1 + \cdots + s_{k-1}x_{k-1}$. This is a function from $\mathcal{B}^k$ to $\mathbb{F}$, where $|\mathcal{B}|^k > |\mathbb{F}|$, and hence it has a *collision*. That is, there are two vectors $\vec{x} = (x_0,\ldots,x_{k-1})$ and $\vec{x}\,' = (x'_0,\ldots,x'_{k-1})$ such that $\vec{x} \neq \vec{x}\,'$, but $f(\vec{x}) = f(\vec{x}\,')$. If we let i be the maximum index such that $x_i \neq x'_i$, we see that

$$(x_0 - x'_0) + s_1(x_1 - x'_1) + \ldots + s_{i-1}(x_{i-1} - x'_{i-1}) = s_i(x'_i - x_i).$$

Dividing by $(x'_i - x_i)$, we get that $s_i = y_0 + s_1y_1 + \cdots + s_{i-1}y_{i-1}$, where all the y_i's are members of $\frac{\mathcal{B}-\mathcal{B}}{\mathcal{B}-\mathcal{B}}$, contradicting our choice of s_i. □

To prove Claim A.2 we let $\mathbb{F}$,δ, $\mathcal{B}$ be as stated in the theorem and choose k such that $|\mathbb{F}|^{1/k} < |\mathcal{B}| \leq |\mathbb{F}|^{1/(k-1)}$. By one invocation of $\frac{\mathcal{B}-\mathcal{B}}{\mathcal{B}-\mathcal{B}}$, we get to a set of size at least $|\mathbb{F}|^{1/(k-1)}$, but since that is not an integer, this set is of size *larger* than $|\mathbb{F}|^{1/(k-1)}$ (this is for $k > 2$; for $k = 2$ we get to the entire field $\mathbb{F}$). Thus, if we compose this expression two times (i.e., let $r(x_1,\ldots,x_{16}) = \frac{r'(x_1,\ldots,x_4)-r'(x_5,\ldots,x_8)}{r'(x_9,\ldots,x_{12})-r'(x_{13},\ldots,x_{16})}$, where $r'(a,b,c,d) = \frac{a-b}{c-d}$), then we get that for $k > 2$,

$$|r(\mathcal{B},\ldots,\mathcal{B})| \geq |\mathbb{F}|^{\frac{1}{k-2}} = |\mathbb{F}|^{\frac{1}{k-1}(1+\frac{1}{k-2})} \geq |\mathcal{B}|^{1+\delta},$$

where for $k = 2$, $r(\mathcal{B},\ldots,\mathcal{B}) = \mathbb{F}$.

A.2. Proof of Claim A.3. We now prove Claim A.3. Before turning to the actual proof, we state two number-theoretic lemmas which we will use. These lemmas are variants of the lemmas presented in section 3.2.

A.2.1. More number-theoretic lemmas.

Stronger form of Gowers's lemma. We will use the following generalized and stronger form of Lemma 3.8 (see [21, 9] and [43, Claim 4.4]). Because it is such a useful

lemma, we state it below in the most general (and, unfortunately, also cumbersome) form.

LEMMA A.5. *Let $\mathcal{A}_1, \ldots, \mathcal{A}_k$ be subsets of some group G with $|\mathcal{A}_i| = M$ for all $i \in [k]$. Then there exists $C = C(k)$ such that for every $\rho > 0$, if $\mathsf{cp}(\sum_{i=1}^k \mathcal{A}_i) \geq \frac{1}{M^{1+\rho}}$, then there are k subsets $\mathcal{A}'_1, \ldots, \mathcal{A}'_k$ with $\mathcal{A}'_i \subseteq \mathcal{A}_i$ and $|\mathcal{A}'_i| \geq M^{1-C\rho}$ for every $i \in [k]$ satisfying*

$$\left|\sum_{i=1}^k \mathcal{A}'_i\right| \leq M^{1+C\rho}. \tag{A.1}$$

Moreover, (A.1) *is demonstrated by the fact that every element $z \in \sum_{i=1}^k \mathcal{A}'_i$ can be represented in $M^{\ell-1-C'\rho}$ different ways as a sum $z = y_1 + \cdots + y_\ell$ (for $C' = C'(k)$ and $\ell = 2k^2 - k$), where each of the y_j's is a member of A_i or $-A_i$ for some $i = i(j)$ (with the choice of a sign also being a function of j).*

Furthermore, if for all $i \in [k]$, $\mathcal{A}_i = \mathcal{A}$ or $\mathcal{A}_i = -\mathcal{A}$ for some set $\mathcal{A}$, then all the subsets $\mathcal{A}'_i$ are of the form $\mathcal{A}'_i = \mathcal{A}'$ or $\mathcal{A}'_i = -\mathcal{A}'$ for some subset $\mathcal{A}' \subseteq \mathcal{A}$.

It is easy to see that by applying Lemma A.5 for $k = 2$ we get Lemma 3.8 (perhaps with 10 replaced by a different constant). Lemma A.5 can be proven by a generalization of the proof of Lemma 3.8; see [43].[20] When using Lemma A.5, we will always be in the case that we have an upper bound on the *set size* of $\sum_{i=1}^k \mathcal{A}_i$ (i.e., $|\sum_{i=1}^k \mathcal{A}_i| \leq M^{1+\rho}$) and not just a lower bound on its collision probability.

We note that in the proofs below we will use several times this technique of showing that some set $\mathcal{B}$ is not much larger than M by showing that every $b \in \mathcal{B}$ can be represented in roughly $M^{\ell-1}$ ways as a sum of ℓ elements, each from some set $\mathcal{D}$ of size M.

Sumset estimates. We will also use the following lemma, which is a variant of Lemma 3.7.

LEMMA A.6 (see [33, 40]). *Let $\mathcal{A}, \mathcal{B}$ be subsets of some Abelian group G with $|\mathcal{A}| = |\mathcal{B}| = M$ and let $\rho > 0$ be some number. If $|\mathcal{A} + \mathcal{B}| \leq M^{1+\rho}$, then*

$$|\mathcal{A} \pm \underbrace{\mathcal{B} \pm \cdots \pm \mathcal{B}}_{h \text{ times}}| \leq M^{1+2h\rho}.$$

We note that this lemma immediately implies a similar result for sets that are of slightly different sizes. That is, if $|\mathcal{A}| = M$ and $|\mathcal{B}| = M^{1-\epsilon}$, then we can break up $\mathcal{A}$ to $\mathcal{A}_1, \ldots, \mathcal{A}_{M^\epsilon}$ that are of the same size as $\mathcal{B}$. We then have that $|\mathcal{A}_i + \mathcal{B}| \leq |\mathcal{A} + \mathcal{B}| \leq M^{1+\rho}$. and hence the lemma implies that for every i,

$$\mathcal{A}_i \pm \underbrace{\mathcal{B} \pm \cdots \pm \mathcal{B}}_{h \text{ times}}| \leq M^{1+2h\rho}.$$

However, $\mathcal{A} \pm \mathcal{B} \pm \cdots \pm \mathcal{B}$ is just the union of $\mathcal{A}_i \pm \mathcal{B} \pm \cdots \pm \mathcal{B}$ for all i, and hence we get that

$$|\mathcal{A} \pm \underbrace{\mathcal{B} \pm \cdots \pm \mathcal{B}}_{h \text{ times}}| \leq M^{1+2h\rho+\epsilon}.$$

[20] We note that this lemma is not stated exactly in this form in [43]. Rather, Claim 4.4 there states that under these conditions every such z can be represented in roughly M^{2k-2} ways as a sum of $2k-1$ elements w_l, where each of these elements can be represented in roughly M^{k-1} ways as a sum of k elements in the A_i's. It is also stated in [43] in terms of distance from having high min-entropy rather than high collision probability; see footnote 16.

Similarly if $|\mathcal{B}| = M$ and $|\mathcal{A}| = M^{1-\epsilon}$, then

$$|\mathcal{A} \pm \underbrace{\mathcal{B} \pm \cdots \pm \mathcal{B}}_{h \text{ times}}| \leq M^{1+2h\rho+h\epsilon}$$

since if we split sB into $\mathcal{B}_1, \ldots, \mathcal{B}_{M^\epsilon}$, then we have that $\mathcal{A} \pm \mathcal{B} \pm \cdots \pm \mathcal{B}$ is the union of sets of the form $\mathcal{A} \pm \mathcal{B}_{i_1} \pm \cdots \pm \mathcal{B}_{i_h}$ for all $i_1, \ldots, i_h \in [M^\epsilon]$.

A.2.2. The actual proof. In fact, to prove Claim A.3 it is enough to prove the following claim.

CLAIM A.7. *For every integer $k > 0$, there exists a constant $C = C(k) > 0$ such that for every $\rho > 0$, if $\mathcal{A} \subseteq \mathbb{F}$ satisfies $|\mathcal{A} + \mathcal{A}|, |\mathcal{A} \cdot \mathcal{A}| \leq |\mathcal{A}|^{1+\rho}$, then there is a set $\mathcal{B} \subseteq \mathcal{A}$ such that $|\mathcal{B}| \geq |\mathcal{A}|^{1-C\rho}$, but $|\mathcal{B}^k - \mathcal{B}^k| \leq |\mathcal{A}|^{1+C\rho}$.*

Obtaining Claim A.3 *from Claim* A.7. Claim A.7 implies Claim A.3 by applying Lemma A.6. Indeed, suppose that r is a rational expression, where both the numerator and denominator have at most k' monomials, each of degree k' (if one of them has monomials of smaller degree than k', we can multiply with a uniform polynomial to make the degree k'). Let $k = 4k'^2$ and let $\mathcal{B}$ be the subset of $\mathcal{A}$ obtained from Claim A.7 such that $|\mathcal{B}^k - \mathcal{B}^k|$ is at most $|\mathcal{A}|^{1+C\rho}$ for some constant $C = C(k)$. Since $|\mathcal{B}| \geq |\mathcal{A}|^{1-C\rho}$, this means that $|\mathcal{B}^k - \mathcal{B}^k| \leq |\mathcal{B}|^{1+C'\rho}$ for some different constant C'. This implies by Lemma A.6 that $|k\mathcal{B}^k| \leq |\mathcal{B}|^{1+2C'k\rho}$. Now, $k\mathcal{B}^k \supseteq \mathcal{C} \stackrel{def}{=} (k'\mathcal{B}^{k'})(k'\mathcal{B}^{k'})$, and hence the size of $\mathcal{C}$ is also at most $|\mathcal{B}|^{1+2C'k\rho}$. Applying Lemma A.6 again we get that $\left|\frac{k'\mathcal{B}^{k'}}{k'\mathcal{B}^{k'}}\right| \leq |\mathcal{A}|^{1+4C'k\rho}$. However, $r(\mathcal{B}, \ldots, \mathcal{B}) \subseteq \frac{k'\mathcal{B}^{k'}}{k'\mathcal{B}^{k'}}$ and hence we are done.

Proof of Claim A.7. We now turn to proving Claim A.7. Let $\mathcal{A}$ be a set satisfying the conditions of the claim, and denote $M = |\mathcal{A}|$. We will prove that for some constant $C = C(k) > 0$, there is a subset $\mathcal{B} \subseteq \mathcal{A}$ of size at least $M^{1-C\rho}$ such that any member of the set $\mathcal{B}^k - \mathcal{B}^k$ can be represented in at least $M^{\ell-1-C\rho}$ different ways as a sum $d_1 + \cdots + d_\ell$, where all the elements d_i come from a set $\mathcal{D}$ of size at most $M^{1+C\rho}$ for some $\ell = \ell(k)$. This clearly implies that $|\mathcal{B}^k - \mathcal{B}^k| \leq M^{1+C'\rho}$ for some constant C', thus proving the claim.

Proof idea: The case $k = 2$. To illustrate the proof idea, we now sketch the proof for the case $k = 2$. The proof for general k follows in exactly the same way, although with more cumbersome notation. Under the conditions of the claim, $|\mathcal{A}+\mathcal{A}| \leq |\mathcal{A}|^{1+\rho}$, and hence $\mathsf{cp}(\mathcal{A} - \mathcal{A}) = \mathsf{cp}(\mathcal{A} + \mathcal{A}) \geq |\mathcal{A}|^{1-\rho}$. Hence, by Lemma A.5, there exists a set $\mathcal{A}' \subseteq \mathcal{A}$ with $|\mathcal{A}'| \geq M^{1-C\rho}$ (where C is some absolute constant independent of ρ) such that not only $|\mathcal{A}' - \mathcal{A}'| \leq M^{1+C\rho}$ but actually this fact can be demonstrated by the fact that every element of $\mathcal{A}' - \mathcal{A}'$ can be represented in at least $M^{5-C\rho}$ different ways in the form $a_1 - a_2 + a_3 - a_4 + a_5 - a_6$, where for $i = 1, \ldots, 6$, $a_i \in \mathcal{A}$.[21]

We know that every member of $\mathcal{A}' - \mathcal{A}'$ can be represented in roughly M^5 different ways as $a_1 - a_2 + a_3 - a_4 + a_5 - a_6$ with the a_i's in $\mathcal{A}$. Now, if we multiply this by an arbitrary element of $\mathcal{A}$, we get that every member of $(\mathcal{A}' - \mathcal{A}')\mathcal{A}$ can be represented in roughly M^5 different ways as $b_1 - b_2 + b_3 - b_4 + b_5 - b_6$ with the b_i's in $\mathcal{A} \cdot \mathcal{A}$. Since by the conditions of the claim, $\mathcal{A} \cdot \mathcal{A}$ is also of roughly size M, we get that the set $(\mathcal{A}' - \mathcal{A}')\mathcal{A}$ is also "not large" (i.e., of size $M^{1+C'\rho}$ for some absolute constant C'). Now consider an element $y - z$ of the set $\mathcal{A}'^2 - \mathcal{A}'^2$. For the sake of simplicity, we assume for a moment (*with* loss of generality) that any y in $\mathcal{A}' \cdot \mathcal{A}'$ can be represented in roughly

[21] For simplicity of exposition we assumed that the pattern of $+$ and $-$ signs that is obtained from Lemma A.5 is as written above. The proof clearly follows through regardless of the fixed pattern we use.

M different ways as $y = y_1 y_2$ with $y_1, y_2 \in \mathcal{A}'$.[22] Since $z \in \mathcal{A}' \cdot \mathcal{A}'$, it is equal to $z_1 z_2$ for some $z_1, z_2 \in \mathcal{A}'$. Every representation $y_1 y_2$ of y induces a representation of $y - z = y_1 y_2 - z_1 z_2$ as $(y_1 - z_1)y_2 + z_1(y_2 - z_2)$, and so we get that every element of $\mathcal{A}'^2 - \mathcal{A}'^2$ can be represented as $d_1 - d_2$, with $d_1, d_2 \in \mathcal{D} \stackrel{def}{=} (\mathcal{A}' - \mathcal{A}')\mathcal{A}$ in roughly M different ways. However, since we already showed that the size of $\mathcal{D}$ is roughly M, this proves that $\mathcal{A}'^2 - \mathcal{A}'^2$ is also of size roughly M, and hence we are done (for the case $k = 2$).

Proving for general k. We now turn to proving the claim rigorously and for any k. Recall that our goal is to find a not-too-small subset $\mathcal{B}$ of $\mathcal{A}$ such that $\mathcal{B}^k - \mathcal{B}^k$ can be represented in roughly $M^{\ell-1}$ ways as a sum of ℓ elements from a set $\mathcal{D}$ that is not too large. We will start by defining the set $\mathcal{D}$.

Let ℓ be some number, and let $a, b \in \mathcal{A}'$ (where $\mathcal{A}'$ is obtained from Lemma A.5 as above) and $c \in \mathcal{A}'^{\ell}$. By the same reasoning as above, we get that the element $(a-b)c$ can be represented in at least $M^{5-C\rho}$ different ways as $a_1c - a_2c + a_3c - a_4c + a_5c - a_6c$, or in other words, it can be represented in at least $M^{5-C'\rho}$ different ways as $b_1 - b_2 + b_3 - b_4 + b_5 - b_6$ for $b_i \in \mathcal{A}^{\ell+1}$ for $i = 1, \ldots, 6$. By the multiplicative version of Lemma A.6, $|\mathcal{A}^{\ell+1}| \leq M^{1+3\ell\rho}$, and hence the set $(\mathcal{A}' - \mathcal{A}')\mathcal{A}'^{\ell}$ is "small" (i.e., of size at most $M^{6(1+3\ell\rho)} M^{-(5-C'\rho)} \leq M^{1+20C'\ell\rho}$). Using again the multiplicative version of Lemma A.6 (setting $\mathcal{A} = (\mathcal{A}' - \mathcal{A}')\mathcal{A}'^{\ell-1}$, $\mathcal{B} = \mathcal{A}'$), we get that the set $(\mathcal{A}' - \mathcal{A}')\mathcal{A}'^{\ell}\mathcal{A}'^{-\ell'}$ is also "small" (i.e., of size at most $M^{1+C''\rho}$ for some constant $C'' = C''(\ell, \ell')$).[23] Using the fact that $(\mathcal{A}'^{-1} - \mathcal{A}'^{-1}) \subseteq (\mathcal{A}' - \mathcal{A}')\mathcal{A}'^{-2}$, we get that for any ℓ_1, ℓ_2, the set $\mathcal{D}_{\ell_1,\ell_2}$ defined as

$$\mathcal{D}_{\ell_1,\ell_2} = (\mathcal{A}'^{-1} - \mathcal{A}'^{-1})\mathcal{A}'^{\ell_1}\mathcal{A}'^{-\ell_2} \cup (\mathcal{A}' - \mathcal{A}')\mathcal{A}'^{\ell_1}\mathcal{A}'^{-\ell_2}$$

is of size at most $M^{1+C'''\rho}$ for some constant C''' depending on ℓ_1, ℓ_2. We define $\mathcal{D} \stackrel{def}{=} \cup_{\ell_1+\ell_2=\ell s} D_{\ell_1,\ell_2}$ for ℓ as obtained by Lemma A.5 (i.e., $\ell = 2k^2 - k$). As desired, we have that $|\mathcal{D}| \leq M^{1+C\rho}$, where C is a constant depending on ℓ.

Defining the set $\mathcal{B}$. We now turn to defining the required set $\mathcal{B}$. Utilizing Lemma A.5 again, we obtain that for some absolute constant $D = D(k)$, there is a set $\mathcal{B} \subseteq \mathcal{A}'$ with $|\mathcal{B}| \geq M^{1-D\rho}$, and every element in $\mathcal{B}^k$ can be represented in $M^{\ell-1-D\rho}$ ways as $a_1 \cdots a_\ell$, where a_i is in $\mathcal{A}'$ or in $\mathcal{A}'^{-1}$ for $i = 1, \ldots, \ell$ (for $\ell = 2k^2 - k$). Let $y - z$ be a member of $\mathcal{B}^k - \mathcal{B}^k$. Fix one representation $z = z_1 \cdots z_\ell$ of z as a multiplication of elements of $\mathcal{A}'$ or $\mathcal{A}'^{-1}$. The element y can be represented $M^{\ell-1-D\rho}$ times as $y = y_1 \cdots y_\ell$ with $y_i \in \mathcal{A}'$ for $i = 1, \ldots, \ell$. For each such representation we define $d_i = y_1 \cdot \ldots \cdot y_{i-1}(y_i - z_i)z_{i+1} \cdots z_\ell$. Note that $\sum_{i=1}^{\ell} d_i = \prod_{i=1}^{\ell} y_i - \prod_{i=1}^{\ell} z_j = y - z$. Also note that for every i, $d_i \in \mathcal{D}$, where $\mathcal{D}$ is the set defined above. The map $(y_1, \ldots, y_\ell) \mapsto (d_1, \ldots, d_\ell)$ is one-to-one (indeed, given $z_1, \ldots, z_\ell$ we can recover $y_1, \ldots, y_\ell$ from $d_1, \ldots, d_\ell$). Hence, we get that each member of $\mathcal{B}^k - \mathcal{B}^k$ can be represented in $M^{\ell-1-D\rho}$ different ways as $\sum_{i=1}^{\ell} d_i$ with $d_i \in \mathcal{D}$, implying that $|\mathcal{B}^k - \mathcal{B}^k| \leq M^{1+D'\rho}$ for $D' = D'(k)$. ∎

Acknowledgments. We thank Amir Shpilka for many valuable discussions during the early stages of this research. We also thank the anonymous referee for many useful comments and corrections on an earlier version of this manuscript.

[22] We will not be able to get to that situation in the actual proof that follows, but we will approximate it using the multiplicative version of Lemma A.5.

[23] Again, the case of division by zero does not matter, but for simplicity, we can just remove 0 from the set $\mathcal{A}'$ if it is there.

REFERENCES

[1] M. Agrawal, N. Kayal, and N. Saxena, *PRIMES Is in P*, Technical report, Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, India, 2002.

[2] N. Alon, *Tools from higher algebra*, in Handbook of Combinatorics, Vol. 1, 2, Elsevier, Amsterdam, 1995, pp. 1749–1783.

[3] B. Barak, G. Kindler, R. Shaltiel, B. Sudakov, and A. Wigderson, *Simulating independence: New constructions of condensers, Ramsey graphs, dispersers, and extractors*, in Proceedings of the 46th STOC 05, pp. 1–10, 2005.

[4] B. Barak, A. Rao, R. Shaltiel, and A. Wigderson, 2-*source dispersers for subpolynomial entropy and Ramsey graphs beating the Frankl-Wilson construction*, in Proceedings of the 38th Annual ACM Symposium on Theory of Computing, ACM, New York, 2006, pp. 671–680.

[5] B. Barak, R. Shaltiel, and E. Tromer, *True random number generators secure in a changing environment*, in Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems (CHES), 2003, pp. 166–180. Lecture Notes in Comput. Sci 2779, Springer, Berlin.

[6] M. Blum, *Independent unbiased coin flips from a correlated biased source: A finite state Markov chain*, in Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 1984, pp. 425–433.

[7] J. Bourgain, *More on the sum-product phenomenon in prime fields and its applications*, Internat. J. Number Theory, 1 (2005), pp. 1–32.

[8] J. Bourgain, *On the Construction of Affine Extractors*, Geometric and Functional Analysis, to appear.

[9] J. Bourgain, N. Katz, and T. Tao, *A sum-product estimate in finite fields, and applications*, Geom. Funct. Anal., 14 (2004), pp. 27–57.

[10] B. Chor and O. Goldreich, *Unbiased bits from sources of weak randomness and probabilistic communication complexity*, in Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 1985, pp. 429–442.

[11] B. Chor, O. Goldreich, J. Hastad, J. Friedman, S. Rudich, and R. Smolensky, *The bit extraction problem of t-resilient functions (preliminary version)*, in Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 1985, pp. 396–407.

[12] A. Cohen and A. Wigderson, *Dispersers, deterministic amplification, and weak random sources*, in Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 1989, pp. 14–19.

[13] R. Cole and U. Vishkin, *Deterministic coin tossing and accelerating cascades: Micro and macro techniques for designing parallel algorithms*, in Proceedings of the 18th Annual ACM Symposium on Theory of Computing, ACM, New York, 1986, pp. 206–219.

[14] Y. Dodis and R. Oliveira, *On extracting private randomness over a public channel*, in Approximation, Randomness, and Combinatorial Optimization, Springer, Berlin, 2003, pp. 252–263.

[15] Y. Dodis and J. Spencer, *On the (non)universality of the one-time pad*, in Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 2002, pp. 376–388.

[16] A. Elbaz, *Improved Constructions for Extracting Quasi-Random Bits from Sources of Weak Randomness*, Master's thesis, Weizmann Institute of Science, Rehovot, Israel, 2003.

[17] P. Erdős and E. Szemerédi, *On sums and products of integers*, in Studies in Pure Mathematics, Birkhäuser, Basel, 1983, pp. 213–218.

[18] A. Fiat and M. Naor, *Implicit $O(1)$ probe search*, SIAM J. Comput., 22 (1993), pp. 1–10.

[19] P. Frankl and R. M. Wilson, *Intersection theorems with geometric consequences*, Combinatorica, 1 (1981), pp. 357–368.

[20] A. Gabizon and R. Raz, *Deterministic extractors for affine sources over large fields*, in Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 2005, pp.407–416.

[21] W. T. Gowers, *A new proof of Szemerédi's theorem for arithmetic progressions of length four*, Geom. Funct. Anal., 8 (1998), pp. 529–551.

[22] R. L. Graham, B. L. Rothschild, and J. L. Spencer, *Ramsey Theory*, John Wiley & Sons, New York, 1980.

[23] R. L. Graham and J. H. Spencer, *A constructive solution to a tournament problem*, Canad. Math. Bull., 14 (1971), pp. 45–48.

[24] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby, *A pseudorandom generator from any one-way function*, SIAM J. Comput., 28 (1999), pp. 1364–1396.

[25] J. Kamp and D. Zuckerman, *Deterministic extractors for bit-fixing sources and exposure-resilient cryptography*, in Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 2003, pp. 92–101.

[26] S.V. Konyagin, *A Sum-Product Estimate in Fields of Prime Order*, Arxiv technical report, http://arxiv.org/abs/math.NT/0304217 (2003).

[27] C. J. Lu, O. Reingold, S. Vadhan, and A. Wigderson, *Extractors: Optimal up to constant factors*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, ACM, New York, 2003, pp. 602–611.

[28] J. L. McInnes and B. Pinkas, *On the impossibility of private key cryptography with weakly random keys*, in Crypto '90, Lecture Notes in Comput. Sci. 537, Springer, Berlin, 1990, pp. 421–436.

[29] E. Mossel and C. Umans, *On the complexity of approximating the VC dimension*, J. Comput. System Sci., 65 (2002), pp. 660–671.

[30] J. Naor and M. Naor, *Small-bias probability spaces: Efficient constructions and applications*, SIAM J. Comput., 22 (1993), pp. 838–856.

[31] M. B. Nathanson, *Additive Number Theory. Inverse Problems and the Geometry of Sumsets*, Graduate Texts in Math. 165, Springer-Verlag, New York, 1996.

[32] Y. Peres, *Iterating von Neumann's procedure for extracting random bits*, Ann. Statist., 20 (1992), pp. 590–597.

[33] H. Plünnecke, *Eine zahlentheoretische Anwendung der Graphentheorie*, J. Reine Angew. Math., 243 (1970), pp. 171–183.

[34] P. Pudlak, *On Explicit Ramsey Graphs and Estimates of the Number of Sums and Products*, Topics in Discrete Mathematics, Klazar, Kratochvil, Loebl, Matousek, Thomas, and Valtar, Springer, 2006, pp. 169–175.

[35] P. Pudlak and V. Rodl, *Pseudorandom sets and explicit constructions of Ramsey graphs*, Quaderni di Matematica, Vol. 13, J. Krajicek, ed., Dipartimanto di Matematica, Seconda Universita di Napoli, Caserta, 2004, pp. 327–346.

[36] A. Rao, *Extractors for a constant number of polynomially small min-entropy independent sources*, in Proceedings of the 38th Annual ACM Symposium on Theory of Computing, ACM, New York, 2006, pp. 497–506.

[37] R. Raz, *Extractors with weak random seeds*, in Proceedings of the 37th Annual ACM Symposium on Theory of Computing, ACM, New York, 2005, pp. 11–20.

[38] O. Reingold, M. Saks, and A. Wigderson, *personal communication*, 2003.

[39] I. Ruzsa, *An application of graph theory to additive number theory*, Sci. Ser. A: Math. Sci., 3 (1989), pp. 97–109.

[40] I. Ruzsa, *Sums of finite sets*, in Number Theory (New York, 1991–1995), Springer, New York, 1996, pp. 281–293.

[41] M. Santha and U. V. Vazirani, *Generating quasi-random sequences from slightly-random sources*, in Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 1984, pp. 434–440.

[42] R. Shaltiel, *Recent developments in extractors*, Bull. Eur. Assoc. Theoret. Comput. Sci. EATCS No. 77, (2002), pp. 67–95.

[43] B. Sudakov, E. Szemerédi, and V. H. Vu, *On a question of Erdös and Moser*, Duke Math. J., 129 (2005), pp. 129–155.

[44] L. Trevisan and S. Vadhan, *Extracting randomness from samplable distributions*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 2000, pp. 32–42.

[45] U. Vazirani, *Strong communication complexity or generating quasirandom sequences from two communicating semirandom sources*, Combinatorica, 7 (1987), pp. 375–392.

[46] U. Vazirani, *Efficiency considerations in using semi-random sources*, in Proceedings of the 19th Annual ACM Symposium on Theory of Computing, ACM, New York, 1987, pp. 160–168.

[47] J. von Neumann, *Various techniques used in connection with random digits*, Appl. Math. Ser. 12 (1951), pp. 36–38.

[48] D. Zuckerman, *General weak random sources*, in Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 1990, pp. 534–543.

[49] D. Zuckerman, *Simulating BPP using a general weak random source*, Algorithmica, 16 (1996), pp. 367–391.

[50] D. Zuckerman, *Linear degree extractors and the inapproximability of max clique and chromatic number*, in Proceedings of the 38th Annual ACM Symposium on Theory of Computing, ACM, New York, 2006, pp. 681–690.

SIAM J. COMPUT.
Vol. 36, No. 4, pp. 1119–1159

ON WORST-CASE TO AVERAGE-CASE REDUCTIONS FOR NP PROBLEMS*

ANDREJ BOGDANOV† AND LUCA TREVISAN‡

Abstract. We show that if an NP-complete problem has a nonadaptive self-corrector with respect to any samplable distribution, then coNP is contained in NP/poly and the polynomial hierarchy collapses to the third level. Feigenbaum and Fortnow [*SIAM J. Comput.*, 22 (1993), pp. 994–1005] show the same conclusion under the stronger assumption that an NP-complete problem has a nonadaptive random self-reduction. A self-corrector for a language L with respect to a distribution $\mathcal{D}$ is a *worst-case to average-case reduction* that transforms any given algorithm that correctly decides L on *most* inputs (with respect to $\mathcal{D}$) into an algorithm of comparable efficiency that decides L correctly on *every* input. A random self-reduction is a special case of a self-corrector, where the reduction, given an input x, is restricted to only making oracle queries that are distributed according to $\mathcal{D}$. The result of Feigenbaum and Fortnow depends essentially on the property that the distribution of each query in a random self-reduction is independent of the input of the reduction. Our result implies that the average-case hardness of a problem in NP or the security of a one-way function cannot be based on the worst-case complexity of an NP-complete problem via nonadaptive reductions (unless the polynomial hierarchy collapses).

Key words. average-case complexity, worst-case to average-case reduction, one-way function

AMS subject classification. 68Q17

DOI. 10.1137/S0097539705446974

1. Introduction.

The fundamental question in the study of average-case complexity is whether there exist distributional problems in NP that are intractable on average. A distributional problem in NP is a pair $(L, \mathcal{D})$, where L is a decision problem in NP, and $\mathcal{D}$ is a samplable distribution on instances. We will say that such a problem is tractable on average if for every polynomial p there exists a polynomial-time algorithm A such that for all sufficiently large n, when given a random instance of length n from distribution $\mathcal{D}$, algorithm A determines membership in L correctly with probability at least $1 - 1/p(n)$.

This notion of average-case tractability is essentially equivalent to Impagliazzo's definition of *heuristic* polynomial-time algorithms [28]. Impagliazzo observes that if every problem in distributional NP is tractable on average, then there are no one-way functions, and thus cryptography is impossible. It is therefore generally believed that distributional NP does contain problems that are intractable on average.

The question we consider in this paper concerns the minimal complexity assumption one needs to make in order to guarantee that distributional NP does indeed contain a problem that is intractable on average. Ideally, one would like to base the existence of hard on average problems (and one-way functions) on a worst-case assumption, namely NP $\not\subseteq$ BPP. Equivalently, the question can be formulated as

*Received by the editors January 28, 2005; accepted for publication (in revised form) December 28, 2005; published electronically December 15, 2006. A preliminary version of this paper appeared in *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science* (FOCS), 2003, pp. 308–317.

http://www.siam.org/journals/sicomp/36-4/44697.html

†School of Mathematics, Institute for Advanced Study, Einstein Drive, Princeton, NJ 08540 (adib@ias.edu). This work was done while the author was at the University of California at Berkeley.

‡Computer Science Division, University of California at Berkeley, Berkeley, CA 94720 (luca@cs.berkeley.edu). This author's research was supported by a Sloan Research Fellowship, an Okawa Foundation grant, and NSF grants CCR-9984703 and CCR-0406156.

follows: Is the existence of worst-case hard problems in NP sufficient to show the existence of problems in NP that are hard on average?

In the cryptographic setting, the question of whether there are cryptosystems that are NP-hard to break, that is, whose security can be based on the assumption that NP $\not\subseteq$ BPP, is as old as modern cryptography itself, and it was asked by Diffie and Hellman [14, section 6]. As we review below, there is conflicting evidence about whether the answer to this question exists.

Previous work.

Worst-case versus average-case in NP. Impagliazzo [28] observes that we know oracles relative to which NP $\not\subseteq$ P/poly, but there is no intractable problem in distributional NP (and consequently, one-way functions do not exist). Therefore, any proof that NP $\not\subseteq$ BPP implies the existence of an intractable problem in distributional NP must use a nonrelativizing argument. However, nonrelativizing arguments are commonly used in lattice-based cryptography to establish connections between the worst-case and average-case hardness of certain NP problems that are believed to be intractable (but not NP-complete).

Feigenbaum and Fortnow [17] consider the notion of a *random self-reduction*, which is a natural, and possibly nonrelativizing, way to prove that the average-case complexity of a given problem relates to its worst-case complexity. We begin by discussing the slightly more general notion of *locally random reduction*, introduced in [6] (see also the earlier works [1, 5, 18]). A locally random reduction from a language L to a distributional problem $(L', \mathcal{D})$ is a polynomial-time oracle procedure R such that $R^{L'}$ solves L and, furthermore, each oracle query of $R^{L'}(x)$ is distributed according to $\mathcal{D}$.[1] Clearly, such a reduction converts a heuristic polynomial-time algorithm for $(L', \mathcal{D})$ (with sufficiently small error probability) into a BPP algorithm for L. Observe that the reduction may depend on the choice of distributional problem $(L', \mathcal{D})$, so in general this approach does not relativize. If we had a locally random reduction from, say, satisfiability to some problem $(L', \mathcal{D})$ in distributional NP, then we would have proved that if NP $\not\subseteq$ BPP, then distributional NP contains intractable problems. A locally random reduction from L to $(L', \mathcal{D})$ is called a *random self-reduction* if $L = L'$.

Feigenbaum and Fortnow show that if for an NP-complete language L and a samplable ensemble $\mathcal{D}$ there is a *nonadaptive* random self-reduction from L to $(L, \mathcal{D})$, then NP $\subseteq$ coNP/poly and the polynomial hierarchy collapses to the third level. Their proof also establishes the slightly more general result that if there is a nonadaptive locally random reduction from a problem L to a problem $(L', \mathcal{D})$ in distributional NP, then L is in coNP/poly.

Random self-reductions and locally random reductions are natural notions, and they have been used to establish the worst-case to average-case equivalence of certain PSPACE-complete and EXP-complete problems [39]. Therefore, the result of Feigenbaum and Fortnow rules out a natural and general approach to proving a statement of the following form: "If NP $\not\subseteq$ BPP, then distributional NP contains intractable problems."

Cryptography versus NP*-hardness.* The seminal work of Diffie and Hellman [14] introducing public key cryptography asked if there exists a public key encryption scheme whose hardness can be based on an NP-complete problem. This question was seemingly answered in the affirmative by Even and Yacobi [16], who devised a public

[1] More precisely, according to the restriction of $\mathcal{D}$ to inputs of length polynomially related to x (see section 2).

key cryptosystem that is "NP-hard to break." Namely, they showed a reduction that transforms any adversary that "breaks" the cryptosystem into an algorithm that solves SAT. However, the notion of "breaking the cryptosystem" in [16] is a worst-case one: Specifically, it is assumed that the adversary can break the encryption for *every* key. Lempel [31] later showed that the same cryptosystem can in fact be broken on *most* keys. Therefore, the NP-hardness of breaking a cryptosystem in the worst case does not in general have any implications for cryptographic security.

The gap between worst-case and average-case hardness is even more transparent in the case of symmetric key cryptography, or one-way functions. It is well known that there exist "one-way functions" that are NP-hard to invert in the worst case, but easy to invert on average. (For instance, consider the function that maps a formula ϕ and an assignment a to $(1, \phi)$ if a satisfies ϕ and to $(0, \phi)$ otherwise.)

As these examples show, the notion of hardness in breaking a public key cryptosystem, or inverting a one-way function that one needs in cryptography, is fundamentally an average-case notion.

Brassard [12] addresses the question of the existence of public key cryptosystems that are hard to break from a different perspective. He argues that, under some assumptions on the key-generation algorithm and the encryption procedure, the problem of breaking the encryption is in $\mathrm{NP} \cap \mathrm{coNP}$. Specifically, he shows that if there is a reduction that transforms an oracle that breaks encryptions into an algorithm for a language L, then the reduction can be used to provide NP certificates for membership in both L and $\overline{L}$, proving that $L \in \mathrm{NP} \cap \mathrm{coNP}$. More recent work by Goldreich and Goldwasser [23] reaches the same conclusion under weaker assumptions.

In the setting of symmetric key cryptography, a similar conclusion can be reached about the hardness of inverting a one-way function if one makes additional assumptions about the function in question. For instance, if the function f is a permutation, then the existence of a reduction from any language L to an inverter for f establishes that $L \in \mathrm{NP} \cap \mathrm{coNP}$. A proof for membership in L or $\overline{L}$ consists of the transcript of all the queries made by the reduction, together with unique preimages of the queries under f. The fact that f is a permutation guarantees that this transcript perfectly simulates the reduction when given access to an inverting oracle for f.

These arguments explain why the hardness of breaking a large class of cryptosystems cannot be based on the worst-case complexity of an NP complete problem (assuming $\mathrm{NP} \neq \mathrm{coNP}$). However, neither of them uses the fact that the reduction that transforms the adversary into an algorithm for L is correct even if the adversary only performs its task well on average. In fact, the arguments merely assume that the reduction behaves correctly when given oracle access to an adversary that violates a *worst-case* assumption. Given the existence of public key cryptosystems and one-way functions that are hard to break in the worst case, one cannot expect these arguments to explain why breaking a general one-way function or a general public key encryption scheme should be an $\mathrm{NP} \cap \mathrm{coNP}$ problem, as experience seems to indicate, if this is indeed the case.

If we were to ever hope for such an explanation, we need a stronger notion of "NP hard to break" which allows for the fact that the adversary may err on some fraction of inputs. Again, what we mean by a cryptosystem being "NP-hard to break" is that there exists a reduction that transforms an adversary for the cryptosystem into an algorithm for SAT, but now the reduction is required to solve SAT correctly even if the adversary sometimes outputs an incorrect answer.

This motivates the following definition of a reduction from an NP-complete problem to the problem of inverting well on average a one-way function f: A reduction is

an oracle probabilistic polynomial-time procedure R such that for some polynomial p and for every oracle A that inverts f on a $1 - 1/p(n)$ fraction of inputs of length n, it holds that R^A is a BPP algorithm for SAT. The techniques of Feigenbaum and Fortnow imply that if R is nonadaptive, and if all of its oracle queries are done according to the same distribution (that depends only on the length of the input), then the existence of such a reduction implies that the polynomial hierarchy collapses to the third level.

As we explain below, we reach the same conclusion (see Theorem 4.1 in section 4) without any assumption on the distribution of the queries made by R^A, but also assuming as in [17] that the queries are made nonadaptively.

Worst-case to average-case reductions within NP. The most compelling evidence that the average-case hardness of certain problems in NP *can* be based on worst-case intractability assumptions comes from lattice-based cryptography.

Ajtai [3] shows that an algorithm that solves well on average the shortest vector problem (which is in NP) under a certain samplable distribution of instances implies an algorithm that solves, in the worst case, an approximate version of the shortest vector problem. The latter can be seen as an NP promise problem. If the latter problem were NP-complete, then we would have a reduction relating the average-case hardness of a distributional problem in NP to the worst-case hardness of an NP-complete problem. Unfortunately, the latter problem is known to be in $\mathrm{NP} \cap \mathrm{coNP}$, and therefore it is unlikely to be NP-hard. However, it is conceivable that improved versions of Ajtai's argument could show the equivalence between the average-case complexity of a distributional NP problem and the worst-case complexity of an NP problem. Micciancio [34] and Micciancio and Regev [35] improve Ajtai's reduction by showing that a good-on-average algorithm for the generalized subset sum problem implies better worst-case approximation algorithms for a variety of problems on lattices. Such approximations, however, still correspond to promise problems known to be in $\mathrm{NP} \cap \mathrm{coNP}$.

Average-case complexity in NP. The theory of average-case complexity was pioneered by Levin [32], who defined the notion of "efficient-on-average" algorithms and gave a distributional problem that is complete for a large subclass of distributional NP. Levin's notion of efficient-on-average algorithms is stronger than Impagliazzo's notion of polynomial-time heuristic algorithms that we consider here. Namely, every problem in distributional NP that admits an efficient-on-average algorithm also admits an efficient heuristic algorithm.

The subclass of distributional NP problems considered by Levin imposes a severe restriction on the distribution according to which instances of the problem are sampled. In particular, it does not include the problem of inverting arbitrary one-way functions. In the case of a one-way function f, the notion of "inverting f well on average" amounts to solving the search problem "Given u, find x s.t. $f(x) = u$," where u is chosen according to the distribution obtained by applying f to the uniform distribution. In general, f may be an arbitrary polynomial-time algorithm, so it makes sense to relax the definition so as to allow instances of L to be generated by arbitrary polynomial-time samplers. This yields the class distributional NP (introduced by Ben-David and others [7]) of all pairs $(L, \mathcal{D})$, where L is an NP language and $\mathcal{D}$ is an arbitrary samplable distribution according to which inputs for L are generated.

The class distributional NP turns out to be surprisingly robust for (randomized) heuristic algorithms. In particular, there exists an NP language L such that if L is tractable on average with respect to the uniform distribution, then every problem in NP is tractable on average with respect to any samplable distribution. Moreover, the average-case algorithms for distributional NP are "search algorithms" in the sense

that they provide witnesses of membership for most of the "yes" instances. In particular, average-case tractability of L implies the ability to efficiently invert one-way functions on most inputs $f(x)$, where x is chosen uniformly at random. These results on distributional NP were established by Ben-David and others [7] and Impagliazzo and Levin [29].

For an overview of these and other notions in average-case complexity, their interrelations, and explanations of the various choices made in definitions, the reader is referred to the expository papers by Impagliazzo [28], Goldreich [20], and the authors [11].

Our result. A worst-case to average-case reduction with parameter δ from a language L to a distributional problem $(L', \mathcal{D})$ is a probabilistic polynomial-time oracle procedure R such that, for every oracle A that agrees with L' on inputs of probability mass $1 - \delta$ according to $\mathcal{D}$ on each input length, R^A solves L on every input.

If L and L' are the same language, then the reduction is called a self-corrector, a notion independently introduced by Blum, Luby, and Rubinfeld [10], and by Lipton [33] in the context of program checking [8, 9]. As argued below, a locally random reduction is also a worst-case to average-case reduction and a random self-reduction is also a self-corrector, but the reverse need not be true.

In this paper we show that if there is a worst-case to average-case reduction with parameter $1/\text{poly}(n)$ from an NP-complete problem L to a distributional NP problem $(L, \mathcal{D})$, then NP $\subseteq$ coNP/poly and the polynomial hierarchy collapses to the third level. In particular, if an NP-complete problem has a self-corrector with respect to a samplable distribution, then the polynomial hierarchy collapses to the third level.

We first prove the result for the special case in which the distribution $\mathcal{D}$ is uniform (Theorem 4.1). Then, using reductions by Impagliazzo and Levin [29] and by Ben-David and others [7], we show that the same is true even if the reduction assumes a good-on-average algorithm for the *search* version of L', even if we measure average-case complexity for L' with respect to an arbitrary samplable distribution $\mathcal{D}$ (Theorem 5.3).

The generalization to arbitrary samplable distributions and to search problems also implies that there cannot be any nonadaptive reduction from an NP-complete problem to the problem of inverting a one-way function.

Our result also rules out nonadaptive reductions from an NP-complete problem to the problem of breaking a public key cryptosystem. The constraint of nonadaptivity of the reduction is incomparable to the constraints in the results of Goldreich and Goldwasser [23].

It should be noted that some of the worst-case to average-case reductions of Ajtai [3], Ajtai and Dwork [4], Micciancio [34], Regev [38], and Micciancio and Regev [35] are *adaptive*. Micciancio and Regev [35] observe that their reductions can be made nonadaptive with a slight loss in worst-case approximation factors.

Comparison with Feigenbaum and Fortnow [17]. It is easy to see that a locally random reduction R from L to L' that makes q queries, each of which is generated by the reduction according to a distribution $\mathcal{D}$, is also a worst-case to average-case reduction with parameter $\Omega(1/q)$ from L to $(L', \mathcal{D})$. Indeed, if A is an oracle that has agreement, say, $1 - 1/4q$ with L' (as measured by $\mathcal{D}$), and we access the oracle via q queries, each distributed according to $\mathcal{D}$, queries made to A and queries made to L' are answered in the same way with probability at least $3/4$.

For the result of Feigenbaum and Fortnow, it is not necessary that the distribution of each query made by the reduction be exactly $\mathcal{D}$, but it is essential that the marginal distribution of queries made by the reduction be independent of the reduction's input.

This restriction is quite strong, and in this sense, the result of [17] is extremely sensitive: If one modifies the distribution of queries even by an exponentially small amount that depends on the input, all statistical properties of the reduction are preserved, but one can no longer draw the conclusion of [17]. Our result reaches the same conclusion as [17], yet allows the queries made by the reduction to depend arbitrarily on the input.

One natural setting, where the queries made by the reduction seem to essentially depend on the input, is Levin's theory of average-case complexity. One tool for relating the average-case hardness of two distributional problems is the "average-case to average-case reduction." Such a reduction from $(L, \mathcal{D})$ and $(L', \mathcal{D}')$ uses an oracle that solves L' on most inputs chosen from $\mathcal{D}'$ to solve L on most inputs according to $\mathcal{D}$. Some important reductions, most notably those in [7, 29], choose their queries to the oracle from a distribution that depends on the input in an essential way, making the results of [17] useless for their study.

The relation between locally random reductions and our notion of worst-case to average-case reduction is similar to the relation between one-round private information retrieval and locally decodable codes [13, 40]. In one-round private information retrieval, a "decoder" is given oracle access to the encoding of a certain string, and wants to retrieve one bit of the string by making a bounded number of queries; the restriction is that the ith query must have a distribution independent of the bit that the decoder is interested in. In a locally decodable code, a decoder is given oracle access to the encoding of a certain string, and the encoding has been corrupted in a δ fraction of places; the decoder wants to retrieve a bit of the original string by making a bounded number of queries (with no restriction on the distribution on queries). An intermediate notion that is useful in the study of the relation between private information retrieval and locally decodable codes is that of a smooth decoder: Such a decoder satisfies the additional requirement that the distribution of each query should be dominated by the uniform distribution. Similarly, in the setting of worst-case to average-case reductions one can restrict attention to smooth reductions, where the distribution of queries made by the reduction is dominated by the uniform distribution.

For computationally unbounded decoders, it has been shown (see [30, 24]) that uniform, smooth, and general decoders are equivalent, but the same methods do not work in the computationally bounded setting studied in this paper. One step in our proof is, however, inspired by the techniques used to show this equivalence.

Our proof. As in the work of Feigenbaum and Fortnow, we use the fact that problems in coNP/poly cannot be NP-complete unless the polynomial hierarchy collapses to the third level. Our goal is to show that if L has a $1/\mathrm{poly}(n)$ worst-case to average-case reduction to a language $(L', \mathcal{D})$ in distributional NP, then L is in $\mathrm{NP/poly} \cap \mathrm{coNP/poly}$. In particular, if L were NP-complete, then NP would be contained inside coNP/poly, which in particular implies the collapse of the polynomial hierarchy to the third level. (However, the conclusion $\mathrm{NP} \subseteq \mathrm{coNP/poly}$ appears weaker than the more standard statement $\mathrm{NP} = \mathrm{coNP}$.)

Feigenbaum and Fortnow observe that NP/poly is exactly the class of languages which admit AM protocols with polynomial length advice. Then they show $L \in \mathrm{NP/poly} \cap \mathrm{coNP/poly}$ by giving AM protocols with advice for both L and its complement. The protocols for L and its complement are completely analogous, so we focus on describing the protocol for L.

We begin by discussing the case of a reduction from L to $(L', \mathcal{D})$ when $\mathcal{D}$ is the

uniform distribution.

The Feigenbaum–Fortnow protocol. Let us first briefly review the proof of Feigenbaum and Fortnow [17]. Given x, a prover wants to prove that $R^{L'}(x)$ accepts with high probability (implying $x \in L$), where R makes q nonadaptive queries, each uniformly distributed. The (nonuniform) verifier generates k independent computations of $R^{L'}(x)$ and sends to the prover all the kq queries generated in all the k runs. The prover has to provide all the answers, and certificates for all the "yes" answers. The verifier, nonuniformly, knows the overall fraction p of queries of $R^{L'}(x)$ whose answer is "yes" (recall that we assumed that the queries of R, and thus p, are independent of x). If k is large enough, the verifier expects the number of "yes" answers from the prover to be concentrated around kqp, and in fact this is within $kqp \pm O(q\sqrt{k})$ with high probability. If the prover gives fewer than $kqp - O(q\sqrt{k})$ "yes" answers, this provides strong evidence of cheating, and the verifier rejects. Since a cheating prover must provide certificates for all its "yes" claims, such a prover can cheat only by saying "no" on a "yes" query, and cannot do so on more than $O(q\sqrt{k})$ instances. If k is sufficiently larger than q, then with high probability either the verifier rejects, or a majority of the k computations of $R^{L'}(x)$ yields the correct answer, making the reduction output "yes" and the verifier accept.

Handling smooth reductions: The hiding protocol. The Feigenbaum–Fortnow protocol can be used with *every* oracle procedure R, provided that given x we can get a good estimate of the average fraction p_x of oracle queries made by R on input x that are answered "yes" by an oracle for L'. In general, this fraction will depend on x, so it cannot be provided as advice to the AM circuit certifying membership in L.

For starters, let us allow the distribution of R's queries to depend on x, but restrict it to being "α-smooth": We assume that every query y of R is generated with probability at most $\alpha 2^{-|y|}$. (It is useful to think of α as constant, or at most polynomial in $|y|$, so that a query made by R is not much more likely to hit any specific string than in the uniform distribution.) We devise an AM protocol with advice in which the verifier either rejects or gets a good estimate of p_x. This estimate is then fed into the Feigenbaum–Fortnow protocol to obtain an AM circuit for L.

Suppose that, given a *random* query y made by $R(x)$, we could force the prover to reveal whether or not $y \in L'$. Then by sampling enough such queries y, we can estimate p_x as the fraction of "yes" queries made by the reduction. But how do we force the prover to reveal if $y \in L'$? The idea is to hide the query y among a sequence of queries $z_1, \ldots, z_k$, for which we *do* know whether $z_i \in L'$, in such a way that the prover cannot tell where in the sequence we hid our query y. In such a case, the prover is forced to give a correct answer for y, for if he were to cheat he wouldn't know where in the sequence to cheat, and thus would likely be caught.

The problem is that we do not know a specific set of queries z_i with the desired property. We do, however, know that if we choose z_i independently from the uniform distribution on $\{0,1\}^{|y|}$, then with high probability $pk \pm O(\sqrt{k})$ of these queries will end up in L', where p is the probability that a *uniformly random* query in $\{0,1\}^{|y|}$ is in L'. Since p depends only on the length of x but not on x itself, it can be given to the verifier nonuniformly.

This suggests the following verifier strategy: Set $k = \omega(\alpha^2)$, generate k uniformly random queries $z_1, \ldots, z_k$ of length n, hide y among $z_1, \ldots, z_k$ by inserting it at a random position in the sequence, send all the queries to the prover, and ask for membership in L', together with NP-witnesses that at least $pk - O(\sqrt{k})$ queries belong to L'.

We claim that, with high probability, either the verifier rejects or the answer about

membership of y in L' must be correct. Intuitively, a cheating prover can give at most $O(\sqrt{k})$ wrong answers. The prover wants to use this power wisely and assign one of these wrong answers to the query y. However, smoothness ensures that no matter how the prover chooses the set of $O(\sqrt{k})$ queries to cheat on, it is very unlikely that the query y falls into that set.

For ease of analysis, the actual proof presented in section 3.2 combines into a single step the step of sampling enough ys to estimate p_x with the step of hiding y among a sequence of uniform queries.

This argument already provides an interesting generalization to [17]. Notice that we have not yet used the fact that the reduction is allowed access to any oracle that computes L' well on average.

Handling general reductions. We now consider the case of general reductions, allowing the distribution of a random query on input x to depend arbitrarily on x. Observe that the hiding protocol will, in general, fail to estimate p_x for this type of reduction. If a particular query y made by the reduction is very likely (that is, it occurs with probability much greater than $\alpha 2^{-|y|}$), then it cannot be hidden in a reasonably long sequence of uniform queries.

However, suppose that the verifier had the ability to identify queries y that occur with probability $\geq \alpha 2^{-|y|}$; let us call such queries "heavy" and the other ones "light." The fraction of heavy queries in the uniform distribution is at most $1/\alpha$. Suppose also that the prover answers all light queries correctly. We can then use R to certify membership in L as follows: If the query made by R is heavy, pretend that the oracle for R answered "no"; otherwise use the answer provided by the prover. This process simulates exactly a run of the reduction R^A, where A is an oracle that agrees with L' on all the light queries, and answers "no" on all the heavy queries. In particular, A agrees with L' on a $1 - 1/\alpha$ fraction of inputs, so the reduction is guaranteed to return the correct answer.

In general, the verifier cannot identify which queries made by the reduction are heavy and which are light. However, suppose the verifier knew the probability q_x that a random query, on input x, is heavy. Then, among any set of k independent queries, the verifier expects to see, with high probability, $q_x k \pm O(\sqrt{k})$ heavy queries. Using a protocol of Goldwasser and Sipser [26], the verifier can now obtain approximate AM certificates of heaviness for at least $q_x k - O(\sqrt{k})$ queries from the prover. This leaves at most $O(\sqrt{k})$ queries about whose heaviness the verifier may be misinformed.

A verifier with access to q_x can run a variant of the hiding protocol to calculate the fraction p_x of "yes" instances of L' among the light queries (treating the $O(\sqrt{k})$ heavy queries that "slip in the sample" as a statistical error to this estimate), followed by a variant of the Feigenbaum–Fortnow protocol, simulating "no" answers on all the heavy queries.

Finally, we need a protocol that helps the verifier estimate the probability q_x of heavy queries. The verifier can obtain an approximate lower bound on q_x by sampling random queries and asking for proof that each query is heavy. To obtain an approximate upper bound on q_x, the verifier uses an "upper bound" protocol for the size of certain NP sets due to Fortnow [19]. The explanation of the exact roles of these protocols in estimating q_x is deferred to section 3.1.

We observe that the generalization of the Feigenbaum–Fortnow result about locally random reductions to smooth, and then arbitrary, nonadaptive reductions parallels an analogous sequence of steps establishing the equivalence of uniform, smooth, and arbitrary decoders for locally decodable codes.

General distributions, search problems, and one-way functions. So far we have described our results for the case in which the distribution on inputs $\mathcal{D}$ is the uniform distribution. We now consider the case where $\mathcal{D}$ is an arbitrary samplable distribution. Impagliazzo and Levin [29] show that for every distributional NP problem $(L, \mathcal{D})$ and bound $\delta = n^{-O(1)}$ there is a nonadaptive probabilistic polynomial-time oracle algorithm R, an NP language L', and a bound $\delta' = \delta^{O(1)}$ such that for every oracle A that agrees with L' on a $1 - \delta'$ fraction of inputs, R^A solves L on a subset of inputs of density $1 - \delta$ under the distribution $\mathcal{D}$.

This means that if there were a nonadaptive worst-case to average-case reduction with parameter $1/\text{poly}(n)$ from a problem L to a distributional problem $(L', \mathcal{D})$, there would also be such a reduction from L to $(L'', \mathcal{U})$, where $\mathcal{U}$ is the uniform distribution and L'' is in NP. By the previously described results, this would imply the collapse of the polynomial hierarchy.

A reduction by Ben-David and others [7] implies that for every distributional NP problem $(L, \mathcal{U})$ there is a problem L' in NP such that an algorithm that solves the decision version of $(L', \mathcal{U})$ on a $1 - \delta$ fraction of inputs can be modified (via a nonadaptive reduction) into an algorithm that solves the search version of $(L, \mathcal{U})$ on a $1 - \delta \cdot \text{poly}(n)$ fraction of input. This implies that even if we modify the definition of worst-case to average-case reduction so that the oracle A is supposed to solve the *search* version of the problem, our results still apply. In particular, for every polynomial-time computable function f, the problem of inverting f well on average is precisely the problem of solving well on average a distributional NP search problem. Therefore our results also rule out the possibility of basing one-way functions on NP-hardness using nonadaptive reductions.

Organization. Section 2 provides the relevant definitions of notions in average-case complexity and interactive proof systems. In section 3 we present the protocols for estimating the fraction of heavy queries of a reduction, the fraction of light "yes" queries of a reduction, and for simulating the reduction, respectively. Section 4 contains the proof of our main result (Theorem 4.1) concerning the average-case complexity of languages with respect to the uniform distribution. In section 5 we prove our result for the average-case complexity of distributional search problems (Theorem 5.3).

2. Preliminaries. In this section we formalize the notions from average-case complexity and interactive proof systems needed to state and prove our result on the impossibility of worst-case to average-case reductions in NP.

For a distribution $\mathcal{D}$, we use $x \sim \mathcal{D}$ to denote a sample x chosen according to $\mathcal{D}$. For a finite set S, we use $x \sim S$ to denote a sample x chosen uniformly at random from S. For a sample x, we use $\mathcal{D}(x)$ to denote the probability of x in the distribution $\mathcal{D}$. For a set S, we use $\mathcal{D}(S)$ to denote the probability that a random sample chosen according to $\mathcal{D}$ falls inside the set S.

2.1. Distributional problems and heuristic algorithms. Intuitively, we think of an algorithm A as a "good heuristic algorithm" for distributional problem $(L, \mathcal{D})$ if the set of "yes" instances of A (which we also denote by A) and the set L are close according to $\mathcal{D}$. Formalizing this definition requires one to make choices regarding how $\mathcal{D}$ measures closeness and what the threshold for closeness is.

Roughly, Levin [32] considers two sets A and L to be close according to $\mathcal{D}$ if on a *random* input length n, the measure of the symmetric difference $A \triangle L$ according to the restriction of $\mathcal{D}$ on $\{0,1\}^n$ is small. We will make the stronger requirement that this quantity be small for *all* n. Notice that every heuristic algorithm satisfying the stronger requirement also satisfies the weaker requirement (and therefore reductions

that work for algorithms satisfying the weaker requirement also work for algorithms satisfying the stronger requirement), so the stronger requirement makes our impossibility result more general. This requirement simplifies some of the definitions, as we can now restrict our attention to ensembles of distributions over various input lengths rather than a single distribution over $\{0,1\}^*$.

We now turn to the actual definitions.

DEFINITION 2.1 (samplable ensemble). *An* efficiently samplable ensemble of distributions *is a collection $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \dots\}$, where $\mathcal{D}_n$ is a distribution on $\{0,1\}^n$ for which there exists a probabilistic polynomial-time sampling algorithm S that, on input 1^n, outputs a sample from $\mathcal{D}_n$.*[2]

The *uniform ensemble* is the ensemble $\mathcal{U} = \{\mathcal{U}_1, \mathcal{U}_2, \dots\}$, where $\mathcal{U}_n$ is the uniform distribution on $\{0,1\}^n$.

A *distributional problem* is a pair $(L, \mathcal{D})$, where L is a language and $\mathcal{D}$ is an ensemble of distributions. A distributional problem $(L, \mathcal{D})$ is in the class *distributional* NP, denoted DistNP, if L is in NP and $\mathcal{D}$ is efficiently samplable.

In this paper we study hypothetical reductions that might be used to establish average-case intractability of distributional NP problems based on a worst-case assumption, such as NP $\not\subseteq$ BPP. The notion of average-case intractability that we have in mind is the absence of good-on-average algorithms of the following type. (The definition is in the spirit of the treatment by Impagliazzo [28].)

DEFINITION 2.2 (heuristic polynomial time). *We say that a probabilistic polynomial-time algorithm A is a* heuristic algorithm with success probability $s(n)$ *for a distributional problem $(L, \mathcal{D})$ if, for every n, $\Pr_{x \sim \mathcal{D}_n}[A(x) = L(x)] \geq s(n)$, where the probability is taken over the sampling of x from $\mathcal{D}_n$ and over the internal coin tosses of A. The class of distributional problems for which such algorithms exist is denoted by* $\mathrm{Heur}_{s(n)}\mathrm{BPP}$.

We consider a distributional problem $(L, \mathcal{D})$ to be "hard on average" if there is a polynomial p such that $(L, \mathcal{D}) \notin \mathrm{Heur}_{1-1/p(n)}\mathrm{BPP}$. This is a fairly robust notion with respect to the choice of p: Trevisan [40] proves that there is a constant $c > 0$ such that for every polynomial p,

$$\text{DistNP} \not\subseteq \mathrm{Heur}_{1-1/p(n)}\mathrm{BPP} \text{ iff } \text{DistNP} \not\subseteq \mathrm{Heur}_{1/2+(\log n)^{-c}}\mathrm{BPP}.$$

Stronger collapses are known for nonuniform heuristic classes [37, 27].

Levin's alternative notion of an "efficient-on-average" algorithm [32] imposes the additional requirement that the average-case algorithm A be errorless: For every x for which $A(x) \neq L(x)$, $A(x)$ must output "fail" (with high probability over its internal coin tosses).[3] Hence every efficient-on-average algorithm for $(L, \mathcal{D})$ is also a heuristic algorithm for $(L, \mathcal{D})$, so the class of problems (in distributional NP) that are "hard for efficient-on-average algorithms" is possibly larger than the class of problems that are "hard for heuristic algorithms." Therefore opting for "hard for heuristic algorithms" as our notion of average-case hardness makes our impossibility result weaker, but in fact our result holds even with respect to the notion of "hard for efficient-on-average algorithms" (as explained in section 4).

[2] This definition restricts the strings in the support of $\mathcal{D}_n$ to having length exactly n. It is possible to use a more relaxed definition in which the length of strings in the support of $\mathcal{D}_n$ is variable, as long as S is nonshrinking: Namely, the length of every string in the support of $\mathcal{D}_n$ must be at least n^ϵ for some constant $\epsilon > 0$.

[3] Levin's original definition [32] is formulated differently from the one we give here, but Impagliazzo [28] shows that the two are essentially equivalent.

For a function $\delta(n)$, two languages L and L', and an ensemble of distributions $\mathcal{D}$ on inputs, we say that L and L' are δ-close with respect to $\mathcal{D}$ if for sufficiently large n, the measure of the set $L_n \triangle L'_n$ according to $\mathcal{D}_n$ is at most $\delta(n)$. The definition of "heuristic polynomial time with success probability $s(n)$" requires that A and L be $(1 - s(n))$-close.

2.2. Worst-case to average-case reductions. A worst-case to average-case reduction is a procedure that transforms any average-case algorithm for one problem into an algorithm that works on all inputs for another problem. The reduction is called *nonadaptive* if the reduction decides on all its queries before it makes any of them. The following definition formalizes this notion.

DEFINITION 2.3. *A* nonadaptive worst-case to average-case randomized reduction *from L to $(L', \mathcal{D})$ with* average hardness δ *(in short, a δ* worst-to-average reduction*) is a family of polynomial size circuits $R = \{R_n\}$ such that on input $x \in \{0,1\}^n$ and randomness r, $R_n(x; r)$ outputs strings $y_1, \ldots, y_k$ (called* queries*) and a circuit C (called a* decider*) such that for any L^* that is δ-close to L' with respect to $\mathcal{D}$, it holds that*

$$\Pr_r[C(L^*(y_1), \ldots, L^*(y_k)) = L(x)] \geq 2/3.$$

We can also think of R as a nonadaptive oracle procedure that, when provided any L^* that is δ-close to L' as an oracle, agrees with L on every input (with probability at least $2/3$ over its internal coin tosses).

Notice that if there is a δ worst-to-average reduction from L to $(L', \mathcal{D})$, and $L \notin \text{BPP}$, then $(L', \mathcal{D}) \notin \text{Heur}_{1-\delta}\text{BPP}$. When the distribution $\mathcal{D}$ is uniform, we may denote the distributional problem $(L', \mathcal{D})$ just by L'.

Remarks on the definition.

(i) The choice of constant $2/3$ for the success probability of the reduction in the definition is rather arbitrary. If there exists a worst-to-average reduction R_n from L to $(L', \mathcal{D})$ that succeeds with probability $1/2 + n^{-c}$, there also exists one that succeeds with probability $1 - 2^{-n^{c'}}$ for arbitrary constants c, c'.

(ii) Without loss of generality, we may also assume that the strings $y_1, \ldots, y_k$ are identically distributed. Suppose R is an arbitrary reduction R that makes k queries. We define a new reduction R' that randomly permutes the queries of R. Then each query of R' is distributed as a random query of R.

(iii) Without loss of generality, we can fix two polynomials $k(n)$ and $m(n)$ such that for every n, when given an input x of length n, the reduction makes exactly $k(n)/m(n)$ queries of length i for every i between 1 and $m(n)$ (so that the total number of queries made is $k(n)$). This condition guarantees that for every i between 1 and $m(n)$, and every string y of length i, the probability that a random query made by the reduction equals y is exactly $1/m(n)$ times the probability that a random query made by the reduction equals y, with the condition that the length of this query is i. When working with distributions over queries, it is convenient to fix the query length; this restriction will allow us to relate statistics over queries of fixed length to statistics over queries of arbitrary length produced by the reduction.

(iv) We used a nonuniform definition of reductions, as it gives us a more general impossibility result. In particular, our result holds for uniform reductions.

2.3. Constant-round interactive protocols. We now discuss the types of protocols used in our proof, as well as certain extensions that will be used as building blocks.

Constant-round interactive protocols with advice. All the protocols in this paper are constant-round interactive protocols with polynomially long advice. An interactive protocol with advice consists of a pair of interactive machines (P, V), where P is a computationally unbounded prover and V is a randomized polynomial-time verifier which receive a common input x and advice string a. Feigenbaum and Fortnow [17] define the class $\mathrm{AM}^{\mathrm{poly}}$ as the class of languages L for which there exists a constant c, a polynomial p, and an interactive protocol (P, V) with advice such that for every n, there exists an advice string a of length $p(n)$ such that for every x of length n, on input x and advice a, (P, V) produces an output after c rounds of interaction and

(i) if $x \in L$, then $\Pr[(P, V) \text{ accepts } x \text{ with advice } a] \geq 2/3$;

(ii) if $x \notin L$, then for every prover P^*, $\Pr[(P^*, V) \text{ accepts } x \text{ with advice } a] \leq 1/3$.

We observe that this definition is weaker than the definition of the class AM/poly following the Karp–Lipton notion of classes with advice, which requires that the (P, V) be a valid constant-round interactive protocol (possibly for some language other than L) for all possible settings of the advice. (We note that in both cases, the advice is accessible to both the prover and the verifier.) Even though $\mathrm{AM}^{\mathrm{poly}}$ appears to be larger than AM/poly, they are in fact both equal to NP/poly (cf. [17]). Owing to this, in our description of protocols we will not be concerned with the behavior of the protocol when the advice is bad.

The protocol of Feigenbaum and Fortnow uses public coins. In contrast, our protocol will use private coins. In the case of constant-round interactive protocols without advice, it is known that private coin protocols can be simulated by public coin protocols [26]. The argument extends to protocols with advice, and therefore we may drop the public coin requirement in the definition of $\mathrm{AM}^{\mathrm{poly}}$.

The existence of $\mathrm{AM}^{\mathrm{poly}}$ protocols for all of coNP implies a partial collapse of the polynomial hierarchy, as was also observed in [17]: By the above observations, the assumption $\mathrm{coNP} \subseteq \mathrm{AM}^{\mathrm{poly}}$ implies $\mathrm{coNP} \subseteq \mathrm{NP/poly}$, and by a result of Yap [42], this gives $\Sigma_3 = \Pi_3$.

Protocols with shared auxiliary input. When applying two protocols in sequence, the second protocol has access to the transcript of the interaction from the first protocol. To allow access to this transcript, we extend our definition of interactive protocol to include a shared auxiliary input. This shared auxiliary input comes with a promise Υ, which may depend on the actual input. The completeness and soundness conditions are required to hold for all auxiliary inputs satisfying the promise. In the case of sequential composition of two protocols, the promise of the second protocol includes the set of all transcripts that are not rejecting for the first protocol. The running time of the verifier in a protocol with shared auxiliary input is measured with respect to the length of the concatenation of the actual input and the shared auxiliary input.

Protocols with private verifier input. In a protocol with private verifier input, the verifier is given, in addition to the input x, a private input r not known to the prover. The input r will be a "secret" of the verifier—a random string, uniformly distributed among a range of secrets that may depend on the input x. We represent the range of secrets by an NP relation H: A secret r for input x is chosen uniformly among all r such that $(x; r)$ satisfies H.

In our application, the protocol will be applied to instances of promise problems (see [15, 22]) instead of languages. For this reason, we state a definition of constant-round protocols with private verifier input for promise problems.

DEFINITION 2.4. *An* interactive protocol with private verifier input *consists of a polynomial-time verifier V, an unbounded prover P, and an* NP *relation H. A promise problem $\Pi = (\Pi_Y, \Pi_N)$ admits such a protocol with completeness c and soundness s if*

(i) *for all $x \in \Pi_Y$, $\Pr[(P, V(r))$ accepts $x] \geq c$.*

(ii) *for all $x \in \Pi_N$ and every prover P^*, $\Pr[(P^*, V(r))$ accepts $x] \leq s$.*

In both cases, the randomness is taken over V and over r chosen uniformly from all strings that satisfy $(x; r) \in H$.

Notice that this definition extends the standard notion of proof system without a private input, as we can specialize the definition to an NP relation H that mandates a unique choice of r for every x. The definition can be naturally extended in the case of protocols with shared auxiliary input. For such protocols to be sequentially composable, we must require that the private input of the verifier be independent of the shared auxiliary input, conditioned on the actual input.

Parallel repetition of two-round protocols. Suppose, given n instances $x_1, \ldots, x_n$ of a promise problem Π in $\mathrm{AM}^{\mathrm{poly}}$, we want a $\mathrm{AM}^{\mathrm{poly}}$ protocol that distinguishes between the case when all $x_i \in \Pi_Y$ and the case when at least one $x_i \in \Pi_N$. A natural approach is to run n independent instantiations of the protocol for Π in parallel, and accept if all of them accept. Intuitively, if the protocol for Π has completeness $1 - \epsilon$ and soundness δ, we expect the parallel protocol to have completeness $1 - n\epsilon$ and soundness δ. This worsens the completeness of the original protocol, while leaving the soundness essentially unchanged.

One way to improve the soundness is the following. Suppose that we could settle for distinguishing between the case when all $x_i \in \Pi_Y$ and the case when $x_i \in \Pi_N$ *for at least t of the x_is*. Intuitively, this relaxation makes the work required of a cheating prover much more demanding: Such a prover is now trying to convince the verifier to accept an instance in which at least t of the x_i's are "no" instances of Π. We expect such a prover to have success probability at most δ^t.

We prove that this is indeed the case for public coin two-round protocols (which is sufficient for our application), even for proof systems with private verifier input. We begin by describing the promise problem intended to be solved by parallel composition. We then define parallel composition for two-round protocols with private verifier input, and prove that parallel composition solves the intended problem.[4]

Given a promise problem $\Pi = (\Pi_Y, \Pi_N)$, we define the *n-wise repetition of* Π *with threshold t* to be the promise problem $\Pi^{n,t} = (\Pi_Y^{n,t}, \Pi_N^{n,t})$ as follows:

$$\Pi_Y^{n,t} = \{(x_1, \ldots, x_n) : x_i \in \Pi_Y \text{ for all } i\},$$
$$\Pi_N^{n,t} = \{(x_1, \ldots, x_n) : x_i \in \Pi_N \text{ for at least } t \text{ values of } i\}.$$

Suppose (P, V, H) is a k round protocol with private verifier input and with advice. We define its n-fold parallel composition as the k round protocol (P^n, V^n, H^n) with private verifier input, where

(i) V^n is the machine that, on input $(x_1, \ldots, x_n)$ and private verifier input $(r_1, \ldots, r_n)$, simulates n independent runs of V, where the ith run takes input x_i, private verifier input r_i, uses randomness independent of all other runs, and responds according to the next message function of V given the transcript of messages generated

[4] Goldreich [21, Appendix C.1] proves that parallel composition has the desired completeness and soundness errors for private coin protocols with arbitrary round complexity, but without private verifier input. His proof easily extends to our setting, but for simplicity we present a self-contained proof here.

by the ith run of (P, V) so far. At the end of the interaction, V^n accepts if the transcripts produced by *all* the runs are accepting.

(ii) P^n is the machine, that, on input $(x_1, \ldots, x_n)$, simulates n runs of P, where the ith run takes input x_i and responds according to the next message function of P given the transcript of messages generated by the ith run of (P, V) so far.

(iii) H^n is defined as follows: $((x_1, \ldots, x_n); (r_1, \ldots, r_n)) \in H^n$ iff $(x_i; r_i) \in H$ for all $1 \leq i \leq n$.

LEMMA 2.5. *Suppose (P, V, H) is a two-round protocol (where the first message is sent by the verifier) with private verifier input for promise problem Π with completeness $1 - \epsilon$ and soundness δ. Moreover, suppose that the message sent by V contains all of V's coin tosses. Then (P^n, V^n, H^n) is a protocol for $\Pi^{n,t}$ with completeness $1 - n\epsilon$ and soundness δ^t.*

Proof. The completeness of (P^n, V^n, H^n) follows by taking a union bound over the n runs of (P, V, H). To argue soundness, suppose that $\mathbf{x} = (x_1, \ldots, x_n) \in \Pi_N^{n,t}$. Without loss of generality, assume that specifically $x_1, \ldots, x_t \in \Pi_N$. For an input x and private verifier input r, define $B_{x,r}$ as the set of all messages μ sent by V on input (x, r) for which there exists a response ν that makes V accept.

Consider an arbitrary prover P^* that interacts with V^n. Suppose that V^n receives private verifier input $\mathbf{r} = (r_1, \ldots, r_n)$, and sends the message $\mu = (\mu_1, \ldots, \mu_n)$. In the second round, P^* responds with the message $\nu = (\nu_1, \ldots, \nu_n)$. Note that

$$\Pr_{\mathbf{r},\mu}[V^n(\mathbf{x}, \mathbf{r}, \mu, \nu) \text{ accepts}] \leq \Pr_{\mathbf{r},\mu}[\mu_i \in B_{x_i,r_i} \text{ for all } 1 \leq i \leq t].$$

The independence of the verifier strategies on different runs implies that for every i, the event $\mu_i \in B_{x_i,r_i}$ is independent of any function of μ_j, r_j for $j \neq i$. Therefore

$$\Pr_{\mathbf{r},\mu}[\mu_i \in B_{x_i,r_i} \text{ for all } 1 \leq i \leq t] = \prod_{i=1}^{t} \Pr_{r_i,\mu_i}[\mu_i \in B_{x_i,r_i}] \leq \delta^t$$

by the soundness of (P, V, H) for promise problem Π. ∎

The lemma and the proof also extend to protocols with shared auxiliary input (in addition to the private verifier input).

2.4. Lower and upper bound protocols. We now outline two protocols, used for proving approximate bounds on the number of accepting inputs of a circuit, that will be used as components in our constructions. The lower bound protocol of Goldwasser and Sipser [26] is used to prove an approximate lower bound on the number of accepting inputs of a circuit C. The upper bound protocol of Fortnow [19] (also used by Aiello and Håstad [2]) is used to prove an approximate upper bound on the same quantity, when the verifier is given private access to a random accepting input of the circuit. We stress that our version of the upper bound protocol is somewhat different from the protocols in [19, 2], as for our application we need a better approximation than in the original protocols, but we can allow for a much larger soundness error. The lower and upper bound protocols will allow the verifier to check whether queries made by the reduction are light or heavy.

We state the completeness and soundness conditions of the protocols, and provide proof sketches in section A.2 of the appendix.

The lower bound protocol. The lower bound protocol solves the following problem, which we denote $\Pi_{LB,\epsilon}$ (where $\epsilon > 0$):

Inputs. (C, s), where $C : \{0,1\}^m \to \{0,1\}$ is a circuit, $0 \le s \le 2^m$.

Shared auxiliary input. $\delta, \epsilon > 0$ (represented in unary), where δ is a parameter that controls the completeness and soundness errors, and ϵ controls the precision of the lower bound.

Yes instances. (C, s) such that $|C^{-1}(1)| \ge s$.

No instances. (C, s) such that $|C^{-1}(1)| \le (1-\epsilon)s$.

The protocol. On input (C, s) and shared auxiliary input (δ, ϵ):

1 Verifier: Set $k = \lceil 9/\delta\epsilon^2 \rceil$. Choose a pairwise independent hash function $h : \{0,1\}^m \to \Gamma$ at random, where $|\Gamma| = \lfloor s/k \rfloor$, and send h to the prover.

2 Prover: Send a list $r_1, \ldots, r_l \in \{0,1\}^m$, where $l \le (1+\epsilon/3)k$. An honest prover sends all r_i such that $C(r_i) = 1$ and $h(r_i) = 0$.

3 Verifier: If $C(r_i) \ne 1$ for any i, reject. If $|l - k| > \epsilon k/3$, reject. If $h(r_i) \ne 0$ for any i, reject. Otherwise, accept.

An alternative version of this protocol that achieves somewhat better parameters appears in work by Goldreich, Vadhan, and Wigderson [25]. The protocol presented here parallels the upper bound protocol, described below, more closely.

LEMMA 2.6. *The lower bound protocol is a protocol for $\Pi_{LB,\epsilon}$ with completeness $1-\delta$ and soundness δ.*

In our applications we will need to apply the lower bound protocol on many instances in parallel and to be guaranteed that all runs of the protocol are correct with high probability. For this reason, we resort to Lemma 2.5, observing that the lower bound protocol satisfies the hypothesis of this lemma. Applying Lemma 2.5 for $t = 1$ and setting $\delta = \epsilon/n$ yields a parallel lower bound protocol for $\Pi^{n,1}_{LB,\epsilon}$ with completeness $1-\epsilon$ and soundness ϵ.[5]

COROLLARY 2.7 (parallel lower bound protocol). *For every n and $\epsilon > 0$ there exists a constant round protocol for $\Pi^{n,1}_{LB,\epsilon}$ (with shared auxiliary input ϵ, represented in unary) with completeness $1-\epsilon$ and soundness ϵ.*

The upper bound protocol. The upper bound protocol solves the following problem, which we denote $\Pi_{UB,\epsilon}$ (where $\epsilon > 0$):

Inputs. (C, s), where $C : \{0,1\}^m \to \{0,1\}$ is a circuit, $0 \le s \le 2^m$.

Shared auxiliary input. $\delta, \epsilon > 0$ (represented in unary), where δ is a parameter that controls the completeness error and the soundness error. In contrast to the lower bound protocol, ϵ controls the precision of the protocol and also affects the soundness.

Yes instances. (C, s) such that $|C^{-1}(1)| \le s$.

No instances. (C, s) such that $|C^{-1}(1)| \ge (1+\epsilon)s$. The upper bound protocol is a protocol with private verifier input: The verifier is provided a random sample r of

[5] For this setting of parameters, ϵ controls both the precision of the approximate lower bound and the completeness and soundness errors. Had we wished to do so, we could have used independent parameters for the precision and for the completeness/soundness errors, though this is not necessary for our application. In contrast, in the parallel upper bound protocol presented below, the precision of the protocol and the soundness error are intricately related.

$C^{-1}(1)$, not known to the prover.

Private verifier input. A string $r \in S$. (Notice that the relation $\{((C,s);r) : C(r) = 1\}$ is an NP relation.)

The protocol. On input (C,s), shared auxiliary input (δ,ϵ), and private verifier input r:

1 Verifier: Set $k = \lceil 9/\delta\epsilon^2 \rceil$. Choose a 3-wise independent hash function $h : \{0,1\}^m \to \Gamma$ at random, where $|\Gamma| = \lfloor (s-1)/k \rfloor$ and send the pair $(h, h(r))$ to the prover.

2 Prover: Send a list $r_1, \ldots, r_l \in \{0,1\}^m$, where $l \leq (1+\epsilon/3)k$. An honest prover sends all r_i such that $C(r_i) = 1$ and $h(r_i) = h(r)$.

3 Verifier: If $C(r_i) \neq 1$ for any i, reject. If $l > (1+\epsilon/3)k$ or $r \notin \{r_1, \ldots, r_l\}$, reject. Otherwise, accept.

LEMMA 2.8. *The upper bound protocol is a protocol with private verifier input for $\Pi_{UB,\epsilon}$, completeness $1-\delta$, and soundness $1-\epsilon/6+\delta$.*

Setting $\delta = o(\epsilon)$, this yields a protocol with completeness $1-o(\epsilon)$ and soundness $1-\Omega(\epsilon)$, giving a narrow gap. To improve the soundness of the protocol, which is necessary for our application, we apply parallel repetition. In particular, fixing $\delta = o(\epsilon/n)$ and setting $t = \omega(1/\epsilon)$ in Lemma 2.5 yields a protocol for $\Pi^{n,t}_{UB,\epsilon}$ with completeness $1-o(1)$ and soundness $(1-\epsilon/6+o(\epsilon/n))^t = o(1)$. More generally, we have a parallel upper bound protocol for $\Pi^{n,t}_{UB,\epsilon}$ with the following parameters.

COROLLARY 2.9 (parallel upper bound protocol). *For every n and $\epsilon > 0$ there exists a constant round protocol with private verifier input (and shared auxiliary input ϵ, represented in unary) such that for every $t > 0$ the protocol decides $\Pi^{n,t}_{UB,\epsilon}$ with completeness $1-\epsilon$ and soundness $(1-\epsilon/9)^t$.*

3. The protocols. In this section we describe the constant-round interactive protocols that will constitute the building blocks of our main protocol. The order in which the protocols are presented follows the order in which they will be composed sequentially, which is the opposite of the description in the introduction. Recall that we need protocols that accomplish each of the following tasks:

(i) For a fixed input x, estimate the probability that a random query of a given length produced by the worst-to-average reduction on input x is light (that is, the probability of it being produced by the reduction is smaller than a specified threshold). We consider the following more abstract version of the problem: Given a circuit C, estimate the fraction of heavy outputs of C. The heavy samples protocol, described in section 3.1, solves this problem.

(ii) For a fixed input x, estimate the probability that a random query of a given length produced by the worst-to-average reduction on input x is both light and a "yes" instance. Abstractly, we can think of this problem as follows. We model the worst-to-average reduction as a sampler circuit C and the set of "yes" instances of a given length as a nondeterministic circuit V. As auxiliary input, we are given the fraction of accepting inputs for V as well as the probability that a random output of C is heavy. The task is to construct a protocol that estimates the probability that a random output of C is both light and accepting for V. The hiding protocol, described in section 3.2, accomplishes this task.

(iii) For a fixed input x, simulate an "approximate membership oracle" for queries made by the reduction on input x. This calls for a protocol for the fol-

lowing task: We are given a "querier" Q, describing an instantiation of the reduction on x, and an NP verifier V for "yes" instances. The following promise holds: When provided an oracle for the set

$$S_{Q,V} = \{y : V \text{ accepts } y \text{ and } y \text{ is a light query of } Q\},$$

Q either outputs "yes" with very high probability, or outputs "no" with very high probability (these cases correspond to the reduction saying "$x \in L$" and "$x \notin L$", respectively). The simulation protocol, described in section 3.3, distinguishes between these two cases when given as auxiliary inputs the fraction of heavy queries of Q and the fraction of "yes" instances of $S_{Q,V}$.

A note on approximations. The protocols described in this section cannot be expected to certify the exact values of the probabilities in question, but can only obtain approximations thereof. Intuitively, we will think of a protocol as computing an approximation of p if, for arbitrary ϵ, the protocol runs in time polynomial in $1/\epsilon$ and distinguishes between instances whose probability is p and instances whose probability is outside the interval $(p - \epsilon, p + \epsilon)$. Indeed, additive approximations are sufficient in all our applications.

These protocols also take as inputs (actual and auxiliary) probabilities of various events. We make the assumption that these probabilities are specified exactly, but in fact the completeness and soundness of the protocols are unaffected even if only approximations of these quantities were provided. The quality of approximation required is, in all cases, a fixed inverse polynomial function of the input length.

Statistics. We use the following formulation of the law of large numbers to obtain sampling estimates for various probabilities. For completeness we provide a proof in section A.1 of the appendix.

LEMMA 3.1 (sampling bound). *Let Ω be a sample space, $\epsilon, \eta < 1$, $T \subseteq \Omega$, and $\mathcal{D}$ a distribution on Ω. Suppose that S is a random sample of Ω consisting of at least $3\log(2/\eta)/\epsilon^3$ elements chosen independently at random from the distribution $\mathcal{D}$. Then with probability at least $1 - \eta$,*

$$\big||T \cap S|/|S| - \mathcal{D}(T)\big| \leq \epsilon.$$

High probability means probability $1 - o(1)$, where the $o(\cdot)$ notation is in terms of the length of the input of the protocol.

The notation $A \triangle B$ is for the symmetric difference of sets A and B. We use standard set facts about symmetric difference.

3.1. The heavy samples protocol. In this section we describe the protocol used for estimating the fraction of heavy samples generated by a circuit C. Recall that a string y is *α-heavy* for distribution $\mathcal{D}$ on $\{0,1\}^m$ if $\mathcal{D}(y) \geq \alpha 2^{-m}$. Given a distribution $\mathcal{D}$, the probability that a random sample of $\mathcal{D}$ is α-heavy is given by the quantity

$$h_{\mathcal{D},\alpha} = \Pr_{y \sim \mathcal{D}}[\mathcal{D}(y) \geq \alpha 2^{-m}].$$

The problem we are considering is the following: Given a circuit $C : \{0,1\}^n \to \{0,1\}^m$, a threshold α, a heaviness estimate p, and an error parameter ϵ, we want a protocol that accepts when $p = h_{\mathcal{D}_C,\alpha}$ and rejects when $|p - h_{\mathcal{D}_C,\alpha}| > \epsilon$. Here, as throughout the section, $\mathcal{D}_C$ denotes the output distribution of the circuit C.

A natural approach is to estimate the unknown $h_{\mathcal{D}_C,\alpha}$ by sampling: Suppose that for a random sample $y \sim \mathcal{D}_C$ we could decide with good probability if the sample was

α-heavy or α-light. Then, given $k = O(1/\epsilon^3)$ samples $y_1, \ldots, y_k$ generated by running C on independent inputs, we could estimate $h_{\mathcal{D}_C,\alpha}$ as the fraction of samples that are α-heavy, and sampling bounds would guarantee that the answer is correct with good probability.

However, in general we have no way of efficiently deciding whether a sample $y \sim \mathcal{D}_C$ is heavy or light. However, we have at our disposal the upper and lower bound protocols of section 2.4. For each sample y_i, the verifier first asks the prover to say if sample y_i is α-heavy or α-light. Then the prover is asked to prove the claims for the heavy samples using the lower bound protocol, and the claims for the light samples using the upper bound protocol. In both cases, the claims concern the size of the set $C^{-1}(y_i)$, which is the restriction of an NP set on $\{0,1\}^m$.

There are several difficulties with implementing this approach. First, the upper bound protocol requires the verifier to have a secret, random preimage r_i of the set $C^{-1}(y_i)$. Fortunately, the verifier obtains this secret for free in the course of generating y_i. When generating y_i, the verifier in fact chooses a random $r_i \in \{0,1\}^n$ and sets $y_i = C(r_i)$, so this r_i (which is kept hidden from the prover) is indeed a random preimage of y_i.

Another difficulty is that the upper bound protocol guarantees an ϵ deviation from the true value of $C^{-1}(y_i)$ only with probability $1 - O(\epsilon)$. Therefore the prover has a good chance of getting away with a false upper bound claim on any particular y_i. But how many times can the prover play this trick? If the prover decides to submit t false upper bound claims, its success probability quickly falls to $(1 - O(\epsilon))^t$ (see Corollary 2.7), so for $t = \omega(1/\epsilon)$ the risk of detecting a false upper bound claim becomes quite high for the prover. On the other hand, $O(1/\epsilon)$ false upper bound claims make no real difference for the verifier. In the end, the verifier uses these claims to compute its estimate of $h_{\mathcal{D}_C,\alpha}$, so any set of $O(1/\epsilon)$ false claims among k samples will change its estimate only by $O(k/\epsilon) = O(\epsilon^2)$, much less than the tolerated deviation ϵ.

The final difficulty stems from the fact that both the upper and lower bound protocols are approximate. To illustrate this issue, consider the case of a circuit C for which the distribution $\mathcal{D}_C$ is close to α-flat: Namely, every $y \in \{0,1\}^m$ has probability either 0 or $(1 \pm \epsilon)\alpha 2^{-m}$ under $\mathcal{D}_C$. Now for *every* sample y_i of the verifier, it happens that y_i is approximately both α-heavy and α-light. Hence the verifier has no soundness guarantee about the prover's claims for any y_i.

This issue appears quite difficult to resolve. We sidestep it by weakening the requirement on the protocol. Instead of requiring soundness for all α, we settle for soundness for a *random* choice of α. This avoids the "flatness" issue, because an almost flat distribution is very unlikely to be close to α-flat for a random α. More generally, given any distribution $\mathcal{D}_C$, if α is assigned a random value among any set of $1/\epsilon$ values that are spaced apart by a factor of at least $1 + 2\epsilon$, then the expected probability mass under $\mathcal{D}_C$ of samples that are both $(1-\epsilon)\alpha$-heavy and $(1+\epsilon)\alpha$-light can be at most ϵ. Therefore, the fraction of samples for which the verifier fails to obtain a soundness guarantee cannot be much more than $O(\epsilon)$.

Choosing a heaviness threshold. We formalize the last observation in the following claim. The claim is also used in sections 3.2 and 3.3. For every integer α_0 and fraction $\delta > 0$, define the distribution

$$\mathcal{A}_{\alpha_0,\delta} = \text{uniform distribution on } \{\alpha_0(1+3\delta)^i : 0 \le i \le 1/\delta\}. \tag{1}$$

Observe that every value in $\mathcal{A}_{\alpha_0,\delta}$ is in the range $[\alpha_0, e^3 \cdot \alpha_0]$. Since the intervals $((1-\delta)\alpha, (1+\delta)\alpha)$ are pairwise disjoint over the various $\alpha \in \mathcal{A}_{\alpha_0,\delta}$, the following

result holds.

CLAIM 3.2 (choosing a random threshold). *For every $\alpha_0 > 0$ and $0 < \delta < 1/3$, and every distribution $\mathcal{D}$ on $\{0,1\}^m$,*

$$\mathrm{E}_{\alpha \sim \mathcal{A}_{\alpha_0,\delta}}\left[\Pr_{y\sim\mathcal{D}}[\mathcal{D}(y) \in ((1-\delta)\alpha 2^{-m}, (1+\delta)\alpha 2^{-m})]\right] \leq \delta.$$

3.1.1. The protocol. We formalize the notion of a "protocol that works for a random heaviness threshold α" by defining a family of problems $\{\Pi_{HEAVY,\alpha}\}$, parametrized by the threshold α, and requiring that the protocol be complete and sound for a random problem in this family.

Inputs. (C, p, ϵ), where $C : \{0,1\}^n \to \{0,1\}^m$ is a circuit, and $p \in [0,1]$ is a probability, and $\epsilon > 0$ is an error parameter represented in unary.

Shared auxiliary input. (α, δ), where α is a threshold parameter represented in unary and $0 < \delta < 1/3$ is an error parameter.

Yes instances. (C, p, ϵ) such that $h_{\mathcal{D}_C,\alpha} = p$.

No instances. (C, p, ϵ) such that $|h_{\mathcal{D}_C,\alpha} - p| > \epsilon$.

The heavy samples protocol. On input (C, p, ϵ) and shared auxiliary inputs α and δ:

1. Verifier: Set $k = \lceil 3 \cdot 16^3 \log(2/\delta)/\delta^3 \rceil$. Choose $r_1, \ldots, r_k \sim \{0,1\}^n$. Compute $y_j = C(r_j)$. Send $y_1, \ldots, y_k$ to the prover.
2. Prover: Send a partition (H, L) of $[k]$. An honest prover sets $H = \{i : \mathcal{D}_C(y_i) \geq \alpha 2^{-m}\}$ and $L = \{i : \mathcal{D}_C(y_i) < \alpha 2^{-m}\}$.
3. Verifier and prover: Run the parallel upper bound protocol (see section 2.4) with auxiliary input $(r_i : i \in L)$ with shared auxiliary input (error parameter) δ for the claim "$|C^{-1}(y_i)| < \alpha 2^{n-m}$ for all $i \in L$."

   Run the parallel lower bound protocol (see section 2.4) with shared auxiliary input (error parameter) δ for the claim "$|C^{-1}(y_i)| \geq \alpha 2^{n-m}$ for all $i \in H$." Accept iff $\left||H|/k - p\right| \leq \epsilon/2$.

3.1.2. Analysis of the protocol. The following lemma states the completeness and soundness of the heavy samples protocol.

LEMMA 3.3. *For every integer α_0 and fractions ϵ, δ, with probability $1 - O(\delta/\epsilon)$ over α chosen uniformly from $\mathcal{A}_{\alpha_0,\delta}$, the heavy samples protocol (with input (C, p, ϵ) and auxiliary input (α, δ) satisfying the promise) is a protocol for $\Pi_{HEAVY,\alpha}$ with completeness $1 - O(\delta)$ and soundness $O(\delta)$.*

Proof. We denote by H' and L' the set of α-heavy and α-light samples, respectively,

$$H' = \{i : \mathcal{D}_C(y_i) \geq \alpha 2^{-m}\} \text{ and } L' = \{i : \mathcal{D}_C(y_i) < \alpha 2^{-m}\}.$$

The honest prover always chooses $H = H'$ and $L = L'$.

By the sampling bound, for every prover strategy, with probability $1 - O(\delta)$ over the randomness of the verifier, the fraction of α-heavy samples among $y_1, \ldots, y_k$ should closely approximate $h_{\mathcal{D}_C,\alpha}$, and in particular,

$$\left||H'|/k - h_{\mathcal{D}_C,\alpha}\right| \leq \epsilon/6. \tag{2}$$

Completeness. Completeness (for arbitrary α) follows from high probability estimate (2), together with completeness of the parallel lower bound protocol for $\Pi_{LB,\delta}^{|L|,1}$ (see Corollary 2.7) and completeness of the parallel upper bound protocol for $\Pi_{UB,\delta}^{|H|,\cdot}$ (see Corollary 2.9).

Soundness. Fix an $\alpha \sim \mathcal{A}_{\alpha_0,\delta}$ such that

$$\Pr_{y\sim\mathcal{D}_C}\big[\mathcal{D}_C(y) \in ((1-\delta)\alpha 2^{-m}, (1+\delta)\alpha 2^{-m})\big] \le \epsilon/16.$$

By Claim 3.2 and Markov's inequality, this holds with probability $1 - O(\delta/\epsilon)$ for a random α in $\mathcal{A}_{\alpha_0,\delta}$. For such a choice of α, let B denote the set of samples that are both $(1-\delta)\alpha$-heavy and $(1+\delta)\alpha$-light, that is,

$$B = \{i : \mathcal{D}_C(y_i) \in ((1-\delta)\alpha 2^{-m}, (1+\delta)\alpha 2^{-m})\}.$$

By the sampling bound, the number of samples in B is not much larger than $\epsilon/16$ with high probability over the randomness of the verifier. Indeed,

$$\Pr\big[|B| > \epsilon k/8\big] = \Pr\big[|B|/k - \epsilon/16 > \epsilon/16\big] \le \delta.$$

Now fix a prover strategy for which the verifier accepts instance (C, p, ϵ) with probability $\omega(\delta)$. Then there exists a setting of the verifier's randomness for which $\big||H|/k - p\big| \le \epsilon/2$ (by the last step of the verifier), $\big||H'|/k - h_{\mathcal{D}_C,\alpha}\big| \le \epsilon/6$ (by high probability estimate (2)), and the following conditions hold:

(i) For $t = \lceil \log(1/\delta)/\delta \rceil$ all but $t + \epsilon k/8$ samples in L are α-light, that is, $|L - L'| \le t + \epsilon k/8$. Indeed, this is an event of probability $1 - O(\delta)$ over the randomness of the verifier: By soundness of the parallel upper bound protocol for $\Pi_{UB,\delta}^{|L|,t}$, fewer than t of the samples in L are $(1+\delta)\alpha$-heavy. Moreover, the number of samples in L that are α-heavy but $(1+\delta)\alpha$-light is upper bounded by the size of B. It follows that with probability $1 - O(\delta)$, $|L - L'| \le t + \epsilon k/8$.

(ii) All but $\epsilon k/8$ samples in H are α-heavy, that is, $|H - H'| \le \epsilon k/8$. This is also an event of probability $1 - O(\delta)$ over the randomness of the verifier: By soundness of the parallel lower bound protocol for $\Pi_{LB,\delta}^{|H|,1}$, none of the samples in H are $(1-\delta)\alpha$-light. Moreover, the number of samples in H that are α-light but $(1-\delta)\alpha$-heavy is upper bounded by the size of B. It follows that with probability $1 - O(\delta)$, $|H - H'| \le \epsilon k/8$.

It follows that

$$\big||H| - |H'|\big| \le |H - H'| + |H' - H| = |H - H'| + |L - L'| \le t + \epsilon k/4 < \epsilon k/3.$$

Therefore,

$$|h_{\mathcal{D}_C,\alpha} - p| \le \big|h_{\mathcal{D}_C,\alpha} - |H'|/k\big| + \big||H'|/k - |H|/k\big| + \big||H|/k - p\big| < \epsilon/6 + \epsilon/3 + \epsilon/2 = \epsilon.$$

Thus, (C, p, ϵ) is a "yes" instance of $\Pi_{HEAVY,\alpha}$. □

3.2. The hiding protocol. In this section we describe the protocol for estimating the probability that a random sample generated by a circuit $C : \{0,1\}^n \to \{0,1\}^m$ is both a light sample and a "yes" instance of some NP language V. Let us denote by $\mathcal{D}_C$ the distribution of outputs of the circuit C and by $\mathcal{U}$ the uniform distribution on $\{0,1\}^m$. We assume that we are given as advice the probability p_Y that a random sample in $\{0,1\}^m$ is a "yes" instance of V.

For starters, let us assume that we are guaranteed that the distribution $\mathcal{D}_C$ is α-smooth (for some reasonably small α); that is, no output of C is α-heavy. Let

us consider the following experiment. We choose among distributions $\mathcal{D}_C$ and $\mathcal{U}$ by flipping a coin biased towards $\mathcal{D}_C$ with probability b (the value of interest is around $b = 1/\alpha$) and then generate a sample y according to either $\mathcal{D}_C$ or $\mathcal{U}$, depending on the outcome of the coin. We then give the sample y to the prover and ask which distribution it came from.

The best strategy for the prover to guess the origin of the sample is a maximum likelihood test: Compute the ratio $\mathcal{D}_C(y)/\mathcal{U}(y)$, and declare "$\mathcal{D}_C$" if the ratio exceeds $1/b$, and "$\mathcal{U}$" otherwise. However, when $b < 1/\alpha$, the maximum likelihood test will always guess that the sample came from $\mathcal{U}$. This effectively gives the verifier the ability to hide samples from $\mathcal{D}_C$ among samples from $\mathcal{U}$.

Suppose now that the verifier wants to determine membership in $V \cap \{0,1\}^m$ for a random sample from $\mathcal{D}_C$, but it has only the ability to determine membership for random samples from $\mathcal{U}$. The verifier can then use the prover as follows: Generate $\omega(\alpha)$ samples by choosing each one independently. The sample size is large enough so that at least one of the samples, call it z, will originate from $\mathcal{D}_C$. The prover is then asked to determine membership in V for all the samples. The verifier then checks if the prover determined membership in V correctly for all the samples coming from $\mathcal{U}$. If this is the case, then chances are that the prover also determined membership in V correctly for z, since this sample was hidden among all the other ones.

In our case, the verifier has no way of determining membership in V on samples from $\mathcal{U}$. Instead, it knows only that the fraction of samples from $\mathcal{U}$ that fall in V should be roughly p_Y. Recall that the verifier is interested in determining the fraction $g_{\mathcal{D}_C,V}$ of samples from $\mathcal{D}_C$ that fall in V. It is tempting to try the following protocol: The verifier and prover run the hiding procedure on a sufficiently large sample. The verifier then checks that the fraction of samples originating from $\mathcal{U}$ claimed to be in V by the prover is within a small enough deviation δ of p_Y. If this is the case, the verifier estimates $g_{\mathcal{D}_C,V}$ as the fraction of samples originating from $\mathcal{D}_C$ that are claimed to be in V by the prover.

It is not difficult to see that this protocol is not sound: A prover can "convince" the verifier, for instance, that $g_{\mathcal{D}_C,V} = p_Y$. To do so, for each sample the prover answers independently "yes" with probability p_Y and "no" with probability $1 - p_Y$ about the membership of this sample in V.

However, since V is an NP set, the verifier can impose an additional requirement: Every time the prover makes a "yes" claim for a sample, an NP certificate for membership in V of the sample must be provided. The prover's cheating power now becomes *one-sided* as it can no longer provide a "yes" answer for a sample that is not in V; the only way the prover can now cheat is by providing "no" answers for samples in V. However, if the prover supplies such false answers on more than an $O(\delta)$ fraction of the samples (where $\delta > 0$ is an error parameter), it is likely that most of these falsely answered samples originated from $\mathcal{U}$, because the samples originating from $\mathcal{U}$ comprise an overwhelming majority of all the samples. Therefore it is likely that the fraction of samples from $\mathcal{U}$ claimed "yes" by the prover is smaller than $p_Y - \delta$. On the other hand, statistically it is very unlikely that fewer than a $p_Y - \delta$ fraction of samples from $\mathcal{U}$ are in V. This prevents the prover from providing false answers on more than an $O(\delta)$ fraction of *all* the samples. Since the number of samples from $\mathcal{D}_C$ is roughly a b fraction of the total number of samples, the prover in particular cannot provide false answers on more than an $O(\delta/b)$ fraction of samples originating from $\mathcal{D}_C$. Therefore a cheating prover is unlikely to skew the verifier's estimate of $g_{\mathcal{D}_C,V}$ by more than $O(\delta/b)$. Setting $\delta = \epsilon b$ allows the verifier to obtain an additive ϵ approximation of $g_{\mathcal{D}_C,V}$ in time polynomial in $1/\epsilon$, α, and the sizes of C and V.

Handling the heavy samples. Notice that if we drop the smoothness restriction on $\mathcal{D}_C$, this argument fails because very heavy samples cannot be effectively hidden among uniform samples. However, our goal is to merely estimate the probability that a sample from $\mathcal{D}_C$ is both in V and light. To do so, we run the protocol as for smooth distributions, initially ignoring the heavy samples. The soundness of that protocol still shows that the prover must have answered most of the light samples from $\mathcal{D}_C$ correctly. What remains to be done is to weed out the heavy samples. By the protocol from section 3.1, the verifier can estimate the fraction p_H of heavy samples. (To avoid duplication, we assume the protocol here is given p_H as auxiliary input.) At this point, the verifier reveals its sample to the prover and asks the prover to give lower bound proofs for a $p_H - \epsilon$ fraction of samples from $\mathcal{D}_C$. The prover's cheating power here is also one-sided: By soundness of the parallel lower bound protocol, the prover cannot claim any of the light samples as heavy, so it can cheat only by claiming that some heavy samples are light. However, since the prover is required to provide lower bound proofs for a $p_H - \epsilon$ fraction of samples from $\mathcal{D}_C$, and the number of truly heavy samples from $\mathcal{D}_C$ is likely to be upper bounded by $p_H + \epsilon$, the prover cannot cheat on more than an $O(\epsilon)$ fraction of the samples from $\mathcal{D}_C$. Therefore a cheating prover cannot skew the verifier's estimate of $g_{\mathcal{D}_C,V}$ by more than an additive term of $O(\epsilon)$.

We encounter the same issue regarding the choice of α as in the protocol for heavy samples: If too many samples from $\mathcal{D}_C$ have probability about α, the lower bound protocol provides no soundness guarantee. As in section 3.1, we sidestep this problem by choosing a random α and arguing soundness with high probability over α.

3.2.1. The protocol. We give a protocol for the following family of promise problems, which we denote by $\{\Pi_{HIDE,\alpha}\}$.

Inputs. (C, V, p, ϵ), where $C : \{0,1\}^n \to \{0,1\}^m$ is a circuit, $V : \{0,1\}^m \times \{0,1\}^l \to \{0,1\}$ is a nondeterministic circuit,[6] $p \in [0,1]$ is a probability, and $\epsilon > 0$ is an error parameter represented in unary.

Shared auxiliary input. $(\alpha, \delta, p_Y, p_H)$, where $\alpha > 0$ is a threshold integer represented in unary, $0 < \delta < 1/3$ is an error parameter represented in unary, and p_Y, p_H satisfy

$$p_Y = \Pr_{y \sim \mathcal{U}}[y \in V] \quad \text{and} \quad |p_H - p'_H| < \epsilon/32, \tag{3}$$

where $p'_H = \Pr_{y \sim \mathcal{D}_C}[\mathcal{D}_C(y) \geq \alpha 2^{-m}]$.

Yes instances. (C, V, p, ϵ) such that $p = g_{\mathcal{D}_C,V,\alpha}$, where

$$g_{\mathcal{D}_C,V,\alpha} = \Pr_{y \sim \mathcal{D}_C}[\mathcal{D}_C(y) < \alpha 2^{-m} \text{ and } y \in V].$$

No instances. (C, V, p, ϵ) such that $|p - g_{\mathcal{D}_C,V,\alpha}| > \epsilon$.

---

[6] A *nondeterministic circuit* V computes a relation over $\{0,1\}^m \times \{0,1\}^l$, and we say $y \in \{0,1\}^m$ is accepted by V if there exists a $w \in \{0,1\}^l$ such that $V(y; w)$ accepts. Abusing notation, we also write V for the set of all y accepted by V.

The hiding protocol. On input (C, V, p, ϵ) and auxiliary input $(\alpha, \delta, p_Y, p_H)$:

1. Verifier: Set $b = \delta/\alpha$ and $k = \lceil 6 \cdot 32^3 \log(2/\delta)/(b\epsilon)^3 \rceil$. Choose a set $T_{\mathcal{D}_C} \subseteq [k]$ by assigning each element of $[k]$ to $T_{\mathcal{D}_C}$ independently with probability b. Let $T_{\mathcal{U}} = [k] - T_{\mathcal{D}_C}$.
   Choose strings $y_1, \ldots, y_k \in \{0,1\}^m$ as follows. If $j \in T_{\mathcal{D}_C}$, choose $y_j \sim \mathcal{D}_C$. If $j \in T_{\mathcal{U}}$, choose $y_j \sim \mathcal{U}$. Send the sequence $y_1, \ldots, y_k$ to the prover.
2. Prover: Send sets $Y, H \subseteq [k]$ and strings $(w_j : j \in Y)$ to the prover. An honest prover sends
   (a) Y as the set of all j such that $y_j \in V$,
   (b) H as the set of all j such that y_j is α-heavy for $\mathcal{D}_C$, and
   (c) w_j such that $V(y_j; w_j)$ accepts for all $j \in Y$.
3. Verifier: Reject under any of the following circumstances:
   (a) (Witness checks) $V(y_j; w_j)$ does not accept for some $j \in Y$.
   (b) (Frequency of "yes" samples in $\mathcal{U}$) $\big||Y \cap T_{\mathcal{U}}|/|T_{\mathcal{U}}| - p_Y\big| \geq \epsilon b/32$.
   (c) (Frequency of heavy samples in $\mathcal{D}_C$) $\big||H \cap T_{\mathcal{D}_C}|/|T_{\mathcal{D}_C}| - p_H\big| \geq \epsilon/16$.
   (With prover) Run the parallel lower bound protocol with shared auxiliary input (error parameter) δ for the claim

$$\text{"}|C^{-1}(y_j)| \geq \alpha 2^{n-m} \text{ for all } j \in H \cap T_{\mathcal{D}_C}\text{."}$$

   Accept iff

$$\big||Y \cap \overline{H} \cap T_{\mathcal{D}_C}|/|T_{\mathcal{D}_C}| - p\big| < \epsilon/4. \tag{4}$$

Remark. The parameter b needs to satisfy two constraints. It has to ensure that the α-light queries are hidden well (as a choice of probability for membership in T) and also has to guarantee that the fraction of "yes" samples in $T_{\mathcal{U}}$ is very close to p_U—not only within $O(\epsilon)$, but within $O(b\epsilon)$. This is necessary because $T_{\mathcal{D}_C}$ is smaller than $T_{\mathcal{U}}$ by a factor of b, so to obtain an $O(\epsilon)$ deviation bound for the fraction of light "yes" queries in $T_{\mathcal{D}_C}$, a stronger $O(b\epsilon)$ deviation bound must be assumed for the fraction of "yes" queries in $T_{\mathcal{U}}$.

3.2.2. Analysis of the protocol. The following lemma states the completeness and soundness of the hiding protocol.

LEMMA 3.4. *For every integer α_0 and fractions ϵ, δ, with probability $1 - O(\delta/\epsilon)$ over α chosen uniformly from $\mathcal{A}_{\alpha_0,\delta}$, the hiding protocol (with input (C, V, p, ϵ) and auxiliary input $(\alpha, \delta, p_Y, p_H)$ satisfying the promise) is a protocol for $\Pi_{HIDE,\alpha}$ with completeness $1 - O(\delta)$ and soundness $O(\delta)$.*

Proof. We denote by Y' and H' the set of actual "yes" samples and the set of actual heavy samples, respectively,

$$Y' = \{j : y_j \in V\} \text{ and } H' = \{j : \mathcal{D}_C(y_j) \geq \alpha 2^{-m}\}.$$

An honest prover always chooses $Y = Y'$ and $H = H'$.

The sampling bound implies that, for any fixed prover strategy, all of the following events hold with probability $1 - O(\delta)$ over the randomness of the verifier:

(i) The number of samples in $T_{\mathcal{D}_C}$ is large:

$$|T_{\mathcal{D}_C}| > bk/2 \geq 3 \cdot 32^3 \log(2/\delta)/\epsilon^3. \tag{5}$$

(ii) The number of samples in $T_{\mathcal{U}}$ is large:

$$|T_{\mathcal{U}}| > (1 - b - \epsilon)k. \tag{6}$$

(iii) About a p_Y fraction of samples in $T_{\mathcal{U}}$ are "yes" samples:

$$\big||Y' \cap T_{\mathcal{U}}|/|T_{\mathcal{U}}| - p_Y\big| < \epsilon b/32. \tag{7}$$

(iv) About a p_H fraction of samples in $T_{\mathcal{D}_C}$ are heavy samples:

$$\big||H' \cap T_{\mathcal{D}_C}|/|T_{\mathcal{D}_C}| - p'_H\big| < \epsilon/32. \tag{8}$$

(v) About a $g_{\mathcal{D}_C,V,\alpha}$ fraction of samples in $T_{\mathcal{D}_C}$ are light "yes" samples:

$$\big||Y' \cap \overline{H}' \cap T_{\mathcal{D}_C}|/|T_{\mathcal{D}_C}| - g_{\mathcal{D}_C,V,\alpha}\big| < \epsilon/4. \tag{9}$$

Completeness. Completeness (for arbitrary α) follows from high probability estimates (7), (8), and (9), promise (3), and completeness of the parallel lower bound protocol for $\Pi_{LB,\delta}^{|H\cap T_{\mathcal{D}_C}|,1}$.

Soundness. Fix an $\alpha \sim \mathcal{A}_{\alpha_0,\delta}$ such that

$$\Pr_{y\sim\mathcal{D}_C}\big[\mathcal{D}_C(y) \in ((1-\delta)\alpha 2^{-m}, (1+\delta)\alpha 2^{-m})\big] \le \epsilon/32.$$

By Claim 3.2 and Markov's inequality, this holds with probability $1 - O(\delta/\epsilon)$ for a random α in $\mathcal{A}_{\alpha_0,\delta}$. For such a choice of α, with probability $1 - O(\delta)$ over the randomness of the verifier, the number of samples in $T_{\mathcal{D}_C}$ that are both $(1-\delta)\alpha$-heavy and α-light is at most $\epsilon|T_{\mathcal{D}_C}|/16$. (This follows from the sampling bound and high probability estimate (5).)

Now fix a prover strategy for which the verifier accepts instance (C, V, p, ϵ) with probability $\omega(\delta)$. The analysis will be split into the following two parts:

(i) Show that the fraction of samples in $T_{\mathcal{D}_C}$ that were claimed both "yes" and "light" by the prover is within $O(\epsilon)$ of the fraction of truly light samples that were claimed "yes" by the prover. More generally, we show that for any set of samples $I \subseteq T_{\mathcal{D}_C}$, the fraction of samples in I that are claimed heavy by the prover is within $O(\epsilon)$ of the fraction of truly heavy samples in I. This will be shown in Claim 3.5.

(ii) Show that if the fraction of false "no" claims for samples in $T_{\mathcal{U}}$ is small, then the fraction of false "no" claims for light samples in $T_{\mathcal{D}_C}$ is small. This will be shown in Claim 3.6, which formalizes the hiding property of the protocol and contains the main idea of the soundness analysis.

Observe that step 3(a) of the hiding protocol ensures $Y \subseteq Y'$ whenever the verifier accepts. Let $Y^- = Y' - Y$.

CLAIM 3.5 (heavy samples). *With probability* $1 - O(\delta)$ *over the randomness of the verifier, if the verifier accepts, then for every set* $I \subseteq T_{\mathcal{D}_C}$, $\big||I \cap \overline{H}| - |I \cap \overline{H}'|\big| \le \epsilon|T_{\mathcal{D}_C}|/4$.

CLAIM 3.6 (hiding property). *With probability* $1 - O(\delta)$ *over the randomness of the verifier,* $|Y^- \cap T_{\mathcal{U}}| > \frac{1}{2}|Y^- \cap \overline{H}'|$.

We give the proofs of both these claims at the end of the section. Let us see first how these claims imply soundness. By a union bound, there exists an accepting transcript for the verifier, where high probability estimates (5), (7), and (9) hold, and the properties in Claims 3.5 and 3.6 hold. Fix such an accepting transcript.

By the accepting condition (4) and high probability estimate (9),

$$\begin{aligned}
|p - g_{\mathcal{D}_C,V,\alpha}| &< \frac{\epsilon}{4} + \frac{\epsilon}{4} + \left| \frac{|Y \cap \overline{H} \cap T_{\mathcal{D}_C}|}{|T_{\mathcal{D}_C}|} - \frac{|Y' \cap \overline{H}' \cap T_{\mathcal{D}_C}|}{|T_{\mathcal{D}_C}|} \right| \qquad \text{(by (4) and (9))} \\
&= \frac{\epsilon}{4} + \frac{\epsilon}{4} + \left| \frac{|Y \cap \overline{H} \cap T_{\mathcal{D}_C}|}{|T_{\mathcal{D}_C}|} - \left(\frac{|Y \cap \overline{H}' \cap T_{\mathcal{D}_C}|}{|T_{\mathcal{D}_C}|} + \frac{|Y^- \cap \overline{H}' \cap T_{\mathcal{D}_C}|}{|T_{\mathcal{D}_C}|} \right) \right| \\
&\leq \frac{\epsilon}{4} + \frac{\epsilon}{4} + \left| \frac{|Y \cap \overline{H} \cap T_{\mathcal{D}_C}|}{|T_{\mathcal{D}_C}|} - \frac{|Y \cap \overline{H}' \cap T_{\mathcal{D}_C}|}{|T_{\mathcal{D}_C}|} \right| + \frac{|Y^- \cap \overline{H}' \cap T_{\mathcal{D}_C}|}{|T_{\mathcal{D}_C}|} \\
&\leq \frac{\epsilon}{4} + \frac{\epsilon}{4} + \frac{\epsilon}{4} + \frac{|Y^- \cap \overline{H}' \cap T_{\mathcal{D}_C}|}{|T_{\mathcal{D}_C}|} \qquad \text{(by Claim 3.5).}
\end{aligned}$$

We now apply the hiding property to bound the last term. First,

$$\frac{|Y^- \cap T_{\mathcal{U}}|}{|T_{\mathcal{U}}|} = \frac{|Y' \cap T_{\mathcal{U}}|}{|T_{\mathcal{U}}|} - \frac{|Y \cap T_{\mathcal{U}}|}{|T_{\mathcal{U}}|} = \left(\frac{|Y' \cap T_{\mathcal{U}}|}{|T_{\mathcal{U}}|} - p_Y \right) + \left(p_Y - \frac{|Y \cap T_{\mathcal{U}}|}{|T_{\mathcal{U}}|} \right) < \frac{\epsilon b}{16}$$

by high probability estimate (7) and step 3(b) of the verifier. Now, using the hiding property and high probability estimate (5),

$$\frac{|Y^- \cap \overline{H}' \cap T_{\mathcal{D}_C}|}{|T_{\mathcal{D}_C}|} \leq \frac{|Y^- \cap \overline{H}'|}{|T_{\mathcal{D}_C}|} < 2 \cdot \frac{|Y^- \cap T_{\mathcal{U}}|}{|T_{\mathcal{D}_C}|} < 2 \cdot \frac{\epsilon bk/16}{bk/2} = \frac{\epsilon}{4}.$$

It follows that $|p - g_{\mathcal{D}_C,V,\alpha}| < \epsilon$, so (C, V, p, ϵ) is a yes instance of $\Pi_{HIDE,\alpha}$. □

Proof of Claim 3.5. For ease of notation, let $G = H \cap T_{\mathcal{D}_C}$ and $G' = H' \cap T_{\mathcal{D}_C}$. Denote by $\overline{G}$ and $\overline{G}'$ the complements of G and G' in $T_{\mathcal{D}_C}$, respectively.

Fix a transcript of the verifier for which high probability estimate (8) holds, none of the samples in $T_{\mathcal{D}_C} \cap H$ are $(1-\delta)\alpha$-light, and the number of samples whose weight is between $(1-\delta)\alpha$ and α in G is at most $\epsilon|T_{\mathcal{D}_C}|/16$. By soundness of the parallel lower bound protocol for $\Pi^{|G|,1}_{LB,\delta}$, Claim 3.2, and the sampling bound, all these events hold with probability $1 - O(\delta)$.

Suppose the verifier accepts. Since none of the samples in G are $(1-\delta)\alpha$-light and the number of samples whose weight is between $(1-\delta)\alpha$ and α in G is at most $\epsilon|T_{\mathcal{D}_C}|/16$, it follows that

$$|G - G'|/|T_{\mathcal{D}_C}| < \epsilon/16.$$

On the other hand, step 3(c) of the verifier, promise (3), and high probability estimate (8) give

$$\big||G| - |G'|\big|/|T_{\mathcal{D}_C}| \leq \big||G|/|T_{\mathcal{D}_C}| - p_H\big| + |p_H - p'_H| + \big|p'_H - |G'|/|T_{\mathcal{D}_C}|\big| < \epsilon/8.$$

The last two equations imply that the sets G and G' cannot differ on all but a few elements. Therefore, $\overline{G}$ and $\overline{G}'$ must also be very close, and so must be $I \cap \overline{G}$ and $I \cap \overline{G}'$.

The rest are calculations formalizing these claims. First, $|G' - G|$ must also be small because

$$\big||G'| - |G|\big| = \big||G' - G| - |G - G'|\big| \geq |G' - G| - |G - G'|$$

so that $|G' - G|/|T_{\mathcal{D}_C}| < 3\epsilon/16$. It follows that

$$\begin{aligned}\big||I \cap \overline{G}| - |I \cap \overline{G}'|\big| &\le \big||I \cap \overline{G}| - |I \cap \overline{G} \cap \overline{G}'|\big| + |I \cap (\overline{G}' - \overline{G})| \\ &= |I \cap (\overline{G} - \overline{G}')| + |I \cap (\overline{G}' - \overline{G})| \\ &\le |\overline{G} - \overline{G}'| + |\overline{G}' - \overline{G}| \\ &\le \epsilon |T_{\mathcal{D}_C}|/4. \qquad \square\end{aligned}$$

Proof of Claim 3.6. We will, in fact, reach the stronger conclusion

$$|Y^- \cap \overline{H}' \cap T_{\mathcal{U}}| > \frac{1}{2}|Y^- \cap \overline{H}'|.$$

For every sample $j \in Y^- \cap \overline{H}'$, that is, every light sample for which the prover made a false "no" claim, consider the event "$j \in T_{\mathcal{D}_C}$" from the point of view of the prover. For a fixed prover strategy, the first message of the verifier completely determines the set $Y^- \cap \overline{H}'$. First, we show that for any first message $\mathbf{y} = (y_1, \ldots, y_k)$ of the verifier, the probability of each event "$j \in T_{\mathcal{D}_C}$" for $j \in Y^- \cap \overline{H}'$ (over the randomness of the verifier's first message) is less than any constant:

$$\begin{aligned}\Pr[j \in T_{\mathcal{D}_C} \mid \mathbf{y}] &= \Pr[j \in T_{\mathcal{D}_C} | y_j] \qquad \text{(by independence of samples)} \\ &= \frac{\Pr[y_j \mid j \in T_{\mathcal{D}_C}] \Pr[j \in T_{\mathcal{D}_C}]}{\Pr[y_j]} \\ &\le \frac{\Pr[y_j \mid j \in T_{\mathcal{D}_C}] \Pr[j \in T_{\mathcal{D}_C}]}{\Pr[y_j \mid j \in T_{\mathcal{U}}] \Pr[j \in T_{\mathcal{U}}]} \\ &\le \frac{(\alpha 2^{-m}) \cdot b}{2^{-m} \cdot (1-b)} \qquad \text{(by lightness)} \\ &\le 2\delta \qquad \text{(by choice of } b\text{)}.\end{aligned}$$

For fixed $\mathbf{y}$, the quantity $|Y^- \cap \overline{H}' \cap T_{\mathcal{D}_C}|$ is a sum of indicator random variables for the events "$j \in T_{\mathcal{D}_C}$" (one for each $j \in Y^- \cap \overline{H}'$), so it follows that

$$\mathrm{E}\big[|Y^- \cap \overline{H}' \cap T_{\mathcal{D}_C}| \mid \mathbf{y}\big] = \textstyle\sum_{j \in Y^- \cap \overline{H}'} \Pr[j \in T_{\mathcal{D}_C} \mid \mathbf{y}] \le 2\delta \cdot |Y^- \cap \overline{H}'|.$$

by Markov's inequality, we have that

$$\Pr\left[|Y^- \cap \overline{H}' \cap T_{\mathcal{D}_C}| > \frac{1}{2}|Y^- \cap \overline{H}'| \mid \mathbf{y}\right] < 4\delta.$$

Therefore

$$\Pr\left[|Y^- \cap \overline{H}' \cap T_{\mathcal{U}}| \le \frac{1}{2}|Y^- \cap \overline{H}'| \mid \mathbf{y}\right] < 4\delta.$$

The claim follows by taking expectation over $\mathbf{y}$. $\square$

3.3. Simulating the reduction. In this section we describe the protocol that simulates a querier circuit Q (describing an instantiation of the worst-to-average reduction on a particular input) querying an average-case membership oracle for some NP set V. Let us assume that all the queries made by Q are identically distributed,

and denote by $\mathcal{D}_Q$ the distribution of a single query. The average-case membership oracle is for the set

$$S = \{y \in \{0,1\}^* : y \in V \text{ and } \mathcal{D}_Q(y) \leq \alpha 2^{-|y|}\}.$$

Recall that if Q describes an instantiation of a worst-to-average reduction from some language L to V, then distinguishing between the cases when Q accepts most of its inputs and when Q rejects most of its inputs allows us to determine membership in L.

We assume that the protocol is given as advice the probability p_H that a random query of Q is α-heavy, and the probability p_S that a random query of Q is in S. In reality, the protocol is only given approximations of these values, but for the sake of simplicity we ignore the distinction in this discussion. Suppose that Q on input $r \in \{0,1\}^n$ generates k queries $y_1, \ldots, y_k$ and a circuit $C : \{0,1\}^k \to \{0,1\}$. Moreover, Q satisfies the promise that $C(S(y_1), \ldots, S(y_k))$ either accepts or rejects with probability $1 - \eta$ for some $\eta > 0$.

Let us first consider the case when the distribution $\mathcal{D}_Q$ is α-smooth, that is, all queries are α-light. In this case, $S = V$ and $p_H = 0$, and the protocol of Feigenbaum and Fortnow can be used directly, as the advice p_S gives the probability that a random query generated by Q is a "yes" query, which is the advice needed for the Feigenbaum–Fortnow protocol. Let us recall how this protocol works. The verifier generates $l = \lceil 24 \cdot (k/\eta)^3 \log(2k/\delta) \rceil$ (where $\delta > 0$ is an error parameter) random strings $r_1, \ldots, r_l$, sends these strings to the prover, and asks the prover to simulate the computation of $Q(r_i)$ with oracle S for every i. To certify that most of the simulations are correct, the prover provides, with every "yes" query made in the simulations, an NP witness (for V) for this query. With high probability, for all query indices $j \in [k]$, among the jth queries made by Q, $(p_S \pm \eta/k)l$ of these queries must be "yes" instances of V, so the verifier can ask to see at least $p_S lk - \eta l$ "yes" answers without affecting completeness. But no prover can now make more than ηl false "no" claims, so if the verifier outputs the outcome of a random simulation, it will be correct with probability $1 - \eta$.

For a general distribution $\mathcal{D}_Q$, the difficulty is that the prover can no longer certify membership in S as in the Feigenbaum–Fortnow protocol, as S is not an NP set.[7] Instead of certifying membership in S directly, the verifier will first approximately determine which of its queries are α-heavy. To do so, the verifier uses the fact that heaviness is a certifiable property, thus limiting the cheating power of the prover: Statistically, the fraction of heavy queries is within $p_H \pm \eta/k$ with high probability, and the verifier asks the prover to give proofs of heaviness (using the lower bound protocol) for at least $p_H kl - \eta l$ of its queries. Since the prover's cheating power is one-sided (the prover is likely to be caught cheating if it claims that a light query is heavy), it can fool the verifier about heaviness on at most $2\eta l$ queries.

Once the verifier knows approximately which queries are α-heavy, it can ask the prover to reveal which queries are in V among the ones that are α-light: For each query that the verifier thinks is α-light, the prover is asked to determine membership in V and provide a certificate in case of a "yes" answer. Statistically, the fraction of queries that are light and in V is within $p_S \pm \eta/k$ with high probability, and the verifier asks to see "yes" certificates for at least $p_S kl - \eta l$ queries that it thinks are

[7] In fact, if S were defined as the set of y such that $y \in V$ *or* y is α-heavy, then it would have been an AM set (almost, save the fact that heaviness is an approximate AM property). This provides an alternate way of proving the main theorem: Modify the hiding protocol to calculate the fraction of samples that are either "yes" or heavy, then simulate the reduction using the Feigenbaum–Fortnow protocol.

α-light. If the set of queries that the verifier thinks are α-light coincided exactly with the set of truly α-light queries, the prover would not be able to provide more than $2\eta l$ false answers about membership in V among the α-light queries. In general these two sets will not coincide exactly. However, the number of elements on which they differ is at most $2\eta l$, so that the total number of truly α-light queries on which the prover can cheat about membership in V is still $O(\eta l)$.

It follows that for a random $i \in [l]$, the verifier can correctly simulate membership in S with probability $1 - O(\delta + \eta)$.

Regarding the choice of α, we encounter the same issue as in sections 3.1 and 3.2: If too many queries have probability about α, the lower bound protocol provides no soundness guarantee. Again, we sidestep this issue by arguing completeness and soundness for a random α.

3.3.1. The protocol. We give a protocol for the following family of promise problems, which we denote by $\{\Pi_{SIM,\alpha}\}$.

Inputs. (Q, V, η), where V is a nondeterministic polynomial-time machine, $Q : \{0,1\}^n \to \{0,1\}^{\mathrm{poly}(n)}$ is a querier circuit producing k queries for the set

$$S = S_{\mathcal{D}_Q,V,\alpha} = \{y \in \{0,1\}^* : V \text{ accepts } y \text{ and } \mathcal{D}_Q(y) < \alpha 2^{-|y|}\},$$

and $\eta > 0$ is an error parameter.

Shared auxiliary input. $(\alpha, \delta, p_H, p_S)$, where α is a threshold parameter represented in unary, $0 < \delta < 1/3$ is an error parameter represented in unary, and p_S, p_H satisfy

$$|p_H - p'_H| < \eta/2k \quad \text{and} \quad |p_S - p'_S| < \eta/2k, \tag{10}$$

where $p'_H = \Pr_{y\sim\mathcal{D}_Q}[\mathcal{D}_Q(y) \geq \alpha 2^{-|y|}]$ and $p'_S = \Pr_{y\sim\mathcal{D}_Q}[y \in S]$.

Yes instances. (Q, V, η) such that $\Pr_r[Q^S(r) \text{ accepts}] > 1 - \eta/2$.

No instances. (Q, V, η) such that $\Pr_r[Q^S(r) \text{ accepts}] < \eta/2$.

The simulation protocol. On input (Q, V), and auxiliary input $(\alpha, \delta, p_H, p_S)$:

1. Verifier: Set $l = \lceil 24 \cdot (k/\eta)^3 \log(2k/\delta) \rceil$. Choose l random strings $r_1, \ldots, r_l \in \{0,1\}^n$ and send them to the prover. Denote the jth query of $Q(r_i)$ by y_{ij}.
2. Prover: Send sets $Y, H \subseteq [l] \times [k]$ and strings $(w_{ij} : (i,j) \in Y)$ to the prover. An honest prover sends:
   (a) Y as the set of all (i,j) such that $y_{ij} \in V$,
   (b) H as the set of all (i,j) such that y_{ij} is α-heavy for $\mathcal{D}_Q$, and
   (c) w_{ij} such that $V(y_{ij}; w_{ij})$ accepts for all $(i,j) \in Y$.
3. Verifier: Reject under any of the following circumstances:
   (a) (Witness checks) $V(y_{ij}; w_{ij})$ does not accept for some $(i,j) \in Y$.
   (b) (Frequency of heavy samples) $\left||H|/kl - p_H\right| > \eta/k$.
   (c) (Frequency of light "yes" samples) $\left||Y \cap \overline{H}|/kl - p_S\right| > \eta/k$.

   (With prover) Run the parallel lower bound protocol with shared auxiliary input (error parameter) δ for the claim

   $$|\{r : \text{The first query of } Q(r) \text{ is } y_{ij}\}| > \alpha 2^{n-|y_{ij}|} \text{ for all } (i,j) \in H.$$

   Choose a random $i \in [l]$. Accept if the decider of $Q(r_i)$ accepts the input $(a_1, \ldots, a_k)$, where $a_j =$ "yes" if $(i,j) \in Y \cap \overline{H}$, and $a_j =$ "no" otherwise.

3.3.2. Analysis of the protocol. The following lemma states the completeness and soundness of the simulation protocol.

LEMMA 3.7. *For every integer α_0, querier circuit Q that produces k queries, and fractions η, δ, with probability $1 - O(\delta k/\eta)$ over α chosen uniformly from $\mathcal{A}_{\alpha_0,\delta}$, the simulation protocol (with input (Q, V, η) and auxiliary input $(\alpha, \delta, p_H, p_S)$ satisfying the promise) is a protocol for $\Pi_{SIM,\alpha}$ with completeness $1 - O(\delta + \eta)$ and soundness $O(\delta + \eta)$.*

Proof. We denote by H' and Y' the set of actual heavy samples, and the set of actual "yes" samples, respectively,

$$H' = \{(i,j) : \mathcal{D}_Q(y_{ij}) \geq \alpha 2^{-|y_{ij}|}\} \text{ and } Y' = \{(i,j) : y_{ij} \in V\}.$$

The honest prover always chooses $H = H'$ and $Y = Y'$.

First, we observe the following high probability estimates over the randomness of the verifier (for any prover strategy), which follow directly from the sampling bound:

$$(Q,V,\eta) \text{ is a yes instance} \implies \Pr\big[|\{i : Q^S(r_i) \text{ accepts}\}| > (1-\eta)l\big] = 1 - O(\delta). \tag{11}$$

$$(Q,V,\eta) \text{ is a no instance} \implies \Pr\big[|\{i : Q^S(r_i) \text{ accepts}\}| < \eta l\big] = 1 - O(\delta). \tag{12}$$

Let T denote an arbitrary fixed (that is, independent of both the verifier's randomness and the prover's strategy) subset of $\{0,1\}^*$. The key observation of Feigenbaum and Fortnow is that for any T, with probability $1 - O(\delta)$ over the randomness of the verifier, the fraction of queries y_{ij} that fall inside T is $|T|/kl \pm \eta/k$, even though there are dependencies among the queries. To see this, divide the queries into k sets, where the jth set consists of queries $q_{1j}, \ldots, q_{lj}$. Within each set, the queries are independent, so by the choice of $l = \lceil 24 \cdot (k/\eta)^3 \log(2k/\delta) \rceil$ and by the sampling bound, the fraction of queries in the jth set that fall inside T is $|T|/kl \pm \eta/k$ with probability $1 - O(\delta/k)$. By a union bound over j, it follows that with probability $1 - O(\delta)$ the total fraction of queries y_{ij} that fall inside T is within η/k of $|T|/kl$.

Specifically, in the case when T is the set of α-heavy queries, we obtain that with probability $1 - O(\delta)$ over the randomness of the verifier, it holds that

$$\big||H'|/kl - p_H\big| < \eta/2k. \tag{13}$$

When T is the set of α-light queries that are in V, again with probability $1 - O(\delta)$ over the randomness of the verifier, it holds that

$$\big||Y' \cap \overline{H}'|/kl - p_S\big| < \eta/2k. \tag{14}$$

Completeness. Completeness (for arbitrary α) follows from high probability estimates (11), (13), (14), promise (10), and completeness of the parallel lower bound protocol for the promise $\Pi^{|H|,1}_{LB,\delta}$.

Soundness. Fix an $\alpha \sim \mathcal{A}_{\alpha_0,\delta}$ such that

$$\Pr_{y \sim \mathcal{D}_Q}\big[\mathcal{D}_Q(y) \in ((1-\delta)\alpha 2^{-|y|}, (1+\delta)\alpha 2^{-|y|})\big] \leq \eta/2k.$$

By Claim 3.2 and Markov's inequality, this holds with probability $1 - 2\delta k/\eta$ for a random α in $\mathcal{A}_{\alpha_0,\delta}$. For such a choice of α, it follows from the sampling bound that

with probability $1 - O(\delta)$ over the randomness of the verifier, the number of queries that are both $(1-\delta)\alpha$-heavy and α-light is at most ηl.

Fix a prover strategy for which the verifier accepts instance (Q, V, η) with probability $\omega(\delta) + 11\eta$. We will show that at least an 11η fraction of the transcripts are accepting, satisfy high probability estimate (12), and have the property that the prover is honest on all but at most $10\eta l$ answers provided in step 2 of the protocol: Namely, they satisfy the condition

$$|(Y \cap \overline{H}) \triangle (Y' \cap \overline{H}')| < 10\eta l. \tag{15}$$

Now consider all prefixes of transcripts consisting of the prover-verifier interaction before the verifier's choice of index i in step 3. There must exist at least one such prefix that satisfies estimate (12) and condition (15), and for which at least an 11η fraction of choices for i yield accepting transcripts. For this prefix, condition (15) implies that for at least an $1-10\eta$ fraction of indices i, the verifier correctly simulates the computation $Q^S(r_i)$ using the claims received from the prover in step 2. Therefore, for at least an η fraction of indices i, the transcript resulting from this choice of i is accepting. By condition (12), it follows that (Q, V, η) must be a "yes" instance.

We now show that at least an 11η fraction of transcripts are accepting, satisfy high probability estimate (12), and satisfy condition (15). For an accepting transcript, step 3(a) of the verifier guarantees that $Y \subseteq Y'$. Let $Y^- = Y' - Y$. If $\overline{H}$ were equal to $\overline{H}'$ with high probability, we would have $(Y \cap \overline{H}) \triangle (Y' \cap \overline{H}') = Y^- \cap \overline{H}'$, so the claim would follow from the verifier's step 3(c) and estimates (12) and (14). The only complication is that $\overline{H}$ and $\overline{H}'$ are not equal (and the difference between them can be two-sided), but the set difference is small: $|H \triangle H'| \le 4\eta l$ for a $1 - O(\delta)$ fraction of transcripts because with probability $1 - O(\delta)$, both of the following properties hold:

(i) By soundness of the parallel lower bound protocol for $\Pi_{LB,\delta}^{|H|,1}$, none of the samples in H are $(1-\delta)\alpha$-light. Also, the number of samples whose weight is between $(1-\delta)\alpha$ and α is at most ηl, so it follows that $|H - H'| \le \eta l$.

(ii) Step 3(b) of the verifier, promise (10), and high probability estimate (13) give

$$\big||H| - |H'|\big| \le \big||H| - p_H kl\big| + |p_H - p'_H|kl + \big|p'_H kl - |H'|\big| < 2\eta l.$$

Then $|H' - H| \le 3\eta l$ because

$$\big||H'| - |H|\big| = \big||H' - H| - |H - H'|\big| \ge |H' - H| - |H - H'|.$$

It follows that

$$|H \triangle H'| = |H - H'| + |H' - H| \le 4\eta l. \tag{16}$$

By a union bound, at least an 11η fraction of transcripts are accepting and satisfy high probability estimates (12), (14) and condition (16). For such transcripts, we have

$$\begin{aligned}
\big|(Y \cap \overline{H}) \triangle (Y' \cap \overline{H}')\big| &= \big|(Y \cap \overline{H}) \triangle ((Y \cap \overline{H}') \triangle (Y^- \cap \overline{H}'))\big| \\
&= \big|(Y \cap (\overline{H} \triangle \overline{H}')) \triangle (Y^- \cap \overline{H}')\big| \\
&\le \big|\overline{H} \triangle \overline{H}'\big| + |Y^- \cap \overline{H}'| \\
&\le 4\eta l + |Y^- \cap \overline{H}'|.
\end{aligned}$$

The last line follows from the fact that the symmetric difference stays the same if the sets are complemented. Therefore,

$$\begin{aligned}
|Y^- \cap \overline{H}'| &\le \big||Y^- \cap \overline{H}'| + (|Y \cap \overline{H}'| - |Y \cap \overline{H}|)\big| + \big||Y \cap \overline{H}'| - |Y \cap \overline{H}|\big| \\
&\le \big||Y' \cap \overline{H}'| - |Y \cap \overline{H}|\big| + \big|(Y \cap \overline{H}') \vartriangle (Y \cap \overline{H})\big| \\
&= \big||Y' \cap \overline{H}'| - |Y \cap \overline{H}|\big| + \big|Y \cap (\overline{H}' \vartriangle \overline{H})\big| \\
&\le \big(\big||Y' \cap \overline{H}'| - p'_S kl\big| + |p'_S - p_S|kl + \big|p_S kl - |Y \cap \overline{H}|\big|\big) + |H \vartriangle H'| \\
&< (\eta l/2 + \eta l/2 + \eta l) + 4\eta l \quad \text{(by (14), (10), verifier step 3(c), and (16))}
\end{aligned}$$

so that $|(Y \cap \overline{H}) \vartriangle (Y' \cap \overline{H}')| < 4\eta l + 6\eta l = 10\eta l$. ∎

4. Main theorem and proof. In this section we state and prove the main theorem.

THEOREM 4.1 (main theorem). *For any two languages L and L' such that $L' \in$ NP and every constant c, if there is an n^{-c} nonadaptive worst-to-average reduction from L to L', then $L \in \mathrm{NP/poly} \cap \mathrm{coNP/poly}$.*

In particular, if L were hard for NP, then coNP $\subseteq$ NP/poly, and therefore $\Sigma_3 = \Pi_3$.

Proof. Fix an arbitrarily small constant $\eta > 0$. We will assume that there exist polynomials $k(n)$ and $m(n)$ such that for every n and every input x of length n, the reduction makes exactly $k(n)/m(n)$ queries of every length between 1 and $m(n)$, so that the total number of queries made by the reduction is $k(n)$. We will also assume that the queries made by the reduction are identically distributed, and that (when provided access to an average-case oracle) the reduction either accepts or rejects with probability at least $1 - \eta/2$. In section 2.2 we explained why all these assumptions can be made without loss of generality.

Observe that if there is an n^{-c} worst-to-average reduction from L to L', then there is also a worst-to-average reduction from $\overline{L}$ to L', so it suffices to prove that $L \in$ NP/poly. We describe an AM protocol for L with advice (in which completeness and soundness hold only when the advice is correct) and private coins with completeness $1 - O(\eta)$ and soundness $O(\eta)$. By the remarks in section 2.3, the existence of such a protocol for L shows that $L \in$ NP/poly.

Let R_n denote the circuit computing the worst-to-average reduction from L to L' on inputs of length n. Let V be a nondeterministic machine for L'. Set $\alpha_0 = n^{-c}$.

Input. A string x of length n.

Advice. For every $1 \le i \le m(n)$, the probability $p_{Y,i} = \Pr_{y \sim \{0,1\}^i}[y \in L']$, and the circuit R_n.

Let Q be the circuit obtained by hardwiring the input x to R_n. Thus, Q takes as input a random string r and produces as output $k(n)$ queries and a decider circuit. For every $1 \le i \le m(n)$, let C_i denote the circuit that generates a random query of Q of length i: The circuit C_i simulates the circuit Q, then uses additional randomness to select uniformly one of the $m(n)$ outputs of Q of length i. Let $\mathcal{D}_{C_i}$ be the distribution of a sample of C_i, and $\mathcal{D}_Q$ be the distribution of the first query of Q. Finally, let V_i be the nondeterministic circuit describing the computation of $M_{L'}$ on an input of length i.

The protocol.

1. Verifier: Set the error parameters δ, ϵ_1 and ϵ_2 so that $\delta = \min(\epsilon_1/m(n), 1/3)$, $\epsilon_1 = \epsilon_2/32$, and $\epsilon_2 = \eta/2k(n)$. Choose a random α from the distribution $\mathcal{A}_{\alpha_0,\delta}$ (see (1)). Send α to the prover.
2. Prover: For every $1 \le i \le m(n)$, send two probabilities $h_{\mathcal{D}_{C_i},\alpha}$ and $g_{\mathcal{D}_{C_i},V,\alpha}$ to the verifier. An honest prover sends
$$h_{\mathcal{D}_{C_i},\alpha} = \Pr_{y\sim\mathcal{D}_{C_i}}[\mathcal{D}_{C_i}(y) \ge \alpha 2^{-i}] \text{ and}$$
$$g_{\mathcal{D}_{C_i},V,\alpha} = \Pr_{y\sim\mathcal{D}_{C_i}}[\mathcal{D}_{C_i}(y) < \alpha 2^{-i} \text{ and } y \in V].$$
3. Verifier and prover: For every $1 \le i \le m(n)$, run (in parallel) the heavy samples protocol (see section 3.1) on input $(C_i, h_{\mathcal{D}_{C_i},\alpha}, \epsilon_1)$ and shared auxiliary input (α, δ).
For every $1 \le i \le m(n)$, run (in parallel) the hiding protocol (see section 3.2) on input $(C_i, V_i, g_{\mathcal{D}_{C_i},V_i,\alpha}, \epsilon_2)$ and shared auxiliary input $(\alpha, \delta, p_{Y,i}, h_{\mathcal{D}_{C_i},\alpha})$.
4. Verifier: Let
$$p_H = \sum\nolimits_{i=1}^{m(n)} \frac{1}{m(n)} \cdot h_{\mathcal{D}_{C_i},\alpha} \text{ and } p_S = \sum\nolimits_{i=1}^{m(n)} \frac{1}{m(n)} \cdot g_{\mathcal{D}_{C_i},V,\alpha}.$$
5. Verifier and prover: Run the simulation protocol (see section 3.3) on input $(Q, M_{L'}, \eta)$ and shared auxiliary input $(\alpha, \delta, p_H, p_S)$.

Observe that when the prover is honest, the probability p_H is exactly the probability that a random query of Q is α-heavy regardless of its length. Conversely, if p_H deviates from the probability that a random query of Q is α-heavy by more than ϵ, it must be that at least one of the claims $h_{\mathcal{D}_{C_i},\alpha}$ deviates from the value $\Pr_{y\sim\mathcal{D}_{C_i}}[\mathcal{D}_{C_i}(y) \ge \alpha 2^{-i}]$. Similar considerations apply to the probability p_Y.

Analysis of the protocol. The protocol intends to simulate a run of the reduction R when given oracle access to the set L^*_α for some α, where

$$L^*_\alpha = \{y \in \{0,1\}^* : y \in L' \text{ and } \mathcal{D}_Q(y) < \alpha 2^{-|y|}\}.$$

Observe that for every $\alpha \in \mathcal{A}_{\alpha_0,\delta}$, the languages L' and L^*_α are $1/\alpha_0$-close: The distance between L' and L^*_α equals the measure of the set of α-heavy samples for $\mathcal{D}_Q$ under the uniform distribution. Since the number of α-heavy samples of $\mathcal{D}_Q$ of length i cannot exceed $\alpha^{-1} \cdot 2^{-i}$, the two sets are $1/\alpha \le 1/\alpha_0$-close under the uniform distribution.

Observe that our choice of parameters guarantees that for a random choice of $\alpha \sim \mathcal{A}_{\alpha_0,\delta}$, with probability $1 - O(\eta)$ over the choice of α, all runs of the heavy samples protocol, the hiding protocol, and the query simulation protocol satisfy the completeness and soundness conditions guaranteed by Lemmas 3.3, 3.4, and 3.7. For such a choice of α, completeness and soundness of the protocol follow by inspection. The completeness error is $O(\eta)$, which we obtain by adding the completeness errors of all the component protocols. The soundness error is also $O(\eta)$, which we obtain by observing that parallel composition does not increase the soundness error (see section 2.3) and by adding the soundness errors from steps 3 and 5 of the protocol. □

Remarks.

(i) If the worst-to-average reduction were uniform, the proof of Theorem 4.1 actually gives the stronger conclusion $L \in \mathrm{AM}^{\log}$. This requires small modification to the protocol: Instead of requiring that the protocol be given as advice the values $p_{Y,i}$ for all i between 1 and $m(n)$, we ask only that the advice consist of the average $p_Y = \sum_{i=1}^{m(n)} p_{Y,i}/m(n)$, which can be represented using $O(\log n)$ bits. As a preliminary step of the modified protocol, the prover sends claims for the actual values $p_{Y,i}$, and the verifier checks that p_Y is the average of these values. To check that these claims are correct (within an arbitrarily small additive term ϵ), for each i, the verifier generates $\omega(\log(m(n))/\epsilon^3)$ uniformly random samples of length i and asks the prover to provide certificates for membership in V for at least a $p_{Y,i} - \epsilon$ fraction of them. An honest prover can provide sufficiently many certificates with high probability, and the power of the cheating prover is one-sided: Such a prover cannot understate any $p_{Y,i}$ by more than 2ϵ, so to preserve the average p_Y it cannot overstate any $p_{Y,i}$ by more than $2\epsilon m(n)$. Choosing ϵ small enough provides the verifier with sufficiently good approximations of the values $p_{Y,i}$.

(ii) The condition $L' \in \mathrm{NP}$ can be weakened to $L' \in \mathrm{NP/poly}$, as the advice for the verifier of L' can be incorporated as advice to the protocol.

(iii) The conclusion of the theorem holds even under Levin's notion of hardness for efficient-on-average algorithms. This notion makes the additional requirement that L^* be an "errorless" approximation of L in the proof of Theorem 4.1; that is, L^* now takes values in $\{0, 1, \text{"fail"}\}$ and it is required that if $L^*(x) \neq L'(x)$, then $L^*(x) =$ "fail." Accommodating this change requires merely a slight modification of the simulation protocol: Instead of simulating answers to heavy queries by "no," the modified protocol simulates answers to heavy queries by "fail."

5. Search problems and samplable distributions. In this section, we generalize our results to reductions from worst-case hard languages to average-case search problems (instead of languages) whose average-case complexity is measured with respect to arbitrary samplable distributions (instead of the uniform distribution).

Observe that if a decision problem L in NP is hard on average with respect to some distribution $\mathcal{D}$, then the search version of L is also hard with respect to $\mathcal{D}$. However, the converse is not evident. Thus, even though Theorem 4.1 shows that nonadaptive worst-case to average-case reductions from an NP-hard problem to decision problems in NP are unlikely to exist, it is conceivable that reductions to search problems in NP are possible. In this section we rule out this possibility, showing that reductions to arbitrary search problems in distributional NP are no more powerful than reductions to decision problems in NP with respect to the uniform distribution.

The idea of the proof is to show that if there exists a worst-to-average reduction from some language L to a search problem in distributional NP, then this reduction can be composed with known reductions from the average-case complexity of Impagliazzo and Levin [29] and Ben-David et al. [7] to obtain a worst-to-average reduction from L to some language L' with respect to the uniform distribution, thus reducing this to the special case studied in Theorem 4.1. A crucial property for our purpose of the reductions in [7, 29], which is implicit in those works, is that both reductions are nonadaptive.

We begin by defining the type of reduction under consideration, as well as the types of reductions implicit in [7, 29] that will be used in the proof of the main theorem of this section.

5.1. Average-case reductions for search problems. The notion of a "worst-case to average-case reduction" can be generalized in several ways. Such generalizations are needed in order to extend our impossibility result for worst-case to average-case reductions to the case when the average-case problem is a distributional search problem. To obtain this result, we will need to compose worst-to-average reductions with *average-case reductions*.

We begin with the notion of a heuristic NP search algorithm that not only works well on average, but also provides witnesses for "yes" instances.

Let V be an NP relation. We denote by L_V the NP language corresponding to V; i.e., $L_V(x) = 1$ iff there exists a w such that $V(x; w) = 1$. A family of random functions $F_n : \{0,1\}^n \to \{0,1\}^m$ is a *δ-approximate witness oracle* for V with respect to the ensemble of distributions $\mathcal{D}$ if for all n,[8]

$$\Pr_{x\sim\mathcal{D},F}[V(x; F_{|x|}(x)) = L_V(x)] > 1 - \delta.$$

We will omit the subscript of F when it is implicitly determined by the input length. Note that the definition implies the existence of a set S of measure $\mathcal{D}(S) = 1 - 3\delta$ such that for all $x \in S$,

$$\Pr_F[V(x; F_{|x|}(x)) = L_V(x)] > 2/3.$$

Intuitively, S is the set of inputs for which the oracle has a good chance of producing a witness for the input, when such a witness exists. As usual, the constant $2/3$ is arbitrary, since if one has access to F, it can be queried k times independently in parallel to obtain a good witness with probability $1 - 1/3^k$.

Just as languages in NP represent decision problems, witness oracles represent search problems. For example, inverting a one-way function $f : \{0,1\}^n \to \{0,1\}^n$ on a $1-\delta$ fraction of inputs amounts to finding an algorithm $A : \{0,1\}^n \to \{0,1\}^n$ that is δ-approximate for the relation $V(y; x) \iff y = f(x)$ with respect to the distribution $f(\mathcal{U}_n)$.

Using witness oracles, we can formalize the notion of nonadaptive reductions between search problems, as well as reductions from search to decision problems. Let us focus on the case of a reduction between two search problems $(V, \mathcal{D})$ and $(V', \mathcal{D}')$. As in the case of languages, we want the property that the reduction transforms any heuristic algorithm for $(V', \mathcal{D}')$ into a heuristic algorithm for $(V, \mathcal{D})$. Moreover, if $x \in L_V$, we want the reduction, on most inputs $x \sim \mathcal{D}$, to recover a witness for x based on the answers provided by the witness oracle for V'.

DEFINITION 5.1 (reduction between search problems). *Let V, V' be* NP *relations and $\mathcal{D}, \mathcal{D}'$ be polynomial-time samplable distribution ensembles. A* δ-to-δ' search-to-search reduction *for search problems from $(V, \mathcal{D})$ to $(V', \mathcal{D}')$ is a family of polynomial size circuits $R = \{R_n\}$ such that on input $x \in \{0,1\}^n$, randomness r, $R_n(x; r)$ outputs strings $y_1, \ldots, y_k$ and a circuit C such that for any witness oracle F^* that is δ'-approximate for V' with respect to $\mathcal{D}'$, it holds that*

$$V(x, C(F^*(y_1), \ldots, F^*(y_k))) = L_V(x)$$

with probability $1 - \delta$ over the choice of $x \sim \mathcal{D}$, F^*, and the randomness used by the reduction.

[8] Technically, a witness oracle is an ensemble of distributions over function families $\{F_n\}$, but to simplify notation we will identify samples from this ensemble with the ensemble itself.

This definition subsumes the case of a worst-to-average reduction: A δ' worst-to-average reduction is simply a 0-to-δ' average-to-average reduction. The other type of reduction used in the analysis—the search-to-decision reduction—is formalized in a similar way as follows.

DEFINITION 5.2 (search-to-decision reduction). *Let V be an* NP *relation, L' be an* NP *language, and $\mathcal{D}, \mathcal{D}'$ be polynomial-time samplable distribution ensembles. A* δ-to-δ' search-to-decision reduction *for search problems from $(V, \mathcal{D})$ to $(L', \mathcal{D}')$ is a family of polynomial size circuits $R = \{R_n\}$ such that on input $x \in \{0,1\}^n$, randomness r, $R_n(x; r)$ outputs strings $y_1, \ldots, y_k$, and a circuit C such that for any L^* that is δ'-close to L' with respect to $\mathcal{D}'$, it holds that*

$$V(x, C(L^*(y_1), \ldots, L^*(y_k))) = L_V(x)$$

with probability $1 - \delta$ over the choice of $x \sim \mathcal{D}$ and the randomness used by the reduction.

5.2. Reductions to distributional search problems. We now state the main result of this section.

THEOREM 5.3. *Let L be a language, V' be an* NP *relation, $\mathcal{D}'$ be an arbitrary polynomial-time samplable ensemble of distributions, and c be a constant. If there is a nonadaptive n^{-c} worst-to-average reduction from L to $(V', \mathcal{D}')$, then $L \in$ NP/poly $\cap$* coNP/poly.

To understand the meaning of Theorem 5.3, consider a polynomial-time computable function f and a samplable ensemble of inputs $\mathcal{D} = \{\mathcal{D}_n\}$, and suppose that we want to prove that f is a one-way function with respect to the distribution $\mathcal{D}$. (That is, for a random $x \sim \mathcal{D}_n$, it is hard on average to find a preimage of $f(x)$.) We may set our aim low and try only to prove that f is just infinitely often a weak one-way function. This means that there is a polynomial p such that, for every polynomial-time inverter A, the computation $A(f(x))$ fails with probability at least $1/p(n)$ to output a preimage of $f(x)$, where the probability is over the coin tosses of A and the sampling of x from $\mathcal{D}_n$, and the statement is true for infinitely many n. We could try to provide evidence for the hardness of f by giving a reduction showing that an adversary that inverts A with probability better than $1 - 1/p(n)$ on all input lengths would imply a BPP algorithm for a presumably hard language L. Theorem 5.3 implies that if such a reduction is nonadaptive, then $L \in$ coNP/poly, and if L were NP-hard we would have a collapse of the polynomial hierarchy. Specifically, in order to apply Theorem 5.3 to our setting, consider the NP relation V' made of pairs $(f(x), x)$, and define the distribution $\mathcal{D}'$ as the ensemble $\{f(x)\}$ when x is sampled from $\mathcal{D}$. Then solving the search problem of V' on a random instance of $\mathcal{D}'$ is the same as inverting $f(x)$ on a random x taken from $\mathcal{D}$. A nonadaptive reduction of a decision problem L to such a problem implies that $L \in$ coNP/poly.

Theorem 5.3 is an immediate consequence of Theorem 4.1 and the following two lemmas.

LEMMA 5.4. *For every $\delta = \delta(n)$ and* NP *relation $V \subseteq \cup_n \{0,1\}^n \times \{0,1\}^{m(n)}$ there exists an* NP *language L' for which there is an $O(\delta(m(n))^2)$-to-δ average-to-average reduction from $(V, \mathcal{U})$ to $(L', \mathcal{U})$.*

LEMMA 5.5. *For every $\delta = \delta(n)$,* NP *relation V, and polynomial-time samplable ensemble of distributions $\mathcal{D}$ there exists a constant c and an* NP *relation V' for which there is an $O(\delta n^c)$-to-δ average-to-average reduction from $(V, \mathcal{D})$ to $(V', \mathcal{U})$.*

Analogues of Lemmas 5.4 and 5.5 are known in the context of the distributional hardness of NP-problems. A variant of Lemma 5.4 appears in Ben-David et al. [7],

while a variant of Lemma 5.5 was proved by Impagliazzo and Levin [29]. Our proofs are in essence a recasting of these arguments in the formalism of nonadaptive average-to-average reductions. These proofs are presented in sections A.3 and A.4, respectively.

Appendix A.

A.1. Additive bounds for sampling. We provide the proof of Lemma 3.1 below.

Proof of Lemma 3.1. Let $N = |S|$ and $p = \mathcal{D}(T)$. We use the following form of the Chernoff bound (see [36, section 4.1]):

$$\Pr[|T \cap S| < (1-\xi)Np] < \exp(-\xi^2 Np/2) \text{ for } \xi < 1$$

and

$$\Pr[|T \cap S| > (1+\xi)Np] < \begin{cases} (4/e)^{-(1+\xi)Np} \text{ for } \xi > 1, \\ \exp(-\xi^2 Np/3) \text{ for } \xi \le 1. \end{cases}$$

If $p < \epsilon$, the lower bound holds trivially, and for the upper bound we set $\xi = \epsilon/p > 1$ to obtain

$$\Pr[|T \cap S| > (p+\epsilon)N] < (4/e)^{-(p+\epsilon)N} < \eta.$$

If $p \ge \epsilon$, we set $\xi = \epsilon$ to obtain

$$\Pr[|T \cap S| \notin (1 \pm \epsilon)Np] < 2\exp(-\epsilon^2 Np/3) \le 2\exp(-\epsilon^3 N/3) < \eta. \qquad \square$$

A.2. Proof sketches for the lower and upper bound protocols. We provide proof sketches for Lemmas 2.6 and 2.8 below.

Proof sketch for Lemma 2.6. Let $S = C^{-1}(1)$. For $r \in S$, let I_r be an indicator for the event $h(r) = 0$, and let $R = h^{-1}(0)$, so that $|R| = \sum_{r \in S} I_r$. By the pairwise independence of h, we have $\mathrm{E}[|R|] = |S|k/s$, $\mathrm{Var}[|R|] \le |S|k/s$, so by Chebyshev's inequality,

$$\Pr\left[|R| \notin \left(1 \pm \frac{\epsilon}{3}\right)\frac{|S|k}{s}\right] \le \frac{9}{\epsilon^2} \cdot \frac{s}{|S|k}.$$

The bounds now follow by direct calculation. $\square$

Proof sketch for Lemma 2.8. Let $S = C^{-1}(1)$. Fix r and let $S' = S - \{r\}$, $k' = |S'|/|\Gamma|$, $R = h^{-1}(h(r))$, $R' = R - \{r\}$. For every $r' \in S'$, let $I_{r'}$ be an indicator for the event $r' \in R'$, so that $|R'| = \sum_{r' \in S'} I_{r'}$. By the 3-wise independence of h, the $I_{r'}$ are pairwise independent conditioned on $h(r)$, so that $\mathrm{E}[|R'|] = k'$, $\mathrm{Var}[|R'|] = k'(1 - 1/|\Gamma|) < k'$, and by Chebyshev's inequality, for every $\xi > 0$,

$$\Pr[|R'| \notin (1 \pm \xi)k'] < 1/\xi^2 k'.$$

Suppose $|S| \le s$. Without loss of generality we may assume $|S| = s$, since for larger values of s the acceptance probability may only increase. In this case $k' = k$, so that

$$\Pr[|R| > (1+\epsilon/3)k] = \Pr[|R'| \ge (1+\epsilon/3)k] < 9/\epsilon^2 k.$$

Given that $|R| \le (1+\epsilon/3)k$, the prover can list all elements of R, so that $R = \{r_1, \ldots, r_l\}$ and $l \le (1+\epsilon/3)k$. In particular, this ensures that $r \in R$ and the verifier accepts.

Now suppose $|S| \ge (1+\epsilon)s$, so that $k' > (1+\epsilon)k$. Then

$$\Pr\left[|R'| < \left(1+\frac{\epsilon}{2}\right)k\right] < \Pr\left[|R'| < \frac{1+\epsilon/2}{1+\epsilon}k'\right] < \Pr\left[|R'| < \left(1-\frac{\epsilon}{3}\right)k'\right] < \frac{9}{\epsilon^2 k}.$$

Given that $|R| > (1+\epsilon/2)k$, what is the best strategy for a prover to make the verifier accept? Conditioned on $(h, h(r))$, r is uniformly distributed in R, so the best the prover can do is set $l = (1+\epsilon/3)k$ and pick $\{r_1, \ldots, r_l\}$ to be an *arbitrary* subset of R. In this case,

$$\Pr[r \in \{r_1, \ldots, r_l\}] = \frac{l}{|R|} < \frac{(1+\epsilon/3)k}{(1+\epsilon/2)k} < 1 - \frac{\epsilon}{6}. \qquad \square$$

A.3. Reductions from decision to search problems. We provide the proof of Lemma 5.4 below.

Proof of Lemma 5.4. As in [7], we first reduce the search problem V to a search problem with a unique witness, then encode the bits of the witness in the language L'. The first step is based on the hashing argument of Valiant and Vazirani [41]. The reduction, as described below, succeeds only with probability 1/16, but this can be amplified to 2/3 by applying the reduction three times.

The inputs of the language L' are of the form (x, k, j, h), where x is an instance of L_V, k and j are integers between 0 and $m(|x|)$, and h is a pairwise independent hash function mapping $|x|$ bits to k bits (padded appropriately so the length of (x, k, j, h) is a fixed polynomial of $|x|$ only). Let w_i denote the ith bit of a string w. We define

> $(x, k, j, h) \in L'$ if there exists a w of length $m(|x|)$ for which $h(w) = 0$ and $w_i = 1$.

It is immediate that $L' \in$ NP.

The reduction works as follows: On input $x \in \{0,1\}^n$, choose a random h uniformly at random and generate the queries $q_{kj} = (x, k, j, h)$ for all $1 \le k \le m$ and $1 \le j \le m$. Let $a_{kj} \in \{0,1\}$ denote the claimed answer to query q_{kj} and w_k be the concatenation $a_{k1} \ldots a_{km}$. The decider looks for an index k such that w_k is a witness for x for all $1 \le j \le m$ and outputs w_k; if no such k is found, the decider returns an arbitrary answer.

Let L^* be an arbitrary decision oracle that is δ-close to L'. Say an input x is *good* in L^* if for all $1 \le k \le m$, $1 \le j \le m$,

$$\Pr_{h, L^*}[L^*(x, k, j, h) = L(x, k, j, h)] > 15/16.$$

By a pigeonhole argument, $x \sim \mathcal{U}_n$ is good with probability at least $1 - O(\delta(m(n))^2)$. We show that the reduction succeeds on a good input with probability 1/16. By the Valiant–Vazirani argument, for $k = \lfloor \log_2 |\{w : V \text{ accepts } (x; w)\}| \rfloor$, with probability 1/8 there exists a unique w such that $h(w) = 0$. It follows that whenever x is good and $x \in L_V$, with probability at least 1/16, $L^*(x, k, j, h) = w_k$ for all $1 \le j \le m$, so the decider encounters the witness w_k. $\square$

Remark. The argument can be strengthened to obtain an $O(\delta m(n))$-to-δ average-to-average reduction from (V, U) to (L', U) by applying a Hadamard code to the witness w in L' instead of revealing its bits.

A.4. Reductions for arbitrary samplable distributions. We provide the proof of Lemma 5.5 below.

Proof of Lemma 5.5. Let S denote the sampler that yields the distribution ensemble $\mathcal{D}$: S is a polynomial-time computable function $\{0,1\}^n \times \{0,1\}^{s(n)} \to \{0,1\}^n$, such that $S(1^n, \mathcal{U}_{s(n)}) = \mathcal{D}_n$.

We want to be able to map instances of V into instances of V' in such a way that witnesses for V can be recovered from witnesses for V', and so that for most x, the probability of an image of x in the uniform distribution is polynomially related to the probability of x in distribution $\mathcal{D}$.

Let V' be an NP relation for language L', which we define next. The inputs of L are of the form (n, k, h_1, z, h_2), where

(i) the integer n will denote the length of the input to the reduction coming from $\mathcal{D}$;

(ii) the integer $k \in [s]$ $(s = s(n))$ will encode the approximate likelihood of the input to the reduction according to $\mathcal{D}$;

(iii) $h_1 : \{0,1\}^n \to \{0,1\}^{k+6}$ is a pairwise independent hash function, and $z \in \{0,1\}^{k+6}$ is an element in the range of h_1;

(iv) $h_2 : \{0,1\}^s \to \{0,1\}^{s-k-3}$ is another pairwise independent hash function.

Suppose that the inputs (n, k, h_1, z, h_2) are padded appropriately so that their length depends on n only.

A pair (w, r) is an NP-witness for input (n, k, h_1, z, h_2) in V' if the following three conditions are satisfied: (1) $V(S(1^n, r); w) = 1$; (2) $h_1(S(1^n, r)) = z$; (3) $h_2(r) = 0$.

On input x, where $|x| = n$, the reduction produces queries $(n, k, h_1, h_1(x), h_2)$ for all possible values of k by choosing h_1 and h_2 uniformly at random. The decider looks at all answers (w_k, r_k) and returns w_k if $V(S(1^n, r_k); w_k) = 1$ for some k. If no such k is found, the decider returns the string 0^m.

Suppose F^* is a δ-approximate oracle for V' with respect to the uniform ensemble. Given $x \in \{0,1\}^n$, we call an instance (n, k, h_1, z, h_2) *good* for x if the following three conditions are satisfied:

(i) $|x| = n$, $\lfloor -\log_2 \mathcal{D}(x) \rfloor = k$, and $h_1(x) = z$.

(ii) There exists an r such that $h_1(S(1^n, r)) = z$ and $h_2(r) = 0$.

(iii) If, for some r, $h_1(S(1^n, r)) = z$ and $h_2(r) = 0$, then $S(1^n, r) = x$.

Let $G(x)$ denote the set of all queries in L' that are good for x. It is immediate that the sets $G(x)$ are pairwise disjoint over all $x \in \{0,1\}^n$. On the one hand, we will show that, on input x, the reduction has a constant probability of producing a query that lands in $G(x)$. Moreover, conditioned on k, this query is uniformly distributed in $G(x)$. If $x \in L$ and F^* and V' agree on the query that falls within $G(x)$, then $F^*(x) = (w, r)$ with $S(1^n, r) = x$, so $V(x; w) = 1$. In addition, we will show that when $x \sim \mathcal{D}$, with probability at least $1 - \delta s$, F^* and V' do agree on a constant fraction of $G(x)$ for every k, so that the reduction has a constant probability of producing a query on which F^* and V' agree.

Claim A.1. *Suppose $|x| = n$ and $\lfloor -\log_2 \mathcal{D}(x) \rfloor = k$. With probability 3/4 over the choice of h_1 and h_2, the instance (n, k, h_1, z, h_2) is in $G(x)$.*

Proof. We first show that, with probability 7/8, the instance satisfies the second condition for goodness; i.e., there exists an r such that $S(1^n, r) = x$ and $h_2(r) = 0$. If $S(1^n, r) = x$, let I_r be an indicator for the event $h_2(r) = 0$. By our choice of k, $|\{r : S(1^n, r) = x\}| \geq 2^{s-k}$, so that

$$\mathrm{E}\left[\textstyle\sum_{r:S(1^n,r)=x} I_r\right] \geq 2^{s-k}\mathrm{E}[I_r] = 8.$$

As the I_r are pairwise independent, the variance of this sum is at most the expectation,

so by Chebyshev's inequality at least one $I_r = 1$ with probability 7/8.

Now we look at the probability of satisfying the third condition for goodness. Fix r such that $S(1^n, r) \neq x$. By pairwise independence, $\Pr_{h_1}[h_1(S(1^n, r)) = h_1(x)] = 2^{-k-6}$, and independently, $\Pr_{h_2}[h_2(r) = 0] = 2^{-s+k+3}$. It follows that

$$\begin{aligned}\Pr[\exists r : S(1^n, r) \neq x \text{ and } h_1(S(1^n, r)) = h_1(x) \text{ and } h_2(r) = 0] \\ \leq \sum\nolimits_{r:S(1^n,r)\neq x} \Pr[h_1(S(1^n, r)) = h_1(x)] \Pr[h_2(r) = 0] \\ \leq \sum\nolimits_{r\in\{0,1\}^s} 2^{-k-6}2^{-s+k+3} = 1/8.\end{aligned}$$

It follows that both conditions for goodness are satisfied with probability at least 3/4. ∎

Claim A.2. *For every x, $\mathcal{U}(G(x)) \geq (3/64)(\mathcal{D}(x)/ns)$.*

Proof. Consider a random string (n, k, h_1, z, h_2). With probability $1/ns$, $n = |x|$ and $k = \lfloor -\log_2 \mathcal{D}(x) \rfloor$. Conditioned on this, $z = h_1(x)$ with probability 2^{-k-3}. By the last claim, with probability 3/4 over h_1 and h_2, $(n, k, h_1, h_1(x), h_2)$ is in $G(x)$. Putting this together,

$$\Pr[(n, k, h_1, z, h_2) \in G(x)] \geq \frac{1}{ns} \cdot \frac{3}{4} \cdot 2^{-k-3} \geq \frac{3}{64} \cdot \frac{\mathcal{D}(x)}{ns}. \qquad \blacksquare$$

Let Z denote the set of all $x \in L_V$ for which

$$\Pr_{y\sim\mathcal{U},F^*}[V'(y, F^*(y)) = 1 \mid y \in G(x)] > 8/9,$$

so that if the kth query q_k lands in $G(x)$, the answer (w_k, r_k) has an 8/9 probability of being a good witness for the query. It follows that, unconditionally, $V(q_k, F^*(q_k)) = 1$ with probability at least $8/9 \cdot 3/4 = 2/3$, and the decider is successful on the queries that come from S.

On the other hand, by the disjointness of the sets $G(x)$,

$$\begin{aligned}\delta &\geq \sum\nolimits_{x\in L_V} \mathcal{U}(G(x)) \Pr_{y\sim\mathcal{U}}[V'(y, F^*(y)) = 0 \mid y \in G(x)] \\ &> \sum\nolimits_{x\in\overline{Z}} \mathcal{U}(G(x)) \cdot \frac{1}{9} \\ &\geq \sum\nolimits_{x\in\overline{Z}} \frac{1}{9} \cdot \frac{3}{64} \cdot \frac{\mathcal{D}(x)}{ns} \text{ (by Claim A.2)} \\ &= \Omega(\mathcal{D}(\overline{Z})/ns),\end{aligned}$$

so that $\mathcal{D}(\overline{Z}) = O(\delta ns)$. ∎

Acknowledgments. We thank Madhu Sudan for suggesting the relevance of [29], Oded Goldreich for stressing the relevance of our result to the question of basing cryptography on NP-hardness, and Amit Sahai for helpful discussions. We also thank Oded Goldreich and an anonymous reviewer for many useful comments on the presentation. The hiding protocol was suggested by Manikandan Narayanan.

REFERENCES

[1] M. Abadi, J. Feigenbaum, and J. Kilian, *On hiding information from an oracle*, J. Comput. System Sci., 39 (1989), pp. 21–50.

[2] W. Aiello and J. Håstad, *Statistical zero-knowledge languages can be recognized in two rounds*, J. Comput. System Sci., 42 (1991), pp. 327–345.

[3] M. Ajtai, *Generating hard instances of lattice problems*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing, ACM, New York, 1996, pp. 99–108.

[4] M. Ajtai and C. Dwork, *A public-key cryptosystem with worst-case/average-case equivalence*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, ACM, New York, 1997, pp. 284–293.

[5] D. Beaver and J. Feigenbaum, *Hiding instances in multioracle queries*, in Proceedings of the 7th Annual Symposium on Theoretical Aspects of Computer Science, Springer, Berlin, 1990, pp. 37–48.

[6] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway, *Locally random reductions: Improvements and applications*, J. Crypt., 10 (1997), pp. 17–36.

[7] S. Ben-David, B. Chor, O. Goldreich, and M. Luby, *On the theory of average-case complexity*, in Proceedings of the 21st Annual ACM Symposium on Theory of Computing, ACM, New York, 1989, pp. 204–216.

[8] M. Blum, *Designing Programs to Check Their Work*, Tech. Report 88-09, International Computer Science Institute, Berkeley, CA, 1988. Available online at http://www.icsi.berkeley.edu/cgi-bin/pubs/publication.pl?ID-000495

[9] M. Blum and S. Kannan, *Designing programs that check their work*, J. ACM, 41 (1995), pp. 269–291.

[10] M. Blum, M. Luby, and R. Rubinfeld, *Self-testing/correcting with applications to numerical problems*, J. Comput. System Sci., 47 (1993), pp. 549–595.

[11] A. Bogdanov and L. Trevisan, *Average-Case Complexity: A Survey*, in preparation, 2006.

[12] G. Brassard, *Relativized cryptography*, in Proceedings of the 20th Annual IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 1979, pp. 383–391.

[13] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, *Private information retrieval*, J. ACM, 45 (1998), pp. 965–982.

[14] W. Diffie and M. E. Hellman, *New directions in cryptography*, IEEE Trans. Inform. Theory, I22 (1976), pp. 644–654.

[15] S. Even, A. L. Selman, and Y. Yacobi, *The complexity of promise problems with applications to public-key cryptography*, Inform. Control, 61 (1984), pp. 159–173.

[16] S. Even and Y. Yacobi, *Cryptography and NP-completeness*, in Proceedings of the 7th Annual ICALP, Lecture Notes in Comput. Sci. 85, Springer-Verlag, Berlin, 1980, pp. 195–207.

[17] J. Feigenbaum and L. Fortnow, *Random-self-reducibility of complete sets*, SIAM J. Comput., 22 (1993), pp. 994–1005.

[18] J. Feigenbaum, S. Kannan, and N. Nisan, *Lower bounds on random-self-reducibility*, in Proceedings of the 5th Annual IEEE Conference on Structure in Complexity Theory, IEEE, Los Alamitos, CA, 1990, pp. 100–109.

[19] L. Fortnow, *The complexity of perfect zero-knowledge*, in Proceedings of the 19th Annual ACM Symposium on Theory of Computing, ACM, New York, 1987, pp. 204–209.

[20] O. Goldreich, *Notes on Levin's Theory of Average-Case Complexity*, Tech. Report TR97–058, Electronic Colloquium on Computational Complexity (ECCC), University of Trier, Trier, Germany, 1997. Available online at http://eccc.hpi-web.de/eccc-reports/1997/TR97-058/index.html

[21] O. Goldreich, *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, Algorithms Combin. 17, Springer-Verlag, Berlin, 1999.

[22] O. Goldreich, *On Promise Problems*, Tech. Report TR05–018, Electronic Colloquium on Computational Complexity (ECCC), University of Trier, Trier, Germany, 2005. Available online at http://eccc.hpi-web.de/eccc-reports/2005/TR05-018/index.html

[23] O. Goldreich and S. Goldwasser, *On the Possibility of Basing Cryptography on the Assumption that $P \neq NP$*, unpublished manuscript, 1998.

[24] O. Goldreich, H. Karloff, L. Schulman, and L. Trevisan, *Lower bounds for linear locally decodable codes and private information retrieval*, in Proceedings of the 17th Annual IEEE Conference on Computational Complexity, IEEE, Los Alamitos, CA, 2002, pp. 175–183.

[25] O. Goldreich, S. Vadhan, and A. Wigderson, *On interactive proofs with a laconic prover*, in Proceedings of the 28th Annual ICALP, Lecture Notes in Comput. Sci. 2076, Springer-Verlag, Berlin, 2001, pp. 334–345.

[26] S. Goldwasser and M. Sipser, *Private coins versus public coins in interactive proof systems*, in Proceedings of the 18th Annual ACM Symposium on Theory of Computing, ACM, New York, 1986, pp. 59–68.
[27] A. Healy, S. Vadhan, and E. Viola, *Using nondeterminism to amplify hardness*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, ACM, New York, 2004, pp. 192–201.
[28] R. Impagliazzo, *A personal view of average-case complexity*, in Proceedings of the 10th Annual IEEE Conference on Structure in Complexity Theory, IEEE, Los Alamitos, CA, 1995, pp. 134–147.
[29] R. Impagliazzo and L. Levin, *No better ways to generate hard NP instances than picking uniformly at random*, in Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 1990, pp. 812–821.
[30] J. Katz and L. Trevisan, *On the efficiency of local decoding procedures for error correcting codes*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, ACM, New York, 2000, pp. 80–86.
[31] A. Lempel, *Cryptology in transition*, ACM Computing Surveys, 11 (1979), pp. 285–303.
[32] L. A. Levin, *Average case complete problems*, SIAM J. Comput., 15 (1986), pp. 285–286.
[33] R. Lipton, *New directions in testing*, in Proceedings of DIMACS Workshop on Distributed Computing and Cryptography, AMS, Providence, RI, 1989, pp. 191–202.
[34] D. Micciancio, *Almost perfect lattices, the covering radius problem, and applications to Ajtai's connection factor*, SIAM J. Comput., 34 (2004), pp. 118–169.
[35] D. Micciancio and O. Regev, *Worst-case to average-case reductions based on Gaussian measure*, in Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 2004, pp. 372–381.
[36] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.
[37] R. O'Donnell, *Hardness amplification within NP*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, ACM, New York, 2002, pp. 751–760.
[38] O. Regev, *New lattice based cryptographic constructions*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, ACM, New York, 2003, pp. 407–416.
[39] M. Sudan, L. Trevisan, and S. Vadhan, *Pseudorandom generators without the XOR lemma*, J. Comput. System Sci., 62 (2001), pp. 236–266.
[40] L. Trevisan, *On uniform amplification of hardness in NP*, in Proceedings of the 37th Annual ACM Symposium on Theory of Computing, ACM, New York, 2005, pp. 31–38.
[41] L. Valiant and V. Vazirani, *NP is as easy as detecting unique solutions*, Theoret. Comput. Sci., 47 (1986), pp. 85–93.
[42] C. K. Yap, *Some consequences of nonuniform conditions on uniform classes*, Theoret. Comput. Sci., 26 (1983), pp. 287–300.

SIAM J. COMPUT.
Vol. 36, No. 4, pp. 1160–1214

AN UNCONDITIONAL STUDY OF COMPUTATIONAL ZERO KNOWLEDGE*

SALIL P. VADHAN†

Abstract. We prove a number of general theorems about **ZK**, the class of problems possessing (computational) zero-knowledge proofs. Our results are *unconditional*, in contrast to most previous works on **ZK**, which rely on the assumption that one-way functions exist. We establish several new characterizations of **ZK** and use these characterizations to prove results such as the following:

1. Honest-verifier **ZK** equals general **ZK**.
2. Public-coin **ZK** equals private-coin **ZK**.
3. **ZK** is closed under union.
4. **ZK** with imperfect completeness equals **ZK** with perfect completeness.
5. Any problem in $\mathbf{ZK} \cap \mathbf{NP}$ can be proven in computational zero knowledge by a $\mathbf{BPP}^{\mathbf{NP}}$ prover.
6. **ZK** with black-box simulators equals **ZK** with general, non–black-box simulators.

The above equalities refer to the resulting *class* of problems (and do not necessarily preserve other efficiency measures such as round complexity). Our approach is to combine the conditional techniques previously used in the study of **ZK** with the unconditional techniques developed in the study of **SZK**, the class of problems possessing statistical zero-knowledge proofs. To enable this combination, we prove that every problem in **ZK** can be decomposed into a problem in **SZK** together with a set of instances from which a one-way function can be constructed.

Key words. cryptography, computational complexity, zero-knowledge proofs, pseudoentropy, language-dependent commitment schemes, auxiliary-input one-way functions

AMS subject classifications. 94A60, 68Q15

DOI. 10.1137/S0097539705447207

1. Introduction.

Since their introduction by Goldwasser, Micali, and Rackoff [35], zero-knowledge interactive proofs have become a central tool in cryptographic protocol design, and have also provided fertile grounds for complexity-theoretic investigations into the interplay between fundamental notions such as proofs, randomness, interaction, and secrecy.

The notion of zero-knowledge proofs raised a number of intriguing basic questions, such as the following:

- Can we characterize the class **ZK** of problems possessing zero-knowledge proofs?[1]
- Can we transform proof systems that are zero knowledge for the "honest verifier" (i.e., the verifier that follows the specified protocol) into ones that are zero knowledge in general (i.e., for all polynomial-time verifier strategies)?

*Received by the editors March 18, 2005; accepted for publication (in revised form) April 4, 2006; published electronically December 15, 2006. This work was done while the author was a Fellow at the Radcliffe Institute for Advanced Study. This research was also supported by NSF grants CNS-0430336, CCR-0205423, and CCR-0133096, and by ONR grant N00014-04-1-0478 and a Sloan Research Fellowship. Preliminary versions of this paper have appeared in *Proceedings of the IEEE Symposium on Foundations of Computer Science* [58] and *Electronic Colloquium on Computational Complexity* [59].

http://www.siam.org/journals/sicomp/36-4/44720.html

†Division of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138 (salil@eecs.harvard.edu).

[1] In this paper, we focus on the original notion of *computational* zero-knowledge *proof* systems, as introduced in [35]. That is, the zero-knowledge condition is defined with respect to computationally bounded verifiers (and distinguishers), and the soundness is defined with respect to computationally unbounded prover strategies. In particular, we do not consider *argument* systems [10], which are only computationally sound.

That is, does **HVZK** = **ZK**, where **HVZK** denotes the class of problems possessing honest-verifier zero-knowledge proofs?

- Is it always possible to modify zero-knowledge proofs to have additional useful properties—such as having a small number of rounds, perfect completeness, or public coins? Or do the latter properties restrict the class of problems possessing zero-knowledge proofs?
- What closure properties does **ZK** have? Is it closed under complement? Under union?

Almost all of these questions were seemingly resolved by a series of exciting works that appeared within a few years after zero-knowledge proofs were defined. Specifically, under the assumption that one-way functions (OWF) exist, it was shown that **ZK** "hits the roof," namely, **ZK** = **IP**, where **IP** is the class of problems possessing interactive proofs [29, 39, 7, 45, 37]. Thus, **ZK** is completely characterized and, moreover, has natural complete problems (namely, any complete problem for **IP** = **PSPACE** [42, 54]). This also implies that **HVZK** equals **ZK**, since **ZK** $\subseteq$ **HVZK** $\subseteq$ **IP** is immediate from the definitions. In addition, the equality **ZK** = **IP** is proven by a generic transformation from interactive proofs into zero-knowledge proofs, and this transformation preserves many properties, such as those mentioned above, i.e., the round complexity,[2] public coins, and perfect completeness. Since we already know how to transform interactive proofs into interactive proofs with public coins [36] and perfect completeness [19], it follows that we can also transform zero-knowledge proofs into zero-knowledge proofs with the same properties. **ZK** also inherits all the closure properties of **IP** = **PSPACE**, in particular closure under complement and union. However, *all of these results are based on the assumption that one-way functions exist*, and without this assumption, all the questions listed above are open.

In this paper, we answer most of these questions *unconditionally* (i.e., without any unproven complexity assumption). In particular, we do the following:

- Give several characterizations of **ZK** that make no reference to interaction or zero knowledge. (These characterizations are not quite complete problems, but turn out to have similar utility.)
- Prove that **HVZK** = **ZK**.
- Show how to transform any computational zero-knowledge proof into one with public coins and perfect completeness.
- Establish closure properties of **ZK**, such as closure under union.

This paper is inspired by the work of Ostrovsky and Wigderson [50], who gave the first hint that it might be possible to prove unconditional results about zero knowledge. They showed that if computational zero knowledge is nontrivial (i.e., **ZK** $\neq$ **BPP**), then "some form of one-way functions" exists. Then they made the appealing suggestion that one might prove unconditional results about computational zero knowledge by a case analysis as follows: If **ZK** = **BPP**, then many results about **ZK** hold trivially (because every problem in **BPP** has a trivial zero-knowledge proof, where the prover sends nothing and the verifier decides membership on its own, using the **BPP** algorithm). On the other hand, if **ZK** $\neq$ **BPP**, then we can try to use their "one-way functions" in the known conditional results about **ZK**. Unfortunately, as they point out, this approach does not work because the form of one-way functions they construct (in this case, **ZK** $\neq$ **BPP**) is too weak

[2] The round complexity is preserved up to an additive constant for achieving a polynomially small soundness error. For a negligible error, any superconstant multiplicative factor suffices (by sequential repetition).

for the conditional constructions mentioned above.[3] (See section 7.1 for more details.)

Our approach is to replace **BPP** with **SZK**, the class of problems possessing *statistical* zero-knowledge proofs (to be described in more detail shortly). In particular, in the case when $\mathbf{ZK} \neq \mathbf{SZK}$, we are able to construct a form of one-way functions that is much closer to the standard notion than that in the Ostrovsky–Wigderson result. However, now the case that $\mathbf{ZK} = \mathbf{SZK}$ is not as trivial as before; instead we rely on a large body of previous work giving unconditional results about **SZK** (as described below). To make this approach work, we actually carry out the case analysis on an input-by-input basis. That is, we show that for every problem in **ZK**, we can partition its instances into "**SZK** instances" and "one-way function instances." This characterization is described in more detail below.

1.1. The SZK/OWF characterization.

Statistical zero knowledge. The distinction between general (computational) zero knowledge and statistical zero knowledge involves the formulation of the "zero-knowledge" property, i.e., the requirement that the verifier "learns nothing" from the interaction other than the fact that the assertion being proven is true. The original (and most general) notion discussed above, called *computational zero knowledge*, informally says that a *polynomial-time* verifier learns nothing. *Statistical zero knowledge* guarantees that even a computationally unbounded verifier learns nothing from the interaction.[4] Naturally, the stronger security guarantee of statistical zero knowledge is preferable, but unfortunately it seems to severely constrain the class of statements that can be proven in zero knowledge. Specifically, it is known that the class **SZK** of problems possessing statistical zero-knowledge proofs is contained in $\mathbf{AM} \cap \mathbf{co\text{-}AM}$ [18, 1], and thus **NP**-complete problems are unlikely to have statistical zero-knowledge proofs. Thus statistical zero-knowledge proofs do not seem to have the wide applicability of computational zero-knowledge proofs (which stems from the existence of computational zero-knowledge proofs for all of **NP** [29]).

Nevertheless, the class **SZK** of problems possessing statistical zero-knowledge proofs has turned out to be a rich object of study, and in recent years, there have been a number of results substantially improving our understanding of it. These results include the identification of natural complete problems for class **SZK** [52, 34], showing that **SZK** is closed under complement [48]; that honest-verifier **SZK** equals general **SZK** [32]; and that private-coin **SZK** equals public-coin **SZK** [48]. (See [57] for a unified presentation of all these results.) In contrast to what was known for computational zero knowledge, all these results are unconditional. That is, they do not rely on any unproven complexity assumptions (such as the existence of one-way functions).

It was suggested in [34, 57] that the study of **SZK** could provide a useful testbed for understanding zero knowledge before moving on to more complex models that

[3] A similar approach was used in an attempt to prove $\mathbf{HVSZK} = \mathbf{SZK}$ [15], but subsequently a more direct approach that avoids these difficulties was found [32].

[4] Recall that the zero-knowledge property is formalized by requiring that there be a probabilistic polynomial-time algorithm S that "simulates" the verifier's view of the interaction (when the assertion being proven is true). In computational zero knowledge, the output distribution of the simulator is required only to be computationally indistinguishable from the verifier's view of the interaction, whereas in statistical zero knowledge, it must be statistically close. We note that there is a similar choice in the soundness condition. We, like the authors of [35], focus on interactive *proof* systems, where even a *computationally unbounded* prover cannot convince the verifier to accept a false statement, except with negligible probability. In interactive *argument* systems [10], this soundness condition is required only for *polynomial-time* provers.

incorporate computational intractability (such as **ZK**). In this paper, we make extensive use of that methodology—not just proving results about **ZK** by analogy to **SZK**, but actually making direct use of known results about **SZK** (e.g., in establishing and using the characterization below).

The characterization. In this paper, we provide a new characterization of **ZK** in terms of **SZK** and one-way functions as follows.

DEFINITION 1.1. *A promise problem*[5] $\Pi = (\Pi_Y, \Pi_N)$ *satisfies the* SZK/OWF CONDITION *if there exists a set* $I \subseteq \Pi_Y$ *of* YES *instances, a polynomial-time computable function* f*, and a polynomial* $p(n)$ *such that the following hold:*

- *Ignoring the inputs in* I*, the problem* Π *has a statistical zero-knowledge proof. Formally, we have* $\Pi' \in \mathbf{SZK}$*, where* $\Pi' = (\Pi_Y \setminus I, \Pi_N)$.
- *When* $x \in I$*, the function* $f_x(\cdot) \stackrel{\text{def}}{=} f(x, \cdot)$ *is hard to invert. That is, for every nonuniform polynomial-time algorithm* A*, there exists a negligible function* ϵ *such that for every* $x \in I$*,*

$$\Pr\left[A(f_x(U_{p(|x|)})) \in f_x^{-1}(f_x(U_{p(|x|)}))\right] \leq \epsilon(|x|).$$

Intuitively, this characterization says that for every YES instance x, either one can prove the membership of x in Π_Y in statistical zero knowledge ("x is an **SZK** instance") or one can use x to construct a one-way function that is given x as an auxiliary input ("x is an OWF instance"). Note that if one-way functions exist (in the standard sense, i.e., without auxiliary input), then *all* promise problems satisfy the SZK/OWF CONDITION (by setting $I = \Pi_Y$, and $f_x(y) = g(y)$ where g is the one-way function assumed to exist).

On the other hand, the above condition (regarding Π) alone *cannot* characterize **ZK**, since if one-way functions do exist, Π will satisfy Definition 1.1 even if $\Pi \notin \mathbf{IP}$. We prove that if we simply add the condition $\Pi \in \mathbf{IP}$, then we do indeed obtain an exact characterization.

THEOREM 1.2 (SZK/OWF characterization of ZK). $\Pi \in \mathbf{ZK}$ *if and only if* $\Pi \in \mathbf{IP}$ *and* Π *satisfies the* SZK/OWF CONDITION.

As noted above, the usefulness of this characterization is that it essentially reduces the unconditional study of **ZK** to its conditional study plus the study of **SZK**. Theorem 1.2 is in some sense the central theorem of this paper; all other results are deduced as consequences of it or its proof. When proving each direction of Theorem 1.2, we actually prove stronger statements than required. In the forward ("only if") direction, we actually show that every problem in **HVZK**, not just **ZK**, satisfies the SZK/OWF CONDITION. In the reverse ("if") direction, we show that every problem in **IP** satisfying the SZK/OWF CONDITION is not only in **ZK**, but has a computational zero-knowledge proof with many nice properties, such as public coins, perfect completeness, universal black-box simulation, etc. Combining the two directions, we deduce that **HVZK** = **ZK**, and that every problem in **ZK** has a computational zero-knowledge proof with the aforementioned properties.

1.2. Proof outline. Figure 1 illustrates our main steps in establishing both directions of Theorem 1.2. In proving the forward direction, we first prove that every problem in **HVZK** satisfies a "CONDITIONAL PSEUDOENTROPY CONDITION" and an "INDISTINGUISHABILITY CONDITION." These are computational analogues

[5] A promise problem Π consists of a pair (Π_Y, Π_N) of disjoint sets of strings, corresponding to the YES instances and NO instances of Π, respectively [17]. All complexity classes we consider in this paper, e.g., **ZK**, **SZK**, and **IP**, are taken to be classes of promise problems. See section 2.3.

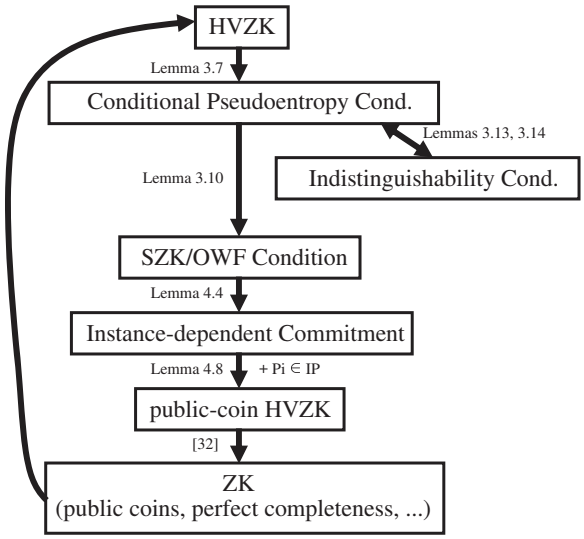


FIG. 1. *Steps of our proof.*

of the known complete problems for **SZK** [52, 34] and are described in more detail below. The reductions from **HVZK** to these characterizations are natural adaptations of the reductions from **HVSZK** = **SZK** to the **SZK**-complete problems (which in turn are based on the simulator analyses of [18, 1, 51]). We then show that every problem satisfying the CONDITIONAL PSEUDOENTROPY CONDITION also satisfies the SZK/OWF CONDITION. This step utilizes the techniques of Håstad et al. [37] to construct the needed one-way functions f_x.

In proving the reverse direction of Theorem 1.2, we first show that every problem satisfying the SZK/OWF CONDITION has a certain kind of "instance-dependent commitment scheme" [40] as discussed in more detail below. We then use the techniques of [29, 39, 7, 40] to show that every problem in **IP** with such a commitment scheme has a public-coin honest-verifier zero-knowledge proof. The honest-verifier zero-knowledge proof is then converted into one that is zero knowledge even for cheating verifiers using the compiler of [32]. The resulting proof system remains public coin, and also has additional nice properties such as perfect completeness and black-box simulation.

Putting these steps together, we deduce that membership in **HVZK**, membership in **ZK** (even with additional nice properties), the SZK/OWF CONDITION, the CONDITIONAL PSEUDOENTROPY CONDITION, the INDISTINGUISHABILITY CONDITION, and having an instance-dependent commitment scheme are all equivalent (for problems in **IP**). The latter three characterizations of **ZK** are of interest beyond their role in establishing Theorem 1.2, so we describe them in more detail below.

1.3. Additional characterizations of ZK.

Computational analogues of the **SZK***-complete problems.* Recall that in [52, 34], it was demonstrated that **SZK** has two natural complete problems, STATISTICAL DIFFERENCE and ENTROPY DIFFERENCE. These problems proved to be very useful tools in the study of **SZK** (cf. [52, 57]) because they reduced the study of the entire class to the study of these specific problems.

In this work, we provide characterizations of **ZK** that are natural computational analogues of these two problems. For example, recall that instances of the STATIS-

TICAL DIFFERENCE problem consist of pairs (X, Y) of probability distributions on strings, specified by circuits that sample from them. The YES instances are pairs that are statistically close and the NO instances are pairs that are statistically far apart. We show that if "statistically close" is replaced with "computationally indistinguishable," the resulting condition characterizes **ZK**. This condition, which we refer to as the INDISTINGUISHABILITY CONDITION, cannot be cast as a complete problem because there can be distributions that are both computationally indistinguishable and statistically far apart. Rather, we say that a promise problem satisfies the INDISTINGUISHABILITY CONDITION if its instances can be efficiently mapped to pairs (X, Y) that are computationally indistinguishable or statistically far apart, according to whether the instance is a YES or NO instance. We show that this characterizes **ZK** in the sense that a promise problem is in **ZK** if and only if it is in **IP** and satisfies the INDISTINGUISHABILITY CONDITION.

The computational analogue of ENTROPY DIFFERENCE is less immediate, and in fact a crucial step towards our establishment of the SZK/OWF characterization theorem was the realization that the "right" problem to generalize is a variant of ENTROPY DIFFERENCE, which we call CONDITIONAL ENTROPY APPROXIMATION rather than ENTROPY DIFFERENCE itself. (See section 3 for more details.)

Instance-dependent commitments. A fundamental tool in the construction of many zero-knowledge proofs is that of a *commitment scheme*. This is a protocol whereby a *sender* can "commit" to a bit b in such a way that the *receiver* learns nothing about b (the scheme is *hiding*), but nevertheless the sender cannot "open" the commitment to a value other than b (the scheme is *binding*). Commitment schemes, which can be constructed from any one-way function [45, 37], play an essential role in the construction of zero-knowledge proofs for all of NP and **IP** [29, 39, 7]. Some evidence that commitments are necessary for zero knowledge came from the work of Damgård [12, 13], who focused on 3-round public-coin zero-knowledge proofs, and Ostrovsky [49] and Ostrovsky and Wigderson [50], who showed that zero-knowledge proofs for *hard-on-average* languages imply one-way functions (and hence standard commitment schemes [45, 37]).

In this work, we show an *equivalence* between zero-knowledge proofs and certain types of commitment schemes, which we now describe. In an *instance-dependent* commitment scheme [6, 40, 44] for a promise problem Π, both the sender and receiver get a common auxiliary input x, which is an instance of Π. It is required that if x is a YES instance of Π, then the scheme is hiding, and if x is a NO instance, then the scheme is binding. Thus, instance-dependent commitment schemes are a relaxation of commitment schemes because the hiding and binding properties are not required to hold at the same time. Nevertheless, this relaxation is still useful in constructing zero-knowledge proofs. The reason is that zero-knowledge proofs based on commitments (see, e.g., [29, 39, 7]) typically use only the hiding property in proving zero knowledge (which is required only when x is a YES instance) and use only the binding property in proving soundness (which is required only when x is a NO instance).

We show that a promise problem is in **ZK** (resp., **SZK**) if and only if it has an instance-dependent commitment scheme that is computationally (resp., statistically) hiding on YES instances (and statistically binding on NO instances). Indeed, the most technical part of this paper is the construction of instance-dependent commitment schemes for all of **SZK**, which utilizes much of the machinery previously developed in the study of **SZK** [48, 52, 34]. The construction of instance-dependent commitments for **ZK** then follows using the SZK/OWF characterization theorem and the known construction of commitment schemes from one-way functions [45, 37].

Two deficiencies in our instance-dependent commitments are that the hiding property holds only against an honest receiver (i.e., one that follows the specified protocol) and that the sender of the commitment scheme is not polynomial time, but rather $\mathbf{BPP}^{\mathbf{NP}}$. The effect of these are that the direct constructions of zero-knowledge proofs that we obtain using the commitments are only honest-verifier zero knowledge and have provers that require an **NP** oracle (rather than just an **NP** witness, as would be preferable for problems in **NP**). The honest-verifier constraint is removed using the compiler of [32], which converts *public-coin* honest-verifier zero-knowledge proofs into general zero-knowledge proofs. The **NP** oracle has been removed in subsequent work [46] by using a new, more relaxed, type of instance-dependent commitment scheme; see section 8.

1.4. Organization. We begin in section 2 with the definitions, notation, and basic results that we will use throughout the paper, in particular covering probability and information theory, promise problems, and zero-knowledge proofs. Section 3 contains the proof of the forward direction of Theorem 1.2, including establishing the computational analogues of the **SZK**-complete problems. Section 4 contains the proof of the reverse direction of Theorem 1.2, except for the construction of instance-dependent commitments for all of **SZK**, which is deferred to section 5. Section 6 ties together the results of sections 3–5, in particular establishing Theorem 1.2. Section 7 contains several applications and extensions of our results, including monotone closure properties of **ZK**, new proofs of the Ostrovsky–Wigderson theorems [50], and an equivalence between strict and expected polynomial-time simulators. In section 8, we conclude with some open problems and directions for further work.

2. Preliminaries.

2.1. Basic notation. If X is a random variable taking values in a finite set $\mathcal{U}$, then we write $x \leftarrow X$ to indicate that x is selected according to X. If S is a subset of $\mathcal{U}$, then $x \leftarrow S$ means that x is selected according to the uniform distribution on S. We adopt the convention that when the same random variable occurs several times in an expression, all occurrences refer to a single sample. For example, $\Pr[f(X) = X]$ is defined to be the probability that when $x \leftarrow X$, we have $f(x) = x$. We write U_n to denote the random variable distributed uniformly over $\{0,1\}^n$. The *support* of a random variable X is $\mathrm{Supp}(X) = \{x : \Pr[X = x] > 0\}$. A random variable is *flat* if it is uniform over its support. If X and Y are random variables, then $X \otimes Y$ denotes the random variable obtained by taking independent random samples $x \leftarrow X$ and $y \leftarrow Y$ and outputting the pair (x, y). We write $\otimes^k X$ to denote the random variable consisting of k independent copies of X. For an event E, $X|_E$ denotes the random variable X conditioned on E.

A function $\mu : \mathbb{N} \to [0,1]$ is called *negligible* if $\mu(n) = n^{-\omega(1)}$. We let $\mathrm{neg}(n)$ denote an arbitrary negligible function (i.e., when we say that $f(n) < \mathrm{neg}(n)$ we mean that *there exists* a negligible function $\mu(n)$ such that for every n, $f(n) < \mu(n)$). Likewise, $\mathrm{poly}(n)$ denotes an arbitrary function $f(n) = n^{O(1)}$.

For a probabilistic algorithm A, we write $A(x; r)$ to denote the output of A on input x and coin tosses r. In this case, $A(x)$ is a random variable representing the output of A for uniformly selected coin tosses. *PPT* refers to probabilistic algorithms (i.e., Turing machines) that run in *strict* polynomial time. A *nonuniform* PPT algorithm is a pair $(A, \bar{z})$, where $\bar{z} = z_1, z_2, \ldots$ is an infinite sequence of strings in which $|z_n| = \mathrm{poly}(n)$, and A is a PPT algorithm that receives pairs of inputs of the form $(x, z_{|x|})$. (The string z_n is called the *advice string* for A for inputs of length n.) Non-

uniform PPT algorithms are equivalent to (nonuniform) families of polynomial-sized Boolean circuits.

A Boolean circuit $C : \{0,1\}^m \to \{0,1\}^n$ defines a probability distribution on $\{0,1\}^n$ by evaluating C on a uniformly chosen input in $\{0,1\}^m$. That is, we view C as specifying a sampling algorithm for the distribution, with C's input gates being the coin tosses; thus we will often refer to distributions specified by circuits as *samplable distributions*. This is a "nonuniform" notion of samplability, because the sampling algorithm C can be tailored to a particular output length n. Later, in Definition 2.11, we will consider a uniform notion of samplability, which refers to ensembles (i.e., sequences) of probability distributions that are generated by a uniform PPT algorithm. Samplable distributions will play a central role in the paper.

2.2. Statistical measures.

Statistical difference. The *statistical difference* (also known as the *variation distance*) between random variables X and Y taking values in $\mathcal{U}$ is defined to be

$$\begin{aligned}\Delta(X,Y) &\stackrel{\text{def}}{=} \max_{S\subset\mathcal{U}} |\Pr[X\in S] - \Pr[Y\in S]| \\ &= \frac{1}{2}\sum_{x\in\mathcal{U}} |\Pr[X=x] - \Pr[Y=x]| \\ &= 1 - \sum_{x\in\mathcal{U}} \min\{\Pr[X=x], \Pr[Y=x]\}.\end{aligned}$$

We say that X and Y are *ε-close* if $\Delta(X,Y)\le\varepsilon$. For basic facts about this metric, see [52, sect. 2.3].

Entropy. The *entropy* of a random variable X is $\mathrm{H}(X) = \mathrm{E}_{x\leftarrow X}[\log(1/\Pr[X=x])]$, where here and throughout the paper all logarithms are to base 2. Intuitively, $\mathrm{H}(X)$ measures the amount of randomness in X *on average* (in bits). The *min-entropy* of X is $\mathrm{H}_\infty(X) = \min_x[\log(1/\Pr[X=x])]$; this is a "worst-case" measure of randomness. In general $\mathrm{H}_\infty(X)\le\mathrm{H}(X)$, but if X is flat, then $\mathrm{H}(X) = \mathrm{H}_\infty(X) = \log|\mathrm{Supp}(X)|$. For $p\in[0,1]$, we define the *binary entropy function* $\mathrm{H}_2(p)$ to be the entropy of a binary random variable that is 1 with probability p and 0 with probability $1-p$, i.e., $\mathrm{H}_2(p) = p\cdot\log(1/p) + (1-p)\cdot\log(1/(1-p))$. For jointly distributed random variables X and Y, we define the *conditional entropy of X given Y* to be

$$\mathrm{H}(X|Y) \stackrel{\text{def}}{=} \mathop{\mathrm{E}}_{y\leftarrow Y}[\mathrm{H}(X|_{Y=y})] = \mathop{\mathrm{E}}_{(x,y)\leftarrow(X,Y)}\left[\log\frac{1}{\Pr[X=x|Y=y]}\right] = \mathrm{H}(X,Y) - \mathrm{H}(Y).$$

A useful fact is that if two random variables are statistically close, then their entropies must also be close, as follows.

LEMMA 2.1 (cf. [34, Fact B.1]). *For any two random variables, X and Y, ranging over a universe $\mathcal{U}$, if $\delta = \Delta(X,Y)$, then*

$$|\mathrm{H}(X) - \mathrm{H}(Y)| \le \log(|\mathcal{U}|-1)\cdot\delta \;+\; \mathrm{H}_2(\delta).$$

Another useful fact is that random variables taking values in a universe $\mathcal{U}$ can be modified so that they do not assign any elements in their support of a probability mass much smaller than $1/|\mathcal{U}|$ without incurring a significant statistical difference or change in entropy.

LEMMA 2.2. *Let (X,Y) be a random variable taking values in $\mathcal{U}\times\mathcal{V}$. Then for any $\delta>0$, there is a random variable (X',Y') that is δ-close to (X,Y) and satisfies*

$\Pr[X' = x | Y' = y] \geq \delta/|\mathcal{U}|$ *for all* $(x, y) \in \mathrm{Supp}(X', Y')$. *Moreover,*

$$|\mathrm{H}(X'|Y') - \mathrm{H}(X|Y)| \leq \log(|\mathcal{U}| - 1) \cdot \delta + \mathrm{H}_2(\delta).$$

Proof. For each $y \in \mathrm{Supp}(Y)$, the set

$$S_y = \{x \in \mathcal{U} : \Pr[X = x | Y = y] < \delta/|\mathcal{U}|\}$$

has total probability mass less than δ under the conditional distribution $X|_{Y=y}$. Thus shifting the probability mass of the points in S_y to other points in $\mathcal{U}$ yields a random variable Z_y that is δ-close to $X|_{Y=y}$. By Lemma 2.1, for every y, the entropy of Z_y differs from that of $X|_{Y=y}$ by at most $\delta' = \log(|\mathcal{U}| - 1) \cdot \delta + \mathrm{H}_2(\delta)$. Thus taking $(X', Y') = (Z_Y, Y)$ satisfies the conclusion of the lemma. ∎

For more background on entropy, see [11].

Direct products. We will often refer to the behavior of the above measures under direct products, i.e., when we take k independent copies of a random variable. For statistical difference, we have the following bounds.

LEMMA 2.3 (cf. [52, Lemma 3.4]). *For random variables X and Y, if $\delta = \Delta(X, Y)$, then for every $k \in \mathbb{N}$, we have*

$$1 - 2\exp(-k\delta^2/2) \leq \Delta(\otimes^k X, \otimes^k Y) \leq k\delta.$$

For entropy, it holds that for every X, Y, $\mathrm{H}(X \otimes Y) = \mathrm{H}(X) + \mathrm{H}(Y)$, and thus $\mathrm{H}(\otimes^k X) = k \cdot \mathrm{H}(X)$. Similarly, for conditional entropy, if we write $\otimes^k(X, Y) = ((X_1, Y_1), \ldots, (X_k, Y_k))$, then $\mathrm{H}((X_1, \ldots, X_k)|(Y_1, \ldots, Y_k)) = k \cdot \mathrm{H}(X|Y)$.

Another well-known and useful feature of taking direct products is that it "flattens" random variables so that probability masses become concentrated around $2^{-\mathrm{H}(X)}$. (This is known as the asymptotic equipartition property in information theory; see [11].) Our formalization of it follows [34], with an extension to conditional distributions.

DEFINITION 2.4 (heavy, light, and typical elements). *Let X be a random variable taking values in a universe $\mathcal{U}$, and let x be an element of $\mathcal{U}$. For a positive real number Φ, we say that x is* Φ-heavy *(resp.,* Φ-light*) if* $\Pr[X = x] \geq 2^{\Phi} \cdot 2^{-\mathrm{H}(X)}$ *(resp.,* $\Pr[X = x] \leq 2^{-\Phi} \cdot 2^{-\mathrm{H}(X)}$*). Otherwise, we say that x is* Φ-typical.

If Y is a random variable jointly distributed with X, and $y \in \mathrm{Supp}(Y)$, we say that x is Φ-heavy given y *(resp.,* Φ-light given y*) if* $\Pr[X = x | Y = y] \geq 2^{\Phi} \cdot 2^{-\mathrm{H}(X|Y)}$ *(resp., if* $\Pr[X = x | Y = y] \leq 2^{-\Phi} \cdot 2^{-\mathrm{H}(X|Y)}$*). Otherwise, we say that x is* Φ-typical given y.

A natural relaxed definition of flatness follows. The definition links the amount of slackness allowed in "typical" elements with the probability mass assigned to non-typical elements.

DEFINITION 2.5 (nearly flat distributions).[6] *A distribution X is called* Φ-flat *if for every $t \geq 1$ the probability that an element chosen from X is $t \cdot \Phi$-typical is at least* $1 - 2^{-t^2}$.

If Y is jointly distributed with X, then we say that X is Φ-flat given Y *if for every $t \geq 1$, when $(x, y) \leftarrow (X, Y)$, the probability that x is $t \cdot \Phi$-typical given y is at least* $1 - 2^{-t^2}$.

[6] The definition in [34] allows any $t > 0$, but requires only that the probability of being $t \cdot \Phi$-typical be $1 - 2^{-t^2+1}$. We find it more convenient to restrict to $t \geq 1$, a setting that is satisfied in all our applications.

A consequence of this definition is that if X is Φ-flat, then for every $t \geq 1$, the random variable X is (2^{-t^2})-close to a random variable X' with min-*entropy* at least $\mathrm{H}(X) - t\Phi$.

LEMMA 2.6 (flattening lemma). *Let X be a distribution, k be a positive integer, and $\otimes^k X$ denote the distribution composed of k independent copies of X. Suppose that for all x in the support of X it holds that $\Pr[X = x] \geq 2^{-m}$. Then $\otimes^k X$ is $\sqrt{k}\cdot m$-flat.*

Suppose Y is jointly distributed with X, and for all (x, y) in the support of (X, Y) it holds that $\Pr[X = x|Y = y] \geq 2^{-m}$. Then, defining the random variable $((X_1, Y_1), \ldots, (X_k, Y_k)) = \otimes^k(X, Y)$, the random variable $(X_1, \ldots, X_k)$ is $\sqrt{k} \cdot m$-flat given $(Y_1, \ldots, Y_k)$.

The key point is that deviation from flatness grows sublinearly with k, while the entropy grows linearly with k. We prove the flattening lemma in Appendix A for completeness.

Hashing. The topic of *randomness extraction* is concerned with efficiently extracting as many almost-uniform random bits as possible from random variables that are not uniformly distributed. The entropy of a random variable does not provide a good measure of how many almost-uniform bits can be extracted, but its min-entropy does, as long as we are willing to let the extraction procedure itself be probabilistic. Surveys of the large body of work on this topic can be found in [47, 53]. Much of that work focuses on minimizing the number of extra random bits used by the extractor; this is not a major concern for us, so we can use relatively simple randomness extractors. In particular, the leftover hash lemma of [8, 37] shows that universal (or pairwise-independent) hash functions can be used for this purpose.

LEMMA 2.7 (leftover hash lemma). *Let H be randomly selected from a family of universal hash functions mapping $\{0,1\}^n$ to $\{0,1\}^m$. Then, for every $\varepsilon > 0$ and every distribution X on $\{0,1\}^n$ of min-entropy at least $m + 2\log(1/\varepsilon)$, the random variable $(H, H(X))$ is ε-close to (H, U_m).*

Recall that for every n, m, there is an explicit family of universal hash functions mapping $\{0,1\}^n$ to $\{0,1\}^m$, where a random hash function in the family can be described by $O(n+m)$ random bits and can be evaluated in time $\mathrm{poly}(n, m)$ (cf. [22, sect. 3.6.1]).

2.3. Promise problems. Roughly speaking, a *promise problem* [17] is a decision problem in which some inputs are excluded. Formally, a promise problem is specified by two disjoint sets of strings $\Pi = (\Pi_Y, \Pi_N)$, where we call Π_Y the set of YES *instances* and Π_N the set of NO *instances*. Such a promise problem is associated with the following computational problem: Given an input that is "promised" to lie in $\Pi_Y \cup \Pi_N$, decide whether it is in Π_Y or in Π_N. Note that languages are special cases of promise problems (namely, a language L over an alphabet Σ corresponds to the promise problem $(L, \Sigma^* \setminus L)$). Thus, working with promise problems makes our results more general. Moreover, even for proving our results just for languages, it turns out to be extremely useful to work with promise problems along the way.

The *complement* of a promise problem $\Pi = (\Pi_Y, \Pi_N)$ is the promise problem $\overline{\Pi} = (\Pi_N, \Pi_Y)$. The *union* of two promise problems Π and Γ is the promise problem $\Pi \cup \Gamma = (\Pi_Y \cup \Gamma_Y,\ \Pi_N \cap \Gamma_N)$. The *intersection* of two promise problems Π and Γ is the promise problem $\Pi \cap \Gamma = (\Pi_Y \cap \Gamma_Y,\ \Pi_N \cup \Gamma_N)$.

Most complexity classes, typically defined as classes of languages, can be extended to promise problems in a natural way by translating conditions on inputs in the language into conditions on YES instances, and conditions on inputs not in the language into conditions on NO instances. For example, a promise problem Π is in

BPP if there is a PPT algorithm A such that $x \in \Pi_Y \Rightarrow \Pr[A(x) = 1] \geq 2/3$ and $x \in \Pi_N \Rightarrow \Pr[A(x) = 0] \leq 1/3$. All complexity classes in this paper denote classes of promise problems.

A promise problem Π *reduces* to promise problem Γ if there is a polynomial-time computable function f such that

$$x \in \Pi_Y \Rightarrow f(x) \in \Gamma_Y,$$
$$x \in \Pi_N \Rightarrow f(x) \in \Gamma_N.$$

That is, we work with polynomial-time mapping reductions (i.e., Karp reductions) unless otherwise specified. If $\mathbf{C}$ is a class of promise problems, then Π is *complete for* $\mathbf{C}$ (or $\mathbf{C}$-*complete*) if $\Pi \in \mathbf{C}$ and every promise problem in $\mathbf{C}$ reduces to Π. Sometimes we will restrict our attention to reductions f that are *nonshrinking*, in the sense that there is a constant $\delta > 0$ such that $|f(x)| \geq |x|^\delta$ for all strings x.

We refer the reader to the recent survey of Goldreich [24] for more on the utility and subtleties of promise problems.

2.4. Auxiliary-input cryptographic primitives. It will be very useful for us to work with cryptographic primitives that are parameterized by an additional "auxiliary" input x, and where the security condition will hold only if x is in some particular set I. Indeed, recall that the SZK/OWF Condition refers to such a variant of the notion of one-way functions (as captured in the definitions below). Auxiliary-input primitives have been considered in the past, such as in the instance-dependent commitments of [40] (which we consider in section 4.1) and the auxiliary-input one-way functions of [50] (which are weaker than our formulation below). In this section, we provide a general framework for discussing and relating such primitives.

Definition 2.8. *An* auxiliary-input function ensemble *is a collection of functions* $\mathcal{F} = \{f_x : \{0,1\}^{p(|x|)} \to \{0,1\}^{q(|x|)}\}_{x \in \{0,1\}^*}$, *where* p *and* q *are polynomials. We call* $\mathcal{F}$ polynomial-time computable *(or just* poly-time*) if there is a (deterministic) polynomial-time algorithm* F *such that for every* $x \in \{0,1\}^*$ *and* $y \in \{0,1\}^{p(|x|)}$, *we have* $F(x,y) = f_x(y)$.

Definition 2.9. *An* auxiliary-input one-way function on I *is a poly-time auxiliary-input function ensemble* $\mathcal{F} = \{f_x : \{0,1\}^{p(|x|)} \to \{0,1\}^{q(|x|)}\}$ *such that for every nonuniform PPT* A, *there exists a negligible function* μ *such that for all* $x \in I$,

$$\Pr\left[A(x, f_x(U_{p(|x|)})) \in f_x^{-1}(f_x(U_{p(|x|)}))\right] \leq \mu(|x|).$$

(We note that since A is nonuniform, it is not essential that we give it the input x, because x can be hardwired in as advice, but the definition seems more natural, as above.) The standard definition of a one-way function is obtained by considering $I = \{1^n : n \geq 0\}$ and $p(n) = n$. The above is a stronger notion of an auxiliary-input one-way function than the notion considered by Ostrovsky and Wigderson [50]. In their formulation (which they denote by $\exists 1WF$), the set I is not fixed for all A, but rather can depend on A. That is, they require that for every PPT A, there must exist an infinite set I_A such that A has small probability of inverting f_x for all $x \in I_A$. (See our Theorem 7.1 for a precise formulation of this notion and the result of [50].)

Given the above definition, we can restate the SZK/OWF Condition as follows.

Definition 2.10. *A promise problem* $\Pi = (\Pi_Y, \Pi_N)$ *satisfies the* SZK/OWF Condition *if there is* $I \subseteq \Pi_Y$ *such that*

- *the promise problem* $\Pi' = (\Pi_Y \setminus I, \Pi_N)$ *is in* **SZK**.
- *there exists an auxiliary-input one-way function on* I.

Similarly, the notion of computational indistinguishability has an auxiliary-input analogue (which is widely used in the definition of zero knowledge; see section 2.5).

DEFINITION 2.11. *An* auxiliary-input probability ensemble *is a collection of random variables* $\{X_x\}_{x\in\{0,1\}^*}$*, where* X_x *takes values in* $\{0,1\}^{p(|x|)}$ *for some polynomial* p*. We call such an ensemble* samplable *if there is a PPT algorithm* M *such that for every* x*, the output* $M(x)$ *is distributed according to* X_x*.*

DEFINITION 2.12. *Two auxiliary-input probability ensembles* $\{X_x\}$ *and* $\{Y_x\}$ *are* computationally indistinguishable *on* $I \subseteq \{0,1\}^*$ *if for every nonuniform PPT* D*, there exists a negligible function* μ *such that for all* $x \in I$*,*

$$|\Pr[D(x, X_x) = 1] - \Pr[D(x, Y_x) = 1]| \leq \mu(|x|).$$

Similarly, we say that $\{X_x\}$ *and* $\{Y_x\}$ *are* statistically indistinguishable *on* $I \subseteq \{0,1\}^*$ *if the above is required for all functions* D *(instead of only nonuniform PPT* D*'s). Equivalently,* $\{X_x\}$ *and* $\{Y_x\}$ *are statistically indistinguishable on* I *if and only if* X_x *and* Y_x *are* $\mu(|x|)$*-close for some negligible function* μ *and all* $x \in I$*. If* X_x *are* Y_x *are identically distributed for all* $x \in I$ *(i.e.,* $\mu = 0$*), we say that they are* perfectly indistinguishable.

Often, we will informally say "X_x and Y_x are computationally indistinguishable when $x \in I$" to mean that the ensembles $\{X_x\}$ and $\{Y_x\}$ are computationally indistinguishable on I. It is well known that indistinguishability is preserved when we take polynomially many direct products. See the following lemma.

LEMMA 2.13 (cf. [23, Chap. 3, Ex. 9]). *If* $\{X_x\}$ *and* $\{Y_x\}$ *are computationally indistinguishable on* I*, then for every polynomial* p*,* $\{\otimes^{p(|x|)}X_x\}$ *and* $\{\otimes^{p(|x|)}Y_x\}$ *are computationally indistinguishable on* I*.*

Now we can naturally define auxiliary-input pseudorandom generators.

DEFINITION 2.14. *An* auxiliary-input pseudorandom generator on I *is a polytime auxiliary-input function ensemble* $\mathcal{G} = \{G_x : \{0,1\}^{p(|x|)} \to \{0,1\}^{q(|x|)}\}$ *such that* $q(n) > p(n)$*, and the probability ensembles* $\{G_x(U_{p(|x|)})\}_x$ *and* $\{U_{q(|x|)}\}_x$ *are computationally indistinguishable on* I*.*

Almost all reductions between cryptographic primitives immediately extend to their auxiliary-input analogues (using the same proof). One example is the equivalence between the existence of pseudorandom generators and the existence of one-way functions.

THEOREM 2.15 (see [37]). *For every set* $I \subseteq \{0,1\}^*$*, there exists a pseudorandom generator on* I *if and only if there exists a one-way function on* I*.*

2.5. Zero-knowledge proofs. An *interactive protocol* (A, B) consists of two algorithms that compute the *next-message function* of the (honest) parties in the protocol. Specifically, $A(x, a, \alpha_1, \ldots, \alpha_k; r)$ denotes the next message α_{k+1} sent by party A when the common input is x, A's auxiliary input is a, A's coin tosses are r, and the messages exchanged so far are $\alpha_1, \ldots, \alpha_k$. There are two special messages, `accept` and `reject`, which immediately halt the interaction. We say that party A (resp., B) is *probabilistic polynomial time (PPT)* if its next-message function can be computed in polynomial time (in $|x| + |a| + |\alpha_1| + \cdots + |\alpha_k|$). Sometimes (though not in this section) we will refer to protocols with a joint output; such an output is specified by a deterministic, polynomial-time computable function of the messages exchanged.

For an interactive protocol (A, B), we write $(A(a), B(b))(x)$ to denote the random process obtained by having A and B interact on common input x, with (private) auxiliary inputs a and b to A and B, respectively (if any), and with independent

random coin tosses for A and B. We call (A, B) *polynomially bounded* if there is a polynomial p such that for all x, a, b, the total length of all messages exchanged in $(A(a), B(b))(x)$ is at most $p(|x|)$ with probability 1. Moreover, if B^* is any interactive algorithm, then A will immediately halt and reject in $(A(a), B^*(b))(x)$ if the total length of the messages ever exceeds $p(|x|)$; we have the analogous requirement for B interacting with any A^*.

The number of *rounds* in an execution of the protocol is the *total* number of messages exchanged between A and B, not including the final `accept`/`reject` message. We call the protocol (A, B) *public coin* if all the messages sent by B are simply the output of its coin tosses (independent of the history), except for the final `accept`/`reject` message, which is computed as a deterministic function of the transcript. (Such protocols are also sometimes known as *Arthur–Merlin games* [2].)

DEFINITION 2.16. *An interactive protocol (P, V) is an* interactive proof system *for a promise problem Π if there are functions $c, s : \mathbb{N} \to [0, 1]$ such that $1 - c(n) > s(n) + 1/\mathrm{poly}(n)$ and the following hold:*

- *(Efficiency) (P, V) is polynomially bounded, and V is computable in PPT.*
- *(Completeness) If $x \in \Pi_Y$, then V accepts in $(P, V)(x)$ with probability at least $1 - c(|x|)$.*
- *(Soundness) If $x \in \Pi_N$, then for every P^*, V accepts in $(P^*, V)(x)$ with probability at most $s(|x|)$.*

We call $c(\cdot)$ the completeness error *and $s(\cdot)$ the* soundness error. *We say that (P, V) has* negligible error *if both c and s are negligible. We say that it has* perfect completeness *if $c = 0$. We denote by* **IP** *the class of promise problems possessing interactive proof systems. We denote by* **AM** *the class of promise problems possessing two-round, public-coin interactive proof systems.*

AM is known to equal the class of promise problems possessing constant-round (possibly private-coin) interactive proof systems [36, 2].

We write $\langle A(a), B(b)\rangle(x)$ to denote B's view of the interaction, i.e., a transcript $(\gamma_1, \gamma_2, \ldots, \gamma_t; r)$, where the γ_i's are all the messages exchanged and r is B's coin tosses.

There are various notions of zero knowledge arising from different choices for the class of verifiers to be considered. The weakest is to consider only the "honest verifier," the one that follows the specified protocol.[7]

DEFINITION 2.17 (honest-verifier zero knowledge). *An interactive proof system (P, V) for a promise problem Π is* (perfect/statistical/computational) honest-verifier zero knowledge *if there exists a PPT* simulator S *such that the ensembles $\{\langle P, V\rangle(x)\}$ and $\{S(x)\}$ are (perfectly/statistically/computationally) indistinguishable on Π_Y.*[8] *We will often omit the word "computational" in reference to computational zero knowledge.*

HVPZK, **HVSZK**, *and* **HVZK** *denote the classes of promise problems that have honest-verifier perfect, statistical, and computational zero-knowledge proofs, respectively.*

While honest-verifier zero knowledge is already a nontrivial and interesting notion, cryptographic applications usually require that the zero-knowledge condition

[7] This is an instantiation of what is called an "honest-but-curious adversary" or "passive adversary" in the literature on cryptographic protocols.

[8] Actually, in the case of perfect zero knowledge, it is common to allow the simulator to output a failure symbol $\perp$ with some small probability (say, at most 1/2) and require that the output of $S(x)$ conditioned on nonfailure be identical to $\langle P, V\rangle(x)$ (cf. [23, Def. 4.3.1]). However, we define perfect zero knowledge only for the sake of context and will not use it anywhere else in the paper.

hold even if the verifier deviates arbitrarily from the specified protocol. This idea is captured by the following definition.

DEFINITION 2.18 (auxiliary-input zero knowledge).[9] *An interactive proof system* (P, V) *for a promise problem* Π *is (perfect/statistical/computational)* (auxiliary-input) zero knowledge *if for every PPT* V^* *and polynomial* p*, there exists a PPT* S *such that the ensembles*

$$\{\langle P, V^*(z)\rangle(x)\} \qquad \text{and} \qquad \{S(x,z)\} \tag{1}$$

are (perfectly/statistically/computationally) indistinguishable on the set

$$\{(x,z) : x \in \Pi_Y,\ |z| = p(|x|)\}.$$

PZK, **SZK**, *and* **ZK** *are the classes of promise problems possessing perfect, statistical, and computational (auxiliary-input) zero-knowledge proofs, respectively.*

The auxiliary input z in the above definition allows one to model a priori information that the verifier may possess before the interaction begins, such as from earlier steps in a larger protocol in which the zero-knowledge proof is being used or from prior executions of the same zero-knowledge proof. As a result, auxiliary-input zero knowledge is closed under sequential composition. That is, if an auxiliary-input zero-knowledge proof is repeated sequentially polynomially many times, then it remains auxiliary-input zero knowledge [30]. Plain zero knowledge (i.e., without auxiliary inputs) is not closed under sequential composition [27], and thus auxiliary-input zero knowledge is the definition typically used in the literature.

Typically, a protocol is proven to be zero knowledge by actually exhibiting a single, universal simulator that simulates an arbitrary verifier strategy V^* by using V^* as a subroutine. That is, the simulator does not depend on or use the code of V^* (or its auxiliary input) and instead requires only black-box access to V^*. This type of simulation is formalized as follows.

DEFINITION 2.19 (black-box zero knowledge). *An interactive proof system* (P, V) *for a promise problem* Π *is (perfect/statistical/computational)* black-box zero knowledge *if there exists an oracle PPT* S *such that for every nonuniform PPT* V^**, the ensembles*

$$\{\langle P, V^*\rangle(x)\}_{x\in\Pi_Y} \qquad \text{and} \qquad \{S^{V^*(x,\cdot,\cdot)}(x)\}_{x\in\Pi_Y}$$

are (perfectly/statistically/computationally) indistinguishable.

Even though the above definition does not explicitly refer to an auxiliary input, the definition encompasses auxiliary-input zero knowledge because we allow V^* to be nonuniform (and thus the auxiliary input can be hardwired in V^* as advice). The recent work of Barak [3] demonstrated that non–black-box zero-knowledge proofs can achieve properties (such as simultaneously being public coin, having a constant number of rounds, and having negligible error) that were known to be impossible for black-box zero knowledge [27]. Nevertheless, our results will show that, when ignoring

[9] Our formulation of auxiliary-input zero knowledge is slightly different than, but equivalent to, the definition in the textbook [23]. We allow V^* to run in polynomial time in the lengths of both its input x and its auxiliary input z, but put a polynomial bound on the length of the auxiliary input. In [23, sect. 4.3.3], V^* is restricted to run in time that is polynomial in just the length of the input x, and no bound is imposed on the length of the auxiliary input z (so V^* may only be able to read a prefix of z). The purpose of allowing the auxiliary input to be longer than the running time of z is to provide additional nonuniformity to the distinguisher (beyond that which the verifier has); we do this directly by allowing the distinguisher to be nonuniform in Definition 2.12.

efficiency considerations, black-box zero knowledge is as rich as standard, auxiliary-input zero knowledge; that is, every problem in **ZK** has a black-box zero-knowledge proof system.

Remarks on the definitions. Our definitions mostly follow the now standard definitions of zero-knowledge proofs as presented in [23], but we highlight the following points:

1. (Promise problems) As has been done numerous times before (see, e.g., [28, 52]), we extended all of the definitions to promise problems $\Pi = (\Pi_Y, \Pi_N)$ in the natural way; i.e., conditions previously required for inputs in the language (e.g., completeness and zero knowledge) are now required for all YES instances, and conditions previously required for inputs not in the language (e.g., soundness) are now required for all NO instances. Similarly, all our complexity classes (e.g., **ZK**, **SZK**, **HVZK**, **HVSZK**, **BPP**) are classes of promise problems. These extensions to promise problems are essential for formalizing our arguments, but all the final characterizations and results we derive about **ZK** automatically hold for the corresponding class of languages, simply because languages are special cases of promise problems.
2. (Nonuniform formulation) As has become standard, we have adopted a nonuniform formulation of zero knowledge, where the computational indistinguishability has to hold even with respect to nonuniform distinguishers and is universally quantified over all YES instances. Uniform treatments of zero knowledge are possible (see [21] and [4, Apdx. A]), but the definitions are much more cumbersome. We do not know whether analogues of our results hold for uniform zero knowledge, and thus we leave that as a problem for future work.
3. (Strict polynomial-time simulators) Following [23], we initially restrict our attention to zero knowledge with respect to simulators that run in *strict* polynomial time. The original definition of zero knowledge [35] allows for simulators that run in *expected* polynomial time. In section 7.3, we extend our techniques to zero knowledge with respect to expected polynomial-time simulators (in fact, an even weaker notion) and ultimately prove that the class of problems having zero-knowledge proofs with expected polynomial-time simulators and the class of problems having zero-knowledge proofs with strict polynomial-time simulators are equal.
4. (Proof systems versus arguments) We restrict our attention to the original notion of interactive *proof* systems [35, 2], where the soundness condition holds even for computationally unbounded prover strategies. A direction for future work is to consider the more relaxed notion of interactive *argument* systems [10], where the soundness condition is required only for polynomial-time prover strategies.
5. (Security parameterization) In the definition of computational indistinguishability (Definition 2.12), and consequently in the formulation of zero knowledge, computational indistinguishability is measured in terms of the input length, $|x|$. That is, only "long" statements can be proven with "high" security. An alternative, and perhaps more natural, formulation of zero knowledge (see [57, sect. 2.3]) measures indistinguishability in terms of a separate security parameter k, given to the prover, verifier, and simulator, and such that the protocol is allowed running time $\mathrm{poly}(|x|, k)$. We stick with the formulation in terms of the input length $|x|$ because it is the original and more commonly used definition. However, all of our results can be

extended to the security-parameterized definition via the observation that a security-parameterized zero-knowledge proof for a promise problem Π is equivalent to a (standard, non–security-parameterized) zero-knowledge proof for its "padded" version Π' defined by $\Pi'_Y = \{(x, 1^k) : x \in \Pi_Y,\ k \in \mathbb{N}\}$ and $\Pi'_N = \{(x, 1^k) : x \in \Pi_N,\ k \in \mathbb{N}\}$. For example, our result that $\mathbf{HVZK} = \mathbf{ZK}$ implies that the security-parameterized versions of these classes are also equal: for any promise problem Π having a security-parameterized honest-verifier zero-knowledge proof, we have $\Pi' \in \mathbf{HVZK} = \mathbf{ZK}$, which implies that Π has a security-parameterized (cheating-verifier) zero-knowledge proof. Note that we are not claiming that every problem in **ZK** has a security-parameterized zero-knowledge proof. (In contrast, it was shown in [52] that every problem in **SZK** has a security-parameterized statistical zero-knowledge proof.)

6. (Closure under reductions) All of the zero-knowledge classes defined above, in particular **HVZK** and **ZK**, are easily seen to be closed under *nonshrinking* reductions f (i.e., ones where $|f(x)| \geq |x|^{\Omega(1)}$): If f reduces Π to $\Gamma \in \mathbf{ZK}$, we can obtain a zero-knowledge proof for Π by having the prover and verifier on input x execute the zero-knowledge proof for Γ on $f(x)$. The nonshrinking condition is needed because the security of the zero-knowledge proof for Γ is measured as a function of $|f(x)|$, and we need to relate it to security in terms of $|x|$. The nonshrinking condition is unnecessary if one works with a security-parameterized definition of zero-knowledge proofs, as in item 5 above (cf. [57, Prop. 2.4.1]).

3. From HVZK to the SZK/OWF Condition. In this section, we prove that every problem in **HVZK** satisfies the SZK/OWF Condition.

A first attempt. To show that every $\Pi \in \mathbf{HVZK}$ satisfies the SZK/OWF Condition, it is tempting to take the following approach. Consider the (honest-verifier) simulator for Π's computational zero-knowledge proof system. Let I be the set of inputs $x \in \Pi_Y$ for which the simulator's output is statistically far from the verifier's view. When we ignore the inputs in I, we have an (honest-verifier) statistical zero-knowledge proof system. On inputs in I, the output of the simulator and the verifier's view are statistically far apart but computationally indistinguishable. Goldreich [20] has shown that from any two samplable distributions that are statistically far apart but computationally indistinguishable, we can construct a one-way function.

This approach has two difficulties:

- What threshold of statistical difference should we use to partition the inputs in Π_Y? The result of Goldreich requires a statistical difference of at least $1/p(n)$ for any fixed polynomial $p(n)$, but the definition of statistical zero knowledge requires negligible statistical difference $1/n^{\omega(1)}$.
- The result of Goldreich [20] requires that both distributions be (polynomial-time) *samplable*, but the verifier's view of the real interaction with the prover will typically not be samplable. Moreover, if we require only one of the two distributions in Goldreich's hypothesis to be samplable, then it is unlikely to imply one-way functions. Indeed, it has been proven unconditionally that the uniform distribution on $\{0,1\}^n$ (which is trivially samplable) is computationally indistinguishable from some (nonsamplable) distributions that are statistically very far from uniform (indeed have entropy polylog(n)) [26].

The first difficulty can be overcome using known results about **SZK**. Specifically, in [34] it is shown that if a problem Π has an interactive proof system that can be simulated to within a statistical difference of $1/p(n)$ for a sufficiently large (but fixed)

polynomial p (e.g., the cube of the communication complexity), then $\Pi \in \mathbf{SZK}$.

For the second difficulty, we use the fact that a samplable distribution that is computationally indistinguishable from an arbitrary (possibly nonsamplable) distribution of noticeably *higher entropy* does imply one-way functions [37]. This leads us to look for "high-entropy" distributions in the real prover-verifier interaction. We find such distributions using the techniques of [1, 51, 34]. This approach leads us to establish two other characterizations of **ZK** en route to the SZK/OWF Condition. These characterizations are computational analogues of the complete problems for **SZK**, and may be of independent interest.

3.1. Analogues of the SZK-complete problems. We establish two characterizations of **ZK** that are related to the complete problems for **SZK**, so we begin by recalling those.

The complete problems for **SZK**. The first problem is Statistical Difference, the promise problem $\mathrm{SD} = (\mathrm{SD}_Y, \mathrm{SD}_N)$ defined by

$$\mathrm{SD}_Y = \{(X,Y) : \Delta(X,Y) \leq 1/3\},$$
$$\mathrm{SD}_N = \{(X,Y) : \Delta(X,Y) \geq 2/3\},$$

where X and Y are samplable distributions specified by circuits that sample from them, and $\Delta(X,Y)$ denotes statistical difference. (See sections 2.1 and 2.2.)

The second problem is Entropy Difference, the promise problem $\mathrm{ED} = (\mathrm{ED}_Y, \mathrm{ED}_N)$ defined by

$$\mathrm{ED}_Y = \{(X,Y) : \mathrm{H}(X) \geq \mathrm{H}(Y) + 1\},$$
$$\mathrm{ED}_N = \{(X,Y) : \mathrm{H}(X) \leq \mathrm{H}(Y) - 1\},$$

where $\mathrm{H}(\cdot)$ denotes the entropy function (see section 2.2).

The completeness theorems of [52, 34] can be stated as follows.

Theorem 3.1 (see [52, 34]). Statistical Difference *and* Entropy Difference *are complete for* **SZK**. *That is, they are both in* **SZK**, *and for every problem* $\Pi \in \mathbf{SZK}$, *there is a polynomial-time computable function mapping strings* x *to pairs of samplable distributions* (X,Y) *such that*

- *if* $x \in \Pi_Y$, *then* $\Delta(X,Y) \leq 1/3$;
- *if* $x \in \Pi_N$, *then* $\Delta(X,Y) \geq 2/3$,

Analogous points hold for Entropy Difference.

Note that the result that **SZK** is closed under complement [48] follows from the fact that Entropy Difference trivially reduces to its complement (via the reduction $(X,Y) \mapsto (Y,X)$).

It turns out that, for obtaining **ZK** analogues of this completeness theorem, it is crucial that we do not work with Entropy Difference, but with a variant, Conditional Entropy Approximation (CEA), defined as follows:

$$\mathrm{CEA}_Y = \{((X,Y),r) : \mathrm{H}(X|Y) \geq r\},$$
$$\mathrm{CEA}_N = \{((X,Y),r) : \mathrm{H}(X|Y) \leq r - 1\}.$$

Here (X,Y) is a *samplable joint distribution* specified by two circuits that use the same coin tosses.

Proposition 3.2. Conditional Entropy Approximation *is complete for* **SZK**.

Proof. To show that CONDITIONAL ENTROPY APPROXIMATION is in **SZK**, we reduce it to ENTROPY DIFFERENCE. Given an instance $((X,Y),r)$ of CEA, we define $X' = \otimes^2(X,Y)$, $Y' = (\otimes^2 Y) \otimes U_{2r-1}$. Then

$$\mathrm{H}(X') - \mathrm{H}(Y') = 2 \cdot \mathrm{H}(X,Y) - (2 \cdot \mathrm{H}(Y) + (2r-1)) = 2 \cdot (\mathrm{H}(X|Y) - r) + 1.$$

It follows that $((X,Y),r) \in \mathrm{CEA}_Y \Rightarrow (X',Y') \in \mathrm{ED}_Y$ and $((X,Y),r) \in \mathrm{CEA}_N \Rightarrow (X',Y') \in \mathrm{ED}_N$.

To show that CEA is **SZK**-hard, we provide a reduction from SD to CEA, based on the reduction from SD to ED given in [57, sect. 4.4]. Given an instance (X_0, X_1) of SD, we construct the following samplable joint distribution:

$$(B,Y)\text{: Select } b \leftarrow \{0,1\}. \text{ Sample } x \leftarrow X_b. \text{ Output } (b,x).$$

Intuitively, both $\mathrm{H}(B|Y)$ and $\Delta(X_0,X_1)$ measure how well B can be predicted from $Y = X_B$. Indeed, it is shown in [57, Claim 4.4.2] that $1-\delta \le \mathrm{H}(B|Y) \le \mathrm{H}_2((1-\delta)/2)$, where $\delta = \Delta(X_0,X_1)$. Plugging in $\delta = 1/3$ and $\delta = 2/3$, we see that

$$\begin{aligned}(X_0,X_1) \in \mathrm{SD}_Y &\Rightarrow \mathrm{H}(B|Y) \ge 1 - 1/3 = 2/3,\\ (X_0,X_1) \in \mathrm{SD}_N &\Rightarrow \mathrm{H}(B|Y) \le \mathrm{H}_2((1-2/3)/2) < .651.\end{aligned}$$

Now we amplify the gap to more than one bit by taking direct products. Specifically, let $(B',Y') = ((B_1,\ldots,B_{66}),(Y_1,\ldots,Y_{66}))$, where the (B_i,Y_i)'s are independent copies of (B,Y). Then

$$\begin{aligned}(X_0,X_1) \in \mathrm{SD}_Y &\Rightarrow \mathrm{H}(B'|Y') \ge 66 \cdot (2/3) = 44,\\ (X_0,X_1) \in \mathrm{SD}_N &\Rightarrow \mathrm{H}(B'|Y') < 66 \cdot .651 < 43.\end{aligned}$$

So $(X_0,X_1) \mapsto ((B',Y'),44)$ is a valid reduction from SD to CEA. ∎

We note that the unconditional version of CEA (where Y is not present and we consider only $\mathrm{H}(X)$), called ENTROPY APPROXIMATION, is known to be complete for *noninteractive* statistical zero knowledge [33].

Analogous characterizations of **ZK**. We present analogous characterizations of **ZK**, albeit not in terms of complete problems.

DEFINITION 3.3. *A promise problem* Π *satisfies the* INDISTINGUISHABILITY CONDITION *if there is a polynomial-time computable function mapping strings* x *to pairs of samplable distributions* (X,Y) *such that*

- *if* $x \in \Pi_Y$*, then* X *and* Y *are computationally indistinguishable (in the sense of Definition* 2.12*);*
- *if* $x \in \Pi_N$*, then* $\Delta(X,Y) \ge 2/3$.

We note that the constant $2/3$ in the second bullet is arbitrary. By taking direct products and applying Lemmas 2.3 and 2.13, we can boost a threshold as low as $1/\mathrm{poly}(n)$ to as high as $1 - 2^{-\mathrm{poly}(n)}$, while preserving the computational indistinguishability in the first bullet.

Like the SZK/OWF CONDITION, if one-way functions exist, then every promise problem satisfies the INDISTINGUISHABILITY CONDITION: On an input x of length n, we can define $X = G(U_n)$ and $Y = U_{2n}$, where G is a length-doubling pseudorandom generator, and then X and Y are both computationally indistinguishable and have large statistical difference. Thus, as before, to obtain a characterization of **ZK**, we need to add the condition $\Pi \in \mathbf{IP}$.

THEOREM 3.4 (indistinguishability characterization of **ZK**). $\Pi \in \mathbf{ZK}$ *if and only if* $\Pi \in \mathbf{IP}$ *and* Π *satisfies the* INDISTINGUISHABILITY CONDITION.

The preceding example, showing that every promise problem satisfies the Indistinguishability Condition if one-way functions exist, also illustrates why Π satisfying the Indistinguishability Condition cannot be cast as a reduction from Π to some promise problem—the conditions for YES instances and NO instances may hold at the same time. Nevertheless, we expect that the indistinguishability characterization of **ZK** will have much the same utility as a complete problem (such as Statistical Difference). Indeed, several of the results about **ZK** presented in section 7 can be established simply by taking the corresponding proof for **SZK** and replacing Statistical Difference with the Indistinguishability Condition. However, we instead choose to use the results for **SZK** as a "black box," and reduce the **ZK** case to the **SZK** case via the SZK/OWF characterization.

In [52], it was already proven that every problem that has a *public-coin* computational zero-knowledge proof satisfies the Indistinguishability Condition. Thus, what is new here is showing that the characterization *holds even for private-coin proofs* and *establishing a converse* (for $\Pi \in \mathbf{IP}$).

A characterization analogous to Conditional Entropy Approximation follows.

Definition 3.5. *A promise problem* Π *satisfies the* Conditional Pseudoentropy Condition *if there is a polynomial-time computable function mapping strings* x *to a* samplable joint distribution (X, Y) *(i.e., two circuits that use the same coin tosses) and a parameter* r *such that*

- *if* $x \in \Pi_Y$*, then there exists a (not necessarily samplable) joint distribution* (X', Y') *such that* (X', Y') *is computationally indistinguishable from* (X, Y) *and* $\mathrm{H}(X'|Y') \geq r$*, and*
- *if* $x \in \Pi_N$*, then* $\mathrm{H}(X|Y) \leq r - 1$*,*

where $\mathrm{H}(\cdot|\cdot)$ *denotes conditional entropy. (See section* 2.2*.)*

As before, this definition is satisfied by all promise problems if one-way functions exist. It is crucial that we generalize Conditional Entropy Approximation instead of Entropy Difference. Indeed, in [57] we pointed out that the condition analogous to Entropy Difference, using $\mathrm{H}(X) - \mathrm{H}(Y)$ instead of $\mathrm{H}(X|Y)$, is satisfied by *all* promise problems (regardless of whether or not one-way functions exist) and thus is useless.[10] (At the time, we saw this as an obstacle to finding **ZK** analogues of the complete problems for **SZK**.) Our use of conditional entropy was inspired in part by its role in the conjectures of [4, sect. 9].

Theorem 3.6 (conditional pseudoentropy characterization of **ZK**). $\Pi \in \mathbf{ZK}$ *if and only if* $\Pi \in \mathbf{IP}$ *and* Π *satisfies the* Conditional Pseudoentropy Condition.

Note that, in contrast to the **SZK**-completeness of Entropy Difference, this theorem does not seem to imply that **ZK** is closed under complement. The reason is that the Conditional Pseudoentropy Condition is not symmetric with respect to YES and NO instances.

Overview of the proofs of Theorems 1.2*,* 3.6*, and* 3.4. In the remainder of this section, we show that every promise problem in **HVZK** satisfies the Conditional Pseudoentropy Condition, that the Conditional Pseudoentropy Condition is equivalent to the Indistinguishability Condition, and that every promise problem

[10] The reason comes from the fact that we do not require X' and Y' above to be samplable. It is known (via the probabilistic method) that there exists a (nonsamplable) distribution D of low entropy (e.g., $n/2$) that is indistinguishable from the uniform distribution U_n [26]. Thus, if the above characterization referred to $\mathrm{H}(X) - \mathrm{H}(Y)$, then it would hold for all promise problems by setting $X = Y = X' = U_n$, $Y' = D$, and $r = 1$ so that $\mathrm{H}(X') - \mathrm{H}(Y') \geq n/2 \geq r$ and $\mathrm{H}(X) - \mathrm{H}(Y) = 0 = r - 1$.

satisfying the CONDITIONAL PSEUDOENTROPY CONDITION satisfies the SZK/OWF CONDITION. This establishes the forward ("only if") directions of Theorems 1.2, 3.6, and 3.4. The reverse directions, showing that problems in **IP** satisfying the characterizations are in **ZK**, follow from our results in sections 4 and 5. Section 6 puts everything together and formally deduces the theorems.

3.2. The CONDITIONAL PSEUDOENTROPY CONDITION.

LEMMA 3.7. *If a promise problem* Π *is in* **HVZK**, *then* Π *satisfies the* CONDITIONAL PSEUDOENTROPY CONDITION.

Proof. The proof is an adaptation of the reduction from **HVZK** to ENTROPY DIFFERENCE in [34]. Let (P, V) be an honest-verifier computational zero-knowledge proof for Π, with simulator S. We modify the proof system to satisfy the following (standard) additional properties:

- The completeness error $c(|x|)$ and soundness error $s(|x|)$ are both negligible. This can be achieved by standard error reduction via (sequential) repetition.
- On every input x, the two parties exchange $2\ell(|x|)$ messages for some polynomial ℓ, with the verifier sending even-numbered messages and sending all its $r(|x|) \geq |x|$ random coin tosses in the last message. Having the verifier send its coin tosses at the end does not affect soundness because it is after the prover's last message and does not affect honest-verifier zero knowledge because the simulator is required to simulate the verifier's coin tosses.
- On every input x, the simulator always outputs *accepting transcripts*, where we call a sequence γ of 2ℓ messages an accepting transcript on x if all the verifier's messages are consistent with its coin tosses (as specified in the last message) and the verifier accepts in such an interaction. To achieve this, we first modify the proof system so that the verifier always accepts if its coin tosses are $0^{r(|x|)}$; this increases the soundness error only negligibly. Then we modify the simulator so that any time it is about to output a nonaccepting transcript, it instead outputs the accepting transcript where all of the prover messages are the empty string and the verifier's coin tosses are $0^{r(|x|)}$. This has a negligible effect on the quality of the simulation because when $x \in \Pi_Y$, the original simulator can only output nonaccepting transcripts with negligible probability (otherwise its output could easily be distinguished from the real interaction, which has nonaccepting transcripts with probability at most $c(|x|) = \mathrm{neg}(|x|)$).

For a transcript γ, we denote by γ_i the *prefix* of γ consisting of the first i messages. For readability, we often drop the input x from the notation, e.g., using $\ell = \ell(|x|)$, $\langle P, V\rangle = \langle P, V\rangle(x)$, etc. Thus, in what follows, $\langle P, V\rangle_i$ and S_i are random variables representing prefixes of transcripts generated by the real interaction and simulator, respectively, on a specified input x.

The following two claims are shown in [1, 51, 34].

CLAIM 3.8. *For every* x,

$$\sum_{i=1}^{\ell} [\mathrm{H}(\langle P, V\rangle_{2i}) - \mathrm{H}(\langle P, V\rangle_{2i-1})] = r.$$

Since $\langle P, V\rangle_{2i-1}$ is a prefix of $\langle P, V\rangle_{2i}$, the term $\mathrm{H}(\langle P, V\rangle_{2i}) - \mathrm{H}(\langle P, V\rangle_{2i-1})$ in the sum equals the conditional entropy $\mathrm{H}(\langle P, V\rangle_{2i} | \langle P, V\rangle_{2i-1})$. Thus, the sum measures the total entropy contributed by the verifier's messages, and it is natural that this should equal the number of coin tosses of the verifier. (Recall that the verifier reveals its coin tosses at the end.)

What is less obvious is that the sum should be significantly smaller when we consider the simulated transcripts for $x \in \Pi_N$ (rather than for $x \in \Pi_Y$).

CLAIM 3.9. *For every $x \in \Pi_N$,*

$$\sum_{i=1}^{\ell} [\mathrm{H}(S_{2i}) - \mathrm{H}(S_{2i-1})] \leq r - \log \frac{1}{s(|x|)} < r - 1.$$

It may seem strange to consider the simulator's output distribution on NO instances, since the zero-knowledge condition does not provide any guarantees about the quality of simulation on NO instances. Indeed, Claim 3.9 is not derived from the zero-knowledge property of the proof system. Rather, it is based on the *soundness* of the proof system and the fact that the simulator always produces accepting transcripts (by our modification above). Intuitively, it says that the simulation captures at most an $s(|x|)$ fraction of the probability space of the verifier's messages. Indeed, it is shown in [1, 51, 34] that if this were not the case, then the simulator could be used to construct a prover strategy that convinces the verifier to accept with probability greater than $s(|x|)$, contradicting the soundness of the proof system. Now, given input x, we construct circuits that sample from the following (joint) random variables:

(X, Y): Select $i \leftarrow \{1, \ldots, \ell(|x|)\}$, choose random coin tosses R for the simulator, and output $(S_{2i}(x; R), S_{2i-1}(x; R))$.

When $x \in \Pi_Y$, then S is computationally indistinguishable from $\langle P, V\rangle$. So (X, Y) is indistinguishable from $(X', Y') = (\langle P, V\rangle_{2I}, \langle P, V\rangle_{2I-1})$, where I denotes a uniform random element of $\{1, \ldots, \ell\}$. By Claim 3.8, we have

$$\mathrm{H}(X'|Y') = \frac{1}{\ell} \sum_{i=1}^{\ell} \mathrm{H}(\langle P, V\rangle_{2i} | \langle P, V\rangle_{2i-1}) = \frac{r}{\ell}.$$

When $x \in \Pi_N$, then by Claim 3.9, we have

$$\mathrm{H}(X|Y) = \frac{1}{\ell} \sum_{i=1}^{\ell} \mathrm{H}(S_{2i}|S_{2i-1}) \leq \frac{r-1}{\ell}.$$

This is what we need to prove, except the entropy gap is only $1/\ell$. This can be increased to 1 by taking ℓ independent samples from the joint distribution. That is, we define $(\overline{X}, \overline{Y}) = ((X_1, \ldots, X_\ell), (Y_1, \ldots, Y_\ell))$, where the (X_i, Y_i)'s are independent copies of (X, Y). When $x \in \Pi_Y$, then $(\overline{X}, \overline{Y})$ is computationally indistinguishable from the analogously defined $(\overline{X'}, \overline{Y'})$, and $\mathrm{H}(\overline{X'}|\overline{Y'}) = \ell \cdot \mathrm{H}(X'|Y') = r$. Also, when $x \in \Pi_N$, then $\mathrm{H}(\overline{X}|\overline{Y}) = \ell \cdot \mathrm{H}(X|Y) \leq r - 1$.

Therefore the mapping $x \mapsto ((\overline{X}, \overline{Y}), r)$ satisfies Definition 3.5. ∎

3.3. The SZK/OWF CONDITION. In this section, we show that the CONDITIONAL PSEUDOENTROPY CONDITION implies the SZK/OWF CONDITION.

LEMMA 3.10. *If a promise problem satisfies the* CONDITIONAL PSEUDOENTROPY CONDITION*, then it also satisfies the* SZK/OWF CONDITION.

The idea behind the proof is the following. If Π satisfies the CONDITIONAL PSEUDOENTROPY CONDITION, then on every YES instance, we obtain a samplable distribution (X, Y) that is computationally indistinguishable from (X', Y'), where $\mathrm{H}(X'|Y')$ is large. We consider two cases. If, for the original distributions X and Y, we have

that $\mathrm{H}(X|Y)$ is large, then the instance is information-theoretically distinguishable from a NO instance (where $\mathrm{H}(X|Y)$ is small), and such instances can be reduced to CONDITIONAL ENTROPY APPROXIMATION, which is complete for **SZK** by Proposition 3.2. If instead $\mathrm{H}(X|Y)$ is small, then (X, Y) is computationally indistinguishable from a joint distribution with higher conditional entropy (namely, (X', Y')). From such a pair, we can construct a one-way function using the techniques of Håstad et al. [37]. This case analysis provides the partition of YES instances into **SZK** instances and OWF instances.

Before proceeding with the actual proof, we state the result we need from [37], adapted to our auxiliary-input setting.

DEFINITION 3.11. *An* auxiliary-input false entropy generator on I *is a samplable auxiliary-input probability ensemble* $\mathcal{D} = \{D_x\}$ *for which there exists a samplable auxiliary-input probability ensemble* $\mathcal{F} = \{F_x\}$ *that is computationally indistinguishable from* $\mathcal{D}$ *on* I *and satisfies* $\mathrm{H}(F_x) \geq \mathrm{H}(D_x) + 1$ *for all* $x \in I$.

Note that the above definition refers to entropy, rather than conditional entropy as in the intuition above. We will need to cope with this in the proof. Also note that the definition requires that $\mathcal{F} = \{F_x\}$ is also samplable. This is actually not necessary (i.e., Lemma 3.12 below holds regardless), but we will achieve samplability of $\mathcal{F}$ in passing from conditional entropy to entropy, so we include the samplability condition for consistency with [37].[11]

LEMMA 3.12 (see [37]; cf. our Appendix B). *If there exists an auxiliary-input false entropy generator on* I*, then there exists an auxiliary-input one-way function on* I.

Håstad et al. [37] actually show how to construct pseudorandom generators, rather than just one-way functions, but we need only a one-way function to establish the SZK/OWF CONDITION. This allows some steps in the construction to be omitted; see Appendix B for a proof of Lemma 3.12 (and a generalization, which we use to handle expected polynomial-time simulators in section 7.3).

Proof of Lemma 3.10. Given an instance x of the promise problem Π, we can efficiently construct two samplable distributions (X, Y) and parameter r such that if $x \in \Pi_Y$, then $\mathrm{H}(X'|Y') \geq r + 1$ for some (X', Y') indistinguishable from (X, Y), and if $x \in \Pi_N$, then $\mathrm{H}(X|Y) \leq r - 1$. (We may assume a gap of 2 rather than 1 by taking multiple independent samples from the joint distribution.)

Let I be the set of instances $x \in \Pi_Y$ such that $\mathrm{H}(X|Y) < r$. First, we show that $\Pi' = (\Pi_Y \setminus I, \Pi_N)$ is in **SZK**. We prove this by reducing Π' to CONDITIONAL ENTROPY APPROXIMATION. Indeed, the reduction is simply $x \mapsto ((X, Y), r + 1)$. Then $\mathrm{H}(X|Y) \geq r$ when $x \in \Pi_Y \setminus I$, and $\mathrm{H}(X|Y) \leq r - 1$ when $x \in \Pi_N$, as needed.

Now we show that we can construct a one-way function from instances $x \in I$. Intuitively, the facts that $\mathrm{H}(X'|Y') \geq r + 1$ and that (X', Y') is indistinguishable from (X, Y) mean we should be able to extract $r+1$ *pseudorandom* bits from X given Y. That is, if we let H be a random hash function mapping to $r + 1$ bits, then the distribution $(H, Y, H(X))$ is computationally indistinguishable from $(H, Y', H(X'))$, which we might hope to be statistically close to (H, Y', U_{r+1}) (because $\mathrm{H}(X'|Y') \geq r+1$), which in turn is computationally indistinguishable from (H, Y, U_{r+1}). However, the entropy of $(H, Y, H(X))$ equals $\mathrm{H}(H)+\mathrm{H}(Y)+\mathrm{H}(X|Y) < \mathrm{H}(H)+\mathrm{H}(Y)+r$, which is one bit less than the entropy of (H, Y, U_{r+1}). So, if this argument worked, then

[11] The samplability of $\mathcal{F}$ is needed only in [37] for proving results with respect to *uniform* adversaries. Indeed, the condition was not included in the conference version [38], which dealt only with nonuniform adversaries.

$(H, Y, H(X))$ would be a false entropy generator.

However, there are two (standard) difficulties in implementing this intuition. First, entropy (much less conditional entropy) is not a strong enough measure of randomness to allow extracting almost-uniform bits. (That is, it is not guaranteed that $(H, Y', H(X'))$ is statistically close to (H, Y', U_{r+1}).) Instead, we need a lower bound on (conditional) *min-entropy*, as required in the leftover hash lemma (Lemma 2.7). Second, randomness extraction (e.g., as provided by the leftover hash lemma) does not extract all the bits of min-entropy, but rather suffers an entropy loss related to the distance ε desired from uniform in the extracted bits. So we need a larger gap than one bit of entropy to tolerate this loss and still obtain a false entropy generator. Both of these difficulties are solved by taking direct products, i.e., many independent samples of (X, Y). Taking a direct product has the effect of both (linearly) growing the entropy gap and converting entropy to min-entropy (with a sublinearly loss in entropy, as shown by the flattening lemma, Lemma 2.6).

We now proceed with the formal details. Let $n = |x|$, let m be the number of bits output by X, set $k = 4n \cdot (m+n)^2$, and let $\mathcal{H}$ be an explicit family of universal hash functions mapping $\{0,1\}^{km}$ to $\{0,1\}^{kr+1}$. Let $s = O(km)$ be the number of random bits to choose a random hash function from $\mathcal{H}$. Consider the samplable distribution

$$D = (H, Y_1, \ldots, Y_k, H(X_1, \ldots, X_k)),$$

where H is a random hash function from $\mathcal{H}$, and the (X_i, Y_i)'s are independent copies of (X, Y). When $x \in I$, we have $\mathrm{H}(D) \leq s + k \cdot \mathrm{H}(Y) + k \cdot r$. On the other hand, we will show below that D is computationally indistinguishable from the samplable distribution

$$F = (H, Y_1, \ldots, Y_k, U_{kr+1}),$$

which has entropy $s + k \cdot \mathrm{H}(Y) + (kr+1)$, which in turn is one bit larger than the entropy of D. Thus, we have constructed an auxiliary-input false entropy generator on I, and thus by Lemma 3.12 there exists a one-way function on I, as desired.

We now proceed to show that when $x \in \Pi_Y$, D is computationally indistinguishable from F. We know that there exist (X', Y') indistinguishable from (X, Y) such that $\mathrm{H}(X'|Y') \geq r+1$. By Lemma 2.2, we can modify (X', Y') to obtain (X^*, Y^*) indistinguishable from (X, Y) such that $\mathrm{H}(X^*|Y^*) \geq r+1$ and $\Pr[X^* = x | Y^* = y] \geq 2^{-n} \cdot 2^{-m}$ for all $(x, y) \in \mathrm{Supp}(X^*, Y^*)$.

By a hybrid argument (Lemma 2.13), D and F are computationally indistinguishable from

$$D^* = (H, Y_1^*, \ldots, Y_k^*, H(X_1^*, \ldots, X_k^*))$$

and

$$F^* = (H, Y_1^*, \ldots, Y_k^*, U_{kr+1}),$$

respectively, where the (X_i^*, Y_i^*)'s are independent copies of (X^*, Y^*).

Now we proceed to show that D^* is statistically indistinguishable from F^*, which will complete the proof. By Lemma 2.6, $\overline{X^*} = (X_1^*, \ldots, X_k^*)$ is Φ-flat given $\overline{Y^*} = (Y_1^*, \ldots, Y_k^*)$ for $\Phi = \sqrt{k} \cdot (m+n)$. This implies that $(\overline{X^*}, \overline{Y^*})$ is (2^{-n})-close to some $(W, \overline{Y^*})$ such that for every $\overline{y} \in \mathrm{Supp}(\overline{Y^*})$, the min-entropy of W conditioned on $\overline{Y^*} = \overline{y}$ is at least

$$\begin{aligned} k \cdot \mathrm{H}(X^*|Y^*) - \sqrt{n} \cdot \Phi &\geq k \cdot (r+1) - \sqrt{n} \cdot \Phi \\ &> kr + 2n + 1, \end{aligned}$$

where in the last inequality we use $\sqrt{n}\Phi \leq k/2$ and $2n+1 \leq k/2$.

Thus, D^* is statistically close to the distribution $(H, \overline{Y^*}, H(W))$, which is (2^{-n})-close to $(H, \overline{Y^*}, U_{kr+1}) = F^*$ by the leftover hash lemma (Lemma 2.7). This completes the proof. □

3.4. The Indistinguishability Condition. In this section, we show that the Indistinguishability Condition is equivalent to the Conditional Pseudoentropy Condition, and is thus satisfied by every problem in **HVZK**. This equivalence is proven using computational analogues of the reductions given in [57, sects. 3.4 and 4.4] between the complete problems for **SZK**. We note that the results of this section are not used later in the paper, except of course to establish the indistinguishability characterization of **ZK** (Theorem 3.4); they are included because this characterization may be of independent interest and may be of use in further studies of **ZK**.

Lemma 3.13. *If a promise problem satisfies the* Conditional Pseudoentropy Condition*, then it satisfies the* Indistinguishability Condition.

Proof. The reduction is identical to the one used in the proof of Lemma 3.10 to construct a pseudoentropy generator on the instances in I. Let Π be a promise problem satisfying the Conditional Pseudoentropy Condition. As in the proof of Lemma 3.10, given an instance x of the promise problem Π, we can efficiently construct two samplable distributions (X, Y) and parameter r such that if $x \in \Pi_Y$, then $\mathrm{H}(X'|Y') \geq r+1$ for some (X', Y') indistinguishable from (X, Y), and if $x \in \Pi_N$, then $\mathrm{H}(X|Y) \leq r-1$. From X and Y, we can construct the samplable distributions D and F as in the proof of Lemma 3.10. In that proof, it is shown that when $x \in \Pi_Y$, then D and F are computationally indistinguishable. It is also shown that when $\mathrm{H}(X|Y) < r$ (in particular if $x \in \Pi_N$), then $\mathrm{H}(F) \geq \mathrm{H}(D) + 1$. By Lemma 2.1, this implies that $\Delta(D, F) \geq 1/2\ell$, where $\ell = \mathrm{poly}(n)$ is the number of bits output by D and F. Applying Lemmas 2.3 and 2.13, we can increase the statistical difference to $2/3$ on NO instances while maintaining computationally indistinguishability on YES instances. Thus, we conclude that Π satisfies the Indistinguishability Condition. □

Lemma 3.14. *If a promise problem satisfies the* Indistinguishability Condition*, then it satisfies the* Conditional Pseudoentropy Condition.

Proof. This is proved in the same way that we reduced Statistical Difference to Conditional Entropy Approximation in the proof of Proposition 3.2. Let Π be a promise problem satisfying the Indistinguishability Condition. This means that given an instance x of Π, we can efficiently construct two samplable distributions (X_0, X_1) such that X_0 and X_1 are computationally indistinguishable if $x \in \Pi_Y$ and such that $\Delta(X_0, X_1) \geq 2/3$ if $x \in \Pi_N$. Consider the following pair of jointly distributed random variables:

$$(B, Y): \quad \text{Select } b \leftarrow \{0,1\}. \ \text{Sample } x \leftarrow X_b. \ \text{Output } (b, x).$$

When $x \in \Pi_Y$, the distributions X_0 and X_1 are computationally indistinguishable. This implies that (B, Y) is computationally indistinguishable from (B', Y), where B' is a random bit independent of Y. Note that $\mathrm{H}(B'|Y) = 1$.

When $x \in \Pi_N$, it holds that $\Delta(X_0, X_1) \geq 2/3$. Then, as in the proof of Proposition 3.2, we have $\mathrm{H}(B|Y) < .651 < 2/3$.

Thus, the mapping $x \mapsto (B, Y), r = 1$ meets the requirements of the Conditional Pseudoentropy Condition, except that the gap in conditional entropies between the two cases is only $1 - 2/3 = 1/3$ bits. The gap can be amplified to one bit by taking direct products in the usual manner. □

4. From the SZK/OWF Condition to ZK. In this section, we construct a computational zero-knowledge proof system for every problem Π in **IP** that satisfies the SZK/OWF Condition. A first approach is for the prover to use the **SZK** proof system, when the input is in $\Pi_Y \setminus I$, and to use the proof system obtained by the generic, one-way-function-based compiler from **IP** to **ZK** [39, 7] when the input is in I. The difficulty with this is that the set I may not be efficiently recognizable, so this approach leaks information to the verifier (namely, whether or not the input is in I). Because of this difficulty, we take a more indirect approach. Instead of trying to construct separate zero-knowledge proofs for the "**SZK** instances" and the "OWF instances" and then combining them, we construct a certain type of bit-commitment scheme in each of the two cases. The advantage is that the commitment schemes are easy to combine (via simple secret sharing). We then use the combined commitment scheme in the generic compiler from **IP** to **ZK** [39, 7].

4.1. Instance-dependent commitments. Recall that a *commitment scheme* is a two-phase protocol between a sender and a receiver. In the first phase, called the *commit phase*, the sender "commits" to a private bit b. In the second phase, called the *reveal phase*, the sender reveals b and "proves" that it was the bit to which she committed in the first phase. We require two properties of commitment schemes. The *hiding* property says that the receiver learns nothing about m in the first phase. The *binding* property says that after the commit phase, the sender is bound to a particular value of m; that is, she cannot successfully open the commitment to two different messages in the reveal phase. It is impossible to have commitment schemes that are both statistically hiding and statistically binding, but it is known how to construct commitment schemes that are computationally hiding and statistically binding, assuming one-way functions exist [45, 37]. In fact, this is the only way that one-way functions are used in the construction of computational zero-knowledge proofs for all of **IP** [29, 39, 7] and all the resulting theorems about **ZK** that rely on the assumption that one-way functions exist.

In this section, we will show how to use the fact that a promise problem Π satisfies the SZK/OWF Condition to construct a relaxed form of commitment scheme, tailored to Π, that still suffices for obtaining a zero-knowledge proof for Π. Specifically, we will construct an *instance-dependent commitment scheme* for Π. This is an auxiliary-input version of a commitment protocol, where the auxiliary input x (given to both the sender and receiver) is viewed as an instance of the promise problem Π. It is required that the scheme is hiding when $x \in \Pi_Y$ and is binding when $x \in \Pi_N$. Thus, they are a relaxation of standard commitment schemes, since we do not require that the hiding and binding properties hold at the same time. Nevertheless, this relaxation is still useful in constructing zero-knowledge proofs. The reason is that zero-knowledge proofs based on commitments (in, e.g., [29, 39, 7]) typically use only the hiding property in proving zero knowledge (which is required only when x is a YES instance) and use only the binding property in proving soundness (which is required only when x is a NO instance).

An example, used in Bellare, Micali, and Ostrovsky [6], is based on the Graph Isomorphism problem: Given graphs (G_0, G_1), a commitment to bit $b \in \{0,1\}$ is a random isomorphic copy of G_b. When $G_0 \cong G_1$, the commitment is perfectly hiding, and when $G_0 \ncong G_1$, then the commitment is perfectly binding. This idea was abstracted by Itoh, Ohta, and Shizuya [40], who studied the general utility of instance-dependent commitment schemes for constructing zero-knowledge proofs. Specifically, they showed that every language possessing a noninteractive instance-dependent

commitment scheme that is perfectly binding and perfectly hiding is in **PZK**, as is the complement of every such language. Recently, in [44], the notion was further generalized to allow interactive commitments, statistical security, and promise problems, and was suggested as a possible tool for proving that every problem in $\mathbf{SZK} \cap \mathbf{NP}$ has a statistical zero-knowledge proof system with an efficient prover.

Here we consider further relaxations of the definition. First, we allow the hiding property to be computational, since we will use them to construct computational zero-knowledge proofs. Second, we require security only for an honest receiver (i.e., one that follows the specified protocol); this means that the zero-knowledge proofs we construct with them will only be *honest-verifier* zero knowledge. However, since our honest-verifier zero-knowledge proofs will also be public coin (due to the instance-dependent commitments being public coin), we will be able to make them robust against cheating verifiers using the compiler of [32]. Third, and most significantly, we allow the sender's algorithm to be computationally unbounded. This is okay when we use the instance-dependent commitments to construct zero-knowledge proofs, because the sender's role is played by the prover, who is allowed to be computationally unbounded. (Though this naturally renders the commitments useless for the application in [44], which focused on prover efficiency.)

The fact that the sender is not polynomial time, however, complicates the definition substantially, because many commonly used properties of commitment schemes implicitly use the fact that the sender algorithm is polynomial time. For example, with a standard commitment scheme, one can assume without loss of generality that we have a "canonical reveal phase," whereby the sender gives the message m and her coin tosses r to the receiver and the receiver checks that the transcript of the commit phase is consistent with m and r. (See [23, sect. 4.4.1].) This is not possible when the sender is computationally unbounded, because the receiver cannot run the sender's algorithm to check the transcript. Another example is the fact that commitments are automatically "zero knowledge" in the sense that the receiver learns nothing (from both phases) other than the bit b to which the sender commits; this is the case because the receiver can simulate a commitment to bit b by simply running the sender's algorithm. Instead, we will need to explicitly include such properties in the following definition.

DEFINITION 4.1. *An (unbounded-sender, honest-receiver)* instance-dependent commitment scheme *for a promise problem* Π *consists of two interactive protocols* (S_1, R_1) *(the* commitment phase*) and* (S_2, R_2) *(the* reveal phase*) and a promise problem* $\textsc{Val} = (\textsc{Val}_Y, \textsc{Val}_N)$ *(capturing the "valid" commitments). In the commitment phase, both* S_1 *and* R_1 *receive a common input* $x \in \{0,1\}^*$*,* S_1 *receives a private input* $b \in \{0,1\}$*, and the protocol produces as output a* commitment z*. In the reveal phase, both* S_2 *and* R_2 *receive the common input* $x \in \{0,1\}^*$*, a commitment* z*, and a bit* $b \in \{0,1\}$*, and at the end of the protocol,* R_2 *accepts or rejects. We allow* S_1 *and* S_2 *(resp.,* R_1 *and* R_2*) to share the same coin tosses (as a way to maintain private state beyond the public commitment value* z*). We write* $(S_1(b), R_1)(x)$*,* $(S_2, R_2)(x, z, b)$*, and* $(S, R)(x, b)$ *to denote the interaction between* S *and* R *in the commit phase, reveal phase, and the two phases combined, respectively.*

We require the following conditions:

1. *(Efficiency)* $R = (R_1, R_2)$ *is computable in PPT (in the length of the common input* x*). (In contrast,* S *is allowed to be computationally unbounded.)*
2. *(Completeness) For all* $x \in \{0,1\}^n$ *and all* $b \in \{0,1\}$*, if we let* z *be the output of* $(S_1(b), R_1)(x)$*, then* $(x, z, b) \in \textsc{Val}_Y$ *with probability* $1 - \mathrm{neg}(n)$*.*

3. *(Validity tests)* (S_2, R_2) *is an interactive proof system (with negligible error probabilities) for* VAL. *Moreover, the promise problem* VAL *is in* **AM**.
4. *(Statistical zero knowledge) There is a PPT algorithm* M *such that for every* $x \in \{0,1\}^*$ *and* $b \in \{0,1\}$, *the distribution* $M(x,b)$ *has statistical difference* $\mathrm{neg}(n)$ *from* R*'s view of* $(S,R)(x,b)$.
5. *(Computationally hiding on* YES *instances) If* $x \in \Pi_Y$, *then* R*'s views in* $(S_1(0), R_1)(x)$ *and* $(S_1(1), R_1)(x)$ *are computationally indistinguishable. In case these views are statistically indistinguishable, we will refer to the scheme as* statistically hiding.
6. *(Statistically binding on* NO *instances) If* $x \in \Pi_N$, *then for every* S^*, *if we let* z *be the output of* $(S_1^*, R_1)(x)$, *then with probability at least* $1 - \mathrm{neg}(n)$, *either* $(x,z,0)$ *or* $(x,z,1)$ *is in* VAL_N.

The commitment scheme is called public coin *if it is public coin for the receiver* R.

We make a few remarks on the above conditions, as follows:

- As mentioned earlier, the fact that we allow S to be computationally unbounded results in several differences between the above definition and standard definitions of commitment schemes. When S is restricted to be polynomial time, the zero-knowledge condition is trivial to satisfy (because $M(x,b)$ could carry out an execution of $(S,R)(x,b)$) and thus is typically omitted, and the reveal phase can, without loss of generality, consist of S just sending its coin tosses to R.
- The completeness and zero-knowledge conditions (and the validity tests) are required for all inputs $x \in \{0,1\}^*$, not just those that satisfy the promise of Π. This will be useful in combining two instance-dependent commitment schemes to obtain one for the union of the corresponding promise problems.
- The definition provides for two different kinds of validity tests. One is the specified protocol (S_2, R_2) (which may have many rounds, but is "zero knowledge" according to item 4). The other is the (unspecified) **AM** protocol for VAL (which has only two rounds). Both will be useful for us.
- Both the zero-knowledge and hiding conditions are required only for honest receivers. The result is that the proof systems we construct using such commitments will be only honest-verifier zero knowledge. We will then obtain zero knowledge against cheating-verifier strategies using the compiler of [32] and the fact that our commitment schemes are public coin.

As shown in section 6, our results yield characterizations of **ZK** and **SZK** in terms of instance-dependent commitment schemes, as follows.

THEOREM 4.2 (commitment characterization of **ZK**). $\Pi \in$ **ZK** *if and only if* $\Pi \in$ **IP** *and* Π *has a public-coin,* computationally *hiding instance-dependent commitment scheme in the sense of Definition* 4.1.

THEOREM 4.3 (commitment characterization of **SZK**). $\Pi \in$ **SZK** *if and only if* $\Pi \in$ **IP** *and* Π *has a* statistically *hiding instance-dependent commitment scheme in the sense of Definition* 4.1.

These theorems demonstrate that commitment schemes are at the heart of all zero-knowledge proofs. This intuition has been held by researchers for a number of years, based first on the construction of zero-knowledge proofs for all of NP and **IP** from commitment schemes [29, 39, 7]. Partial converses were given by Damgård [12, 13], who showed that every problem having a 3-*round, public-coin* zero-knowledge proof has an instance-dependent commitment scheme[12] (as above, the commitment is

[12] Damgård's result is not stated in the language of instance-dependent commitments, but this formulation seems to follow from his technique.

statistically hiding if the proof system is statistical zero knowledge), and by Ostrovsky [49] and Ostrovsky and Wigderson [50], who showed that zero-knowledge proofs for *hard-on-average* languages imply one-way functions (and hence standard commitment schemes [45, 37]). As far as we know, the above theorems are the first to establish a genuine *equivalence* between zero-knowledge proofs and some form of commitment schemes.

In this section, we focus on proving the forward direction of Theorem 4.2.

LEMMA 4.4. *If a promise problem* Π *satisfies the* SZK/OWF CONDITION*, then* Π *has an instance-dependent commitment scheme (in the sense of Definition* 4.1*). Moreover, the scheme is public coin and the sender is PPT given an* **NP** *oracle.*

We will prove this lemma by dealing separately with the **SZK** instances and OWF instances. This is done by combining two instance-dependent commitment schemes, one that is hiding on the "OWF instances" and the other that is hiding on the "**SZK** (YES) instances." For the OWF instances, we use a straightforward application of the known construction of commitment schemes from one-way functions [45, 37].

LEMMA 4.5. *If there exists an auxiliary-input one-way function on set* I*, then there is an instance-dependent commitment scheme for the promise problem* $\Pi = (I, \overline{I})$*. Moreover, this commitment scheme is public coin and the sender is PPT.*

Proof. By Theorem 2.15, we can construct an auxiliary-input pseudorandom generator $\{G_x : \{0,1\}^{p(|x|)} \to \{0,1\}^{3p(|x|)}\}$ on I. Now we adapt Naor's commitment scheme from pseudorandom generators [45] as follows:

Commit phase. $(S_1(b), R_1)(x)$, where $|x| = n$.

1. R_1 chooses $v \leftarrow \{0,1\}^{3p(n)}$ and sends v to S_1. Both parties set $v_1 = v$ and $v_0 = 0^{3p(n)}$.
2. S_1 chooses $r \leftarrow \{0,1\}^{p(n)}$ and sends $w = G_x(r) \oplus v_b$ to R_1.
3. The commitment z is defined to be the pair (v, w).

We define the promise problem $\text{VAL} = (\text{VAL}_Y, \text{VAL}_N)$ by

$$\text{VAL}_Y = \{(x,(v,w),b) : \exists r \in \{0,1\}^{p(|x|)}\ w = G_x(r) \oplus v_b\},$$
$$\text{VAL}_N = \overline{\text{VAL}_Y},$$

where again we define $v_1 = v$ and $v_0 = 0^{3p(|x|)}$. Clearly $\text{VAL} \in \mathbf{NP}$, and in fact the reveal phase (S_2, R_2) simply consists of the sender S_2 providing the standard **NP** proof that $(x,(v,w),b) \in \text{VAL}_Y$ (namely, r such that $w = G_x(r) \oplus v_b$). Thus we have the required validity tests.

The completeness and public coin properties hold by inspection. The zero-knowledge condition holds because the sender is polynomial time. Following [45], the (computational) hiding property on $x \in I$ follows from the pseudorandomness of G_x on such instances. Specifically, we know that $G_x(U_{p(n)})$ is indistinguishable from $U_{3p(n)}$. Thus, if we let the random variable V denote the message of R_1, we see that R_1's view of a commitment to 1, $(V, G_x(U_{p(n)}) \oplus V)$, is indistinguishable from $(V, U_{3p(n)} \oplus V) \equiv (V, U_{3p(n)})$, which in turn is indistinguishable from R_1's view of a commitment to 0, $(V, G_x(U_{p(n)}))$. Following [45], the (statistical) binding property on $x \notin I$ (in fact on all $x \in \{0,1\}^*$) follows from the fact that G_x is length-tripling. Specifically, with probability at least $1 - 2^{-p(n)}$ over $v \leftarrow \{0,1\}^{3p(n)}$, the image of G_x will be disjoint from the image of $G_x \oplus v$, in which case there is no w such that (v, w) is a valid commitment of both 0 and 1. ∎

For the **SZK** instances, we prove the following (which is the forward direction of Theorem 4.3) in section 5.

LEMMA 4.6. *Every problem* Π *in* **SZK** *has an instance-dependent commitment*

scheme. Moreover, the scheme is public coin and statistically hiding, and the sender is PPT given an **NP** *oracle.*

We now show how to combine these two commitment schemes to prove Lemma 4.4.

LEMMA 4.7. *If promise problems* $\Pi = (\Pi_Y, \Pi_N)$ *and* $\Gamma = (\Gamma_Y, \Gamma_N)$ *each have instance-dependent commitment schemes, then the promise problem* $\Pi \cup \Gamma \stackrel{\text{def}}{=} (\Pi_Y \cup \Gamma_Y, \Pi_N \cap \Gamma_N)$ *has an instance-dependent commitment scheme. If the commitment schemes for* Π *and* Γ *are both public coin, then so is the commitment scheme for* $\Pi \cup \Gamma$*. Moreover, the strategy of the sender in the commitment scheme for* $\Pi \cup \Gamma$ *on auxiliary input* x *is PPT given oracle access to the strategies of the senders in the commitment schemes for* Π *and* Γ *on auxiliary input* x*.*

Proof. Let (S', R') be the instance-dependent commitment scheme for Π, and (S'', R'') the one for Γ, with valid commitments defined by promise problems VAL' and VAL''. Intuitively, on an input x, we would like to use (S', R') if $x \in \Pi_Y$ and use (S'', R'') if $x \in \Gamma_Y$. Unfortunately, we do not know which is the case. So we will use *both*, and do so in such a way that the resulting scheme is hiding even when only one of the two is hiding. The natural thing to do is for the sender to commit to two "shares" of its bit b, one with each scheme, and this is indeed what we do.

Specifically the new scheme $(S, R) = ((S_1, S_2), (R_1, R_2))$ is constructed as follows:

Commit phase $(S_1(b), R_1)(x)$:
1. S_1 chooses random $b', b'' \leftarrow \{0,1\}$ such that $b' \oplus b'' = b$.
2. S_1 and R_1 execute $(S_1'(b'), R_1')(x)$ and $(S_1''(b''), R_1'')(x)$ to obtain commitments z' and z'', respectively.
3. The output commitment is $z = (z', z'')$.

Valid commitments: The promise problem of valid commitments is defined to be $\text{VAL} = (\text{VAL}_Y, \text{VAL}_N)$, where

$$\text{VAL}_Y = \{(x, (z', z''), b) : \exists\, b', b'' \in \{0,1\} \quad [b' \oplus b'' = b] \wedge [(x, z', b') \in \text{VAL}_Y'] \wedge [(x, z'', b'') \in \text{VAL}_Y'']\},$$

$$\text{VAL}_N = \{(x, (z', z''), b) : \forall\, b', b'' \in \{0,1\} \quad [b' \oplus b'' \neq b] \vee [(x, z', b') \in \text{VAL}_N'] \vee [(x, z'', b'') \in \text{VAL}_N'']\}.$$

Reveal phase $(S_2, R_2)(x, (z', z''), b)$:
1. S_2 sends b', b''.
2. R_2 checks that $b' \oplus b'' = b$ and rejects immediately if not.
3. S_2 and R_2 execute $(S_2', R_2')(x, z', b')$ and $(S_2'', R_2'')(x, z'', b'')$, and R_2 accepts if both R' and R'' accept.

The completeness property of (S, R) on all x follows from the completeness properties of (S', R') and (S'', R''), which guarantee that with high probability $(x, z', b') \in \text{VAL}_Y'$ and $(x, z'', b'') \in \text{VAL}_Y''$, and hence $(x, (z', z''), b) \in \text{VAL}_Y$. (Here it is important that we require completeness to hold on all instances, rather than just on YES instances, since Π_Y and Γ_Y need not be the same.) The fact that (S_2, R_2) is an interactive proof for VAL follows by inspection, and the fact that $\text{VAL} \in \mathbf{AM}$ follows from the fact that both VAL' and VAL'' are in **AM** (combining the **AM** proof systems in the same way that we combined (S_2', R_2') and (S_2'', R_2'') to get (S_2, R_2)). For the zero-knowledge property, we have the new simulator $M(x, b)$ choose $b', b'' \leftarrow \{0,1\}$ such that $b' \oplus b'' = b$, run the original simulators $M'(x, b')$ and $M''(x, b'')$, and combine their outputs to simulate the view of R.

For the hiding property on $\Pi_Y \cup \Gamma_Y$, suppose w.l.o.g. that $x \in \Gamma_Y$. Note that the view of R_1 in $(S_1(b), R_1)(x)$ consists of the view of R_1' in $(S_1'(b'), R_1')(x)$ concatenated

with the view of R_1'' in $(S_1''(b''), R_1'')(x)$, where b' and b'' are chosen randomly such that $b' \oplus b'' = b$. The first part of the view (namely, the R_1'-view) has the same distribution regardless of the value b, because b' is a random bit. Thus, it suffices to show that for every fixed value of b' and the R_1'-view, the R_1''-view in case $b'' = b'$ (i.e., $b = 0$) is indistinguishable from the R_1''-view in case $b'' \neq b'$ (i.e., $b = 1$). But this follows from the hiding property of (S'', R'') on $x \in \Gamma_Y$.

The binding property on $x \in \Pi_N \cap \Gamma_N$ follows from the binding properties of the two commitment schemes: For every strategy S^*, we know that with high probability, the output (z', z'') of (S^*, R) satisfies the following. There is at most one $b' \in \{0,1\}$ such that $(x, z', b') \notin \text{VAL}_N'$, and there exists at most one $b'' \in \{0,1\}$ such that $(x, z'', b'') \notin \text{VAL}_N''$. Thus there is at most one b (namely, $b = b' \oplus b''$) such that $(x, (z', z''), b) \notin \text{VAL}_N$, as desired.

By inspection, the above transformation maintains public coins and the sender's complexity. □

Putting the above together, we can prove Lemma 4.4, stating that every language satisfying the SZK/OWF CONDITION has an instance-dependent commitment scheme.

Proof of Lemma 4.4. Let Π be any promise problem satisfying the SZK/OWF CONDITION, and let $I \subseteq \Pi_Y$ be the set of "OWF instances." Since $\Pi' = (\Pi_Y \setminus I, \Pi_N)$ is in **SZK**, Lemma 4.6 gives us an instance-dependent commitment scheme for Π', with public coins and a sender that is PPT given an **NP** oracle. Since we have an auxiliary-input one-way function on I, Lemma 4.5 gives us an instance-dependent commitment scheme for $\Gamma = (I, \overline{I})$, with public coins and a PPT sender. Combining these via Lemma 4.7, we get an instance-dependent commitment scheme for $\Pi' \cup \Gamma = \Pi$, with public coins and a sender that is PPT given an **NP** oracle. □

4.2. The zero-knowledge proof. We now show that to obtain a zero-knowledge proof for a problem $\Pi \in$ **IP**, it suffices for Π to have an instance-dependent commitment scheme in the sense of the previous section. This is done by simply using the instance-dependent commitment scheme to implement the **IP**-to-**ZK** compiler of [29, 39, 7].

LEMMA 4.8. *If a promise problem* Π *is in* **IP** *and has a computationally hiding (resp., statistically hiding) instance-dependent commitment scheme (in the sense of Definition* 4.1*), then* $\Pi \in$ **HVZK** *(resp.,* $\Pi \in$ **HVSZK***). Moreover, if the instance-dependent commitment scheme is public coin, then so is the honest-verifier zero-knowledge proof for* Π*. Also, the prover's strategy* P_x' *in the honest-verifier zero-knowledge proof is PPT given with oracles for* S_x *and* P_x*, where* S *is the sender algorithm in the instance-dependent commitment scheme and* P *is a prover in any public-coin interactive proof system for* Π*.*

Proof. We begin with the special case that $\Pi \in$ **NP**, where we follow the approach of Itoh, Ohta, and Shizuya [40] using our more general notion of instance-dependent commitments. The idea is to use the zero-knowledge proofs of Goldreich, Micali, and Wigderson [29] for all of **NP**, replacing the commitment scheme used there with the instance-dependent commitment for Π. An outline of the steps of the resulting protocol follows.

Zero-knowledge proof $(P, V)(x)$:

1. Both parties reduce x to an instance G of 3-COLORING.
2. P selects an arbitrary 3-coloring C_0 of G and lets C be the coloring obtained by permuting the three colors in C_0 uniformly at random.
3. P commits to the color of each vertex under C by engaging with V in (polynomially many executions of) the commitment phase of instance-dependent

commitment scheme for Π on common input x.

4. V selects a random edge e in G and sends e to P.
5. P reveals the colors of the endpoints of e and proves their validity to V via the reveal phase of the instance-dependent commitment scheme.
6. V accepts if the colors of the endpoints are different and accepts in both executions of the reveal phase.

Completeness follows from completeness of the instance-dependent commitment scheme. Soundness follows from the binding property of the instance-dependent commitment scheme when $x \in \Pi_N$. (Honest-verifier) zero knowledge follows from the hiding and zero-knowledge properties of the commitment scheme when $x \in \Pi_Y$. Specifically, the simulator chooses a random edge e in the graph (to be the verifier's challenge), chooses two random distinct colors for its endpoints, and assigns arbitrary colors for the rest of the graph. It uses the simulator for the commitment scheme to simulate all the commitments, using the simulated commitment phase for all the commitments, but the simulated reveal phase only for the edge e. (Note that here, unlike in [29], we deal with an honest verifier, and thus the verifier's challenge e is equivalent to its coin tosses and is indeed chosen uniformly at random.) The computational (resp., statistical) hiding property of the commitment scheme implies that this simulation is computationally (resp., statistically) indistinguishable from the (honest) verifier's view.

For the general case that $\Pi \in \mathbf{IP}$, we follow [39, 7] and transform an interactive proof (P, V) for Π into a zero-knowledge proof. By [36], we may assume that (P, V) is public coin. An outline of the zero-knowledge proof follows.

Zero-knowledge proof $(P', V')(x)$:

1. (P', V') simulate the public-coin interactive proof $(P, V)(x)$, but instead of sending P's messages explicitly, P' commits to P's messages using the commit phase of the instance-dependent commitment scheme on common input x. (The public-coin nature of (P, V) ensures that V can compute its messages without seeing P's messages explicitly.) Let $(z_1, \ldots, z_m)$ be all the commitments obtained in this way.
2. V chooses and sends a random strings $r_1, \ldots, r_m$ for the **AM** proof system for VAL.
3. Now P proves the following **NP** statement to V using the zero knowledge protocol described above (i.e., that of Goldreich, Micali, and Wigderson implemented with instance-dependent commitments): There exist values $b_1, \ldots, b_m$ such that (a) V would have accepted in the interactive proof if the prover responses were given by $b_1, \ldots, b_m$, and (b) there are prover responses $s_1, \ldots, s_m$ such that the **AM** verifier for VAL would accept on transcript $((x, z_i, b_i), r_i, s_i)$ for $i = 1, \ldots, m$.

The analysis of this proof system is similar to the previous one. The claim about the prover complexity follows by inspection. □

The above gives honest-verifier zero-knowledge proofs. These can be converted into zero-knowledge proofs that tolerate cheating verifiers using the following compiler of Goldreich, Sahai, and Vadhan [32].

THEOREM 4.9 (see [32]). *Any honest-verifier public-coin zero-knowledge proof system can be transformed into a (cheating-verifier) public-coin zero-knowledge proof system. Furthermore,*

1. *the resulting proof system has twice as many rounds as the original one.*
2. *the resulting prover strategy on any input x is PPT given oracle access to the original prover strategy on the same input x.*

3. *the resulting proof system has completeness error* $2^{-\Omega(n)}$ *and soundness error* $1/n$ *on input length* n*. In case the original proof system has perfect completeness, so does the resulting one.*
4. *if the original proof system is statistical zero knowledge, then so is the resulting proof system.*
5. *the resulting proof system has a black-box simulator.*

Note that the above theorem provides a zero-knowledge proof with a nonnegligible soundness error (namely $1/n$). This can be reduced to a negligible error by performing $\omega(1)$ sequential repetitions.

5. Instance-dependent commitments for SZK. In this section, we construct our instance-dependent commitment schemes for **SZK**, thereby proving Lemma 4.6. This is technically the most involved part of our work.

5.1. Overview. We will construct an instance-dependent commitment scheme for the **SZK**-complete problem STATISTICAL DIFFERENCE [52]. This means that we will design a commitment protocol in which both the sender and receiver get as auxiliary input a pair (X_0, X_1) of samplable distributions. The commitment scheme should be statistically hiding when X_0 and X_1 are statistically close and should be statistically binding when X_0 and X_1 are statistically far apart. By the polarization lemma of [52], we may assume, w.l.o.g., that the statistical difference between X_0 and X_1 is either exponentially small (for YES instances) or exponentially close to 1 (for NO instances).

A natural idea, suggested in [44], is the following. To commit to a bit b, the sender sends a random sample $x \leftarrow X_b$. To decommit, the sender reveals b and the coin tosses r used to generate x, and the receiver verifies that $x = X_b(r)$.

When X_0 and X_1 are statistically close, this scheme is indeed hiding. On the other hand, when $\Delta(X_0, X_1) = 1$ (i.e., X_0 and X_1 have disjoint supports), then the scheme is perfectly binding. But we are guaranteed only that $\Delta(X_0, X_1)$ is exponentially close to 1, and this does not suffice for any sort of binding. Indeed, two distributions can have statistical difference exponentially close to 1 and yet have identical supports (which means that every commitment can be opened in two ways).

To get around this difficulty, we notice that the intersection of the supports can consist of two kinds of elements. First, there can be samples that are atypically light for at least one of the distributions (i.e., have probability mass much smaller than 2^{-h}, if we assume (w.l.o.g.) that $\mathrm{H}(X_0) = \mathrm{H}(X_1) = h$). Note that there can be many ($\gg 2^h$) such elements. Second, there can be samples that are not atypically light for either distribution. However, it can be shown that there can be only a relatively few ($\ll 2^h$) elements of this second type, provided the distributions have statistical difference exponentially close to 1. Still, we need to cope with both kinds of samples.

To deal with these problems, we replace both the commit phase and reveal phase with interactive protocols. The commit phase protocol constrains the sender's choice of the sample/commitment x. Even if the sender deviates from the protocol, with high probability the commit phase will produce a sample that is atypically light for at least one of the two distributions, in which case we will regard it as a commitment to the bit corresponding to the *other* distribution. The fact that this is feasible relies on the fact that there are relatively few samples that are not atypically light for both distributions. The reveal phase protocol, then, is simply an interactive proof that the sample is not atypically light for X_b.

Needless to say, the challenge is to design both of these protocols so that the

hiding property is maintained in the case of YES instances. Fortunately, there are two protocols due to Okamoto [48] (see also [34, 57]) that turn out to be well-suited for these tasks. Specifically, we use an adaptation of Okamoto's "sample generation protocol" for the commitment phase, and his "sample test protocol" for the reveal phase. The price we pay for using these protocols is that the sender is no longer PPT (but rather in $\mathbf{BPP}^{\mathbf{NP}}$), and also that the round complexity becomes polynomial rather than being a constant.

5.2. Preprocessing the distributions. We will not apply Okamoto's protocols directly to instances of STATISTICAL DIFFERENCE itself, but rather do some preprocessing on the distributions. The first drives the thresholds from 1/3 and 2/3 to be exponentially close to 0 and 1, respectively.

LEMMA 5.1 (polarization lemma [52]). *There is a polynomial-time computable function mapping pairs of distributions (X_0, X_1) (specified by circuits which sample from them) and a unary parameter 1^k to pairs of distributions (Y_0, Y_1) such that*

$$\Delta(X_0, X_1) \leq 1/3 \Rightarrow \Delta(Y_0, Y_1) \leq 2^{-k},$$
$$\Delta(X_0, X_1) \geq 2/3 \Rightarrow \Delta(Y_0, Y_1) \geq 1 - 2^{-k}.$$

The second transformation we will use is simply taking direct products, as analyzed in section 2.2. Combining these two transformations, we prove the following.

LEMMA 5.2. *For every promise problem $\Pi \in \mathbf{SZK}$, there is a polynomial-time computable function mapping instances x of length n and unary parameters $1^k, 1^\ell$ to pairs of distributions (Z_0, Z_1) such that*

- *if $x \in \Pi_Y$, then $\Delta(Z_0, Z_1) \leq \ell \cdot 2^{-k}$.*
- *if $x \in \Pi_N$, then $\Delta(Z_0, Z_1) \geq 1 - 2^{-\ell}$.*
- *for all x, $\mathrm{H}(Z_0) = \mathrm{H}(Z_1)$ and both Z_0 and Z_1 are $\sqrt{\ell} \cdot \mathrm{poly}(n, k)$-flat.*

The key point for us is that the statistical difference in the case of NO instances goes to 1 exponentially fast with ℓ, whereas the deviation from flatness grows sublinearly with ℓ. Specifically, we can take k to be linear in n and take ℓ to be a large polynomial in n and have the deviation from flatness $\sqrt{\ell} \cdot \mathrm{poly}(n, k)$ remain sublinear in ℓ. We will show (in Lemma 5.3 below) that this implies that the intersection of the supports of the two distributions is due only to (a) atypically light elements and (b) a *small* number of other elements (i.e., much fewer than $2^{\mathrm{H}(Z_b)}$).

Proof. Let an instance x of $\Pi \in \mathbf{SZK}$ and the parameters 1^k and 1^ℓ be given. By the completeness of STATISTICAL DIFFERENCE and the polarization lemma (Lemma 5.1), we can produce in polynomial time distributions (Y_0, Y_1) such that

$$x \in \Pi_Y \Rightarrow \Delta(Y_0, Y_1) \leq 2^{-2k},$$
$$x \in \Pi_N \Rightarrow \Delta(Y_0, Y_1) \geq 1 - 2^{-2k} \geq 1/2.$$

Now, let $W_0 = Y_0 \otimes Y_1$ (i.e., a sample of Y_0 followed by an independent sample of Y_1) and $W_1 = Y_1 \otimes Y_0$. This ensures $\mathrm{H}(W_0) = \mathrm{H}(W_1)$, and we have

$$x \in \Pi_Y \Rightarrow \Delta(W_0, W_1) \leq 2 \cdot 2^{-2k},$$
$$x \in \Pi_N \Rightarrow \Delta(W_0, W_1) \geq 1/2.$$

Now we let $Z_0 = \otimes^{c \cdot \ell} W_0$ and $Z_1 = \otimes^{c \cdot \ell} W_1$, for a sufficiently large constant c. Then, by Lemma 2.3,

$$x \in \Pi_Y \Rightarrow \Delta(Z_0, Z_1) \leq c\ell \cdot 2 \cdot 2^{-2k} \leq \ell \cdot 2^{-k},$$
$$x \in \Pi_N \Rightarrow \Delta(Z_0, Z_1) \geq 1 - \exp(-\Omega(c\ell)) \geq 1 - 2^{-\ell},$$

for an appropriate constant c and sufficiently large k. Also, $\mathrm{H}(Z_0) = c\ell \cdot \mathrm{H}(W_0) = \mathrm{H}(Z_1)$. Finally, if $m = \mathrm{poly}(n,k)$ is the number of input gates to W_0 and W_1, then $\Pr[W_b = w] \geq 2^{-m}$ for all $b \in \{0,1\}$ and all w in the support of W_b, so the flattening lemma (Lemma 2.6) tells us that Z_0 and Z_1 are both $\sqrt{\ell} \cdot m$-flat. □

The following lemma shows that for two nearly flat distributions with statistical difference very close to 1, there can be only a relatively small number of strings that are nonlight for both distributions.

LEMMA 5.3. *Suppose Z_0 and Z_1 are random variables such that $\mathrm{H}(Z_0) = \mathrm{H}(Z_1)$ and $\Delta(\mathrm{H}(Z_0), \mathrm{H}(Z_1)) \geq 1 - 2^{-\ell}$. Then for any $\Phi > 0$,*

$$\#\{z : z \text{ is not } \Phi\text{-light for } Z_0 \text{ and } z \text{ is not } \Phi\text{-light for } Z_1\} \leq \frac{2^{\mathrm{H}(Z_0)}}{2^{\ell-\Phi}}.$$

Proof. Let S be the set of z that are neither Φ-light for Z_0 nor for Z_1. Then

$$\begin{aligned} 2^{-\ell} &\geq 1 - \Delta(Z_0, Z_1) \\ &= \sum_z \min\{\Pr[Z_0 = z], \Pr[Z_1 = z]\} \\ &> \sum_{z \in S} \min\left\{2^{-\Phi} \cdot 2^{-\mathrm{H}(Z_0)}, 2^{-\Phi} \cdot 2^{-\mathrm{H}(Z_1)}\right\} \\ &= |S| \cdot 2^{-\Phi} \cdot 2^{-\mathrm{H}(Z_0)}. \end{aligned}$$

Thus, $|S| < 2^{\mathrm{H}(Z_0)}/2^{\ell-\Phi}$, as desired. □

Note that the above lemma gives us a useful bound ($\ll 2^{\mathrm{H}(Z_b)}$) when the slackness parameter of Φ is smaller than ℓ. Fortunately, Lemma 5.2 allows us to obtain $\Phi = o(\ell)$ while still having a statistical difference of $1 - 2^{-\ell}$ on NO instances.

5.3. Okamoto's protocols. We now describe the two protocols of Okamoto [48] that we will use in our commitment scheme. The first is used for generating a random sample from a nearly flat distribution so that even if one party cheats, the output will be unlikely to fall in any sufficiently small set. The second is used to test that a sample from a nearly flat distribution is not too light. Our presentation of these protocols follows [34, 57].

Below, all distributions are given in the form of circuits that generate them. The input to these protocols will include a distribution, denoted X. We will denote by m (resp., n) the length of the input to (resp., output of) the circuit generating the distribution X.

DEFINITION 5.4 (sample generation protocol). *A protocol (S, R) is called a* sample generation protocol *if on common input a distribution X, specified by a circuit with m input gates and n output gates, and parameters Φ and t, such that X is Φ-flat and $1 \leq t \leq \Phi$, the protocol yields a common output in $\{0,1\}^n$ such that the following holds:*

1. *(Efficiency) R is PPT.*
2. *("Completeness") If both parties are honest, then the output of the protocol has a statistical difference of at most $m \cdot 2^{-\Omega(t^2)}$ from X.*
3. *("Soundness I") If R is honest, then no matter how S plays, the output will be $2\sqrt{t\Phi} \cdot \Phi$-heavy with probability at most $m \cdot 2^{-\Omega(t^2)}$.*
4. *("Soundness II") If R is honest, then for every set $T \subseteq \{0,1\}^n$ of size at most $2^{-6\sqrt{t\Phi}\cdot\Phi} \cdot 2^{\mathrm{H}(X)}$, no matter how S plays, the output will be in T with probability at most $m \cdot 2^{-\Omega(t^2)}$.*

5. *(Strong "zero knowledge") There exists a PPT* simulator M *so that for every* (X, Φ, t) *as above, the following two distributions have statistical difference at most* $m \cdot 2^{-\Omega(t^2)}$*:*
   (A) *Execute* (S, R) *on common input* (X, Φ, t) *and output the view of* R*, appended by the output.*
   (B) *Choose* $x \leftarrow X$ *and output* $(M(X, \Phi, t, x), x)$.

A sample generation protocol is said to be public coin *if it is public coin for* R.

In [48, 34, 57], only the first soundness condition is given, but we will actually use the second. (Our proof that the protocol satisfies the second soundness condition will make use of the first.) The above zero-knowledge property is referred to as *strong* since the simulator cannot produce a view-output pair by first generating the view and then computing the corresponding output. Instead, the simulator is forced (by the explicit inclusion of x in distribution (B)) to generate a random view consistent with a given random output (of the protocol). We comment that the trivial protocol in which R uniformly selects an input r to the circuit X and reveals both r and the output $x = X(r)$ cannot be used since the simulator is given only x, and it may be difficult to find an r yielding x in general. Still, a sample generation protocol is implicit in Okamoto's work [48] (where it is called a "pretest"). Note also that the zero-knowledge condition implies the completeness condition; still, conceptually it is convenient to state them separately.

THEOREM 5.5 (implicit in [48]; explicit in [34]). *There exists a sample generation protocol. Furthermore, the protocol is public coin, the sender strategy is PPT given an* **NP** *oracle, and the number of messages exchanged in the protocol is linear in* m*, the input length of the sampling circuit for the input distribution* X.

Actually, in [48, 34], the sample generation protocol is not shown to satisfy either the Soundness II condition of Definition 5.4 or the bound on sender complexity specified in Theorem 5.5. Thus we repeat the description of the protocol here.

Sample generation protocol (S, R):
Input: (X, Φ, t), where X has m input gates and n output gates and $t \leq \Phi$.

1. S: Select $x_0 \in \{0,1\}^n$ according to X and send x_0 to R.
2. S, R: Repeat for i from 1 to m:
   (a) R: Choose h_i uniformly from a family of pairwise independent hash functions mapping $\{0,1\}^{m+n}$ to $\{0,1\}^{m-3t\Phi}$ and send h_i to S.
   (b) S: Choose (r_{i-1}, x_i) from the distribution $\{r : X(r) = x_{i-1}\} \otimes X$, conditioned on $h(r_{i-1}, x_i) = 0$, and send (r_{i-1}, x_i) to R. (If there is no such pair (r, x'), then S sends `fail` to R.)
   (c) R: Check that $X(r_{i-1}) = x_{i-1}$ and $h(r_{i-1}, x_i) = 0$. If either condition fails, reject.

Output: x_m, unless R rejects in some iteration of the above loop, in which case output any canonical string outside $\{0,1\}^n$, e.g., 0^{n+1}.

The sender complexity claimed in Theorem 5.5 follows from the observation that the sender need only sample strings uniformly from efficiently decidable sets (i.e., satisfying assignments to a known, polynomial-sized circuit), and it is known how to do such sampling given an **NP** oracle [41, 5].

LEMMA 5.6. *The above protocol satisfies the Soundness* II *condition of Definition* 5.4.

Proof. Fix a set T of size at most $2^{-6\sqrt{t\Phi}\cdot\Phi} \cdot 2^{\mathrm{H}(X)}$. We need to show that the output x_m is in T with probability at most $m \cdot 2^{-\Omega(t^2)}$, even under a cheating strategy for S. The Soundness I condition says that x_m is $2\sqrt{t\Phi} \cdot \Phi$-heavy with probability

at most $m \cdot 2^{-\Omega(t^2)}$. In fact, the proof of this condition in [34] also shows that x_{m-1} is $2\sqrt{t\Phi} \cdot \Phi$-heavy with probability at most $m \cdot 2^{-\Omega(t^2)}$. (Indeed, the protocol could have been terminated after $m-1$ or even slightly fewer stages, but m was chosen as a clean upper bound on the number of stages needed.) We will show that if x_{m-1} is not $2\sqrt{t\Phi} \cdot \Phi$-heavy, then the probability (over h_m) that S can select x_m to be in T (without R rejecting) is at most $2^{-\Omega(t^2)}$.

The number N of strings r_{m-1} such that $X(r_{m-1}) = x_{m-1}$ is

$$N = 2^m \cdot \Pr[X = x_{m-1}] < 2^m \cdot 2^{2\sqrt{t\Phi}\cdot\Phi} \cdot 2^{-\mathrm{H}(X)},$$

where the inequality is due to x_{m-1} not being $2\sqrt{t\Phi} \cdot \Phi$-heavy. Thus, the number of pairs (r_{m-1}, x_m) such that $X(r_{m-1}) = x_{m-1}$ and $x_m \in T$ equals

$$N \cdot |T| = \left(2^m \cdot 2^{2\sqrt{t\Phi}\cdot\Phi} \cdot 2^{-\mathrm{H}(X)}\right) \cdot \left(2^{-6\sqrt{t\Phi}\cdot\Phi} \cdot 2^{\mathrm{H}(X)}\right) \leq 2^{-t^2} \cdot 2^{m-3t\Phi},$$

where the last inequality uses $t \leq \Phi$. Since $h_m(z)$ is uniformly distributed in its range $\{0,1\}^{m-3t\Phi}$ for every z, the probability that there exists a pair (r_{m-1}, x_{m-1}) such that $h_m(r_{m-1}, x_{m-1}) = 0$ is at most 2^{-t^2}. □

The second protocol tests whether a sample is too light; here we do not need any modifications from the definition in [34].

DEFINITION 5.7 (sample test protocol). *A protocol (S, R) is called a* sample test protocol *if on common input a distribution X, specified by a circuit with m input gates and n output gates, a string $x \in \{0,1\}^n$, and parameters Φ, t, such that X is Φ-flat and $t \leq \Phi$, the following hold:*

1. *(Efficiency) R is PPT.*
2. *("Completeness") If both parties are honest and x is $t \cdot \Phi$-typical, then R accepts with probability at least $1 - m \cdot 2^{-\Omega(t^2)}$.*
3. *("Soundness") If x is $6\sqrt{t\Phi} \cdot \Phi$-light and R is honest, then no matter how S plays, R accepts with probability at most $m \cdot 2^{-\Omega(t^2)}$.*
4. *(Weak "zero knowledge") There exists a PPT* simulator *M so that for every (X, Φ, t) as above and for every $t \cdot \Phi$-typical x, the following two distributions have statistical difference at most $m \cdot 2^{-\Omega(t^2)}$:*
   (A) *Execute (S, R) on common input (X, x, Φ, t), and output the view of R.*
   (B) *Choose r uniformly in $\{r' : X(r') = x\}$, and output $M(X, x, \Phi, t, r)$.*

A sample test protocol is said to be public coin *if it is public coin for R.*

The above zero-knowledge property is referred to as *weak* since the simulator gets a random r giving rise to x (i.e., $x = X(r)$) as an auxiliary input (whereas R is given only x). A sample test protocol is implicit in Okamoto's work [48] (where it is called a "posttest").

THEOREM 5.8 (implicit in [48]; explicit in [34]). *There exists a public-coin sample test protocol. Furthermore, the protocol is public coin, the sender strategy is computable in PPT with an* **NP** *oracle, and the number of messages exchanged in the protocol is linear in m.*

5.4. The commitment scheme. Now we use the above protocols to design instance-dependent commitments for all of **SZK**, and thereby prove Lemma 4.6. Let Π be a promise problem in **SZK**, let x be any string of length n, and let $k = 2n$, and $\ell = n^{7c}$ for a sufficiently large constant c to be determined later. Applying the reduction of Lemma 5.2 to x, we obtain distributions (Z_0, Z_1) such that

- if $x \in \Pi_Y$, then $\Delta(Z_0, Z_1) \leq \ell \cdot 2^{-k} < 2^{-n}$.

- if $x \in \Pi_N$, then $\Delta(Z_0, Z_1) \geq 1 - 2^{-\ell}$.
- for all x, it holds that $\mathrm{H}(Z_0) = \mathrm{H}(Z_1)$ and both Z_0 and Z_1 are Φ-flat for $\Phi = \sqrt{\ell} \cdot \mathrm{poly}(n,k) < n^{4c}$, when c is sufficiently large.

Now we also define a new distribution Z as follows $Z(b,r) = Z_b(r)$. That is, Z outputs a random sample of Z_0 with probability 1/2 and a random sample of Z_1 with probability 1/2. Since $\mathrm{H}(Z_0) = \mathrm{H}(Z_1)$, we have $\mathrm{H}(Z_0) \leq \mathrm{H}(Z) \leq \mathrm{H}(Z_0) + 1$. We also claim that Z inherits the flatness of Z_0 and Z_1.

CLAIM 5.9. *Z is 3Φ-flat.*

The tedious proof of this claim is deferred to Appendix A. Now we construct the instance-dependent commitment scheme (S, R) as follows, setting $t = n$:

Commit phase $(S_1(b), R_1)(x)$:

1. S_1 and R_1 execute the sample generation protocol of Theorem 5.5 on input $(Z, 3\Phi, t)$ to obtain output z, where Z, Φ, and t are as defined above.
2. S_1 chooses (c,r) uniformly s.t. $Z(c,r) = z$ and sends $d = b \oplus c$ to R.
3. The commitment is defined as the pair (z,d).

(Intuitively, if Z_0 and Z_1 are statistically close, then a random sample z of Z is nearly equally likely to have come from Z_0 or Z_1, so the bit c is random and hides b.)

Valid commitments: The promise problem of valid commitments is defined to be $\mathrm{VAL} = (\mathrm{VAL}_Y, \mathrm{VAL}_N)$, where

$$\mathrm{VAL}_Y = \{(x,(z,d),b) : z \text{ is } t\Phi\text{-typical for } Z_{d \oplus b}\},$$
$$\mathrm{VAL}_N = \{(x,(z,d),b) : z \text{ is } 6\sqrt{t\Phi} \cdot \Phi\text{-light for } Z_{d \oplus b}\}.$$

Reveal phase $(S_2, R_2)(x,(z,d),b)$: S_2 and R_2 execute the sample test protocol of Theorem 5.8 on the input $(Z_{d \oplus b}, z, \Phi, t)$, and R_2 accepts or rejects according to its outcome.

CLAIM 5.10. *The above protocol is a statistically hiding instance-dependent commitment scheme in the sense of Definition* 4.1.

Proof.

1. (Receiver's efficiency) This follows from the efficiency of the sample generation and sample test protocols.
2. (Completeness) By the completeness of the sample generation protocol, the string z generated in the $(S_1(b), R_1)(x)$ has statistical difference at most $m \cdot 2^{-t^2} < 2^{-n}$ from Z, where $m = \mathrm{poly}(n)$ is the number of input gates of the circuit generating Z. Thus, (c,r) has statistical difference at most 2^{-n} from uniform. If (c,r) were uniformly distributed, then by the Φ-flatness of $Z_{c \oplus d}$, the probability (over r) that $z = Z_c(r)$ is $t\Phi$-typical for $Z_c = Z_{d \oplus b}$ is at least $1 - 2^{-t^2} > 1 - 2^{-n}$. Therefore, $(x,(z,d),b) \in \mathrm{VAL}_Y$ with probability at least $1 - 2 \cdot 2^{-n}$.
3. (Validity tests) The completeness and soundness of the sample test protocol show that (S_2, R_2) is an interactive proof system for VAL. To show that VAL is in **AM**, we design an **AM** proof system for it as follows. On input $(x,(z,d),b)$, the prover sends an approximation k to $\mathrm{H}(Z_{d \oplus b})$, and then proves that (a) $\mathrm{H}(Z_{d \oplus b}) \lesssim k$, and (b) $|\{r : Z_{d \oplus b}(r) = z\}| \gtrsim 2^m \cdot 2^{-k - t\Phi}$, where again m is the number of input gates of the circuit generating Z. Step (a) can be done because approximating entropy to within an additive constant (say ± 1)

is in **AM** $\cap$ **co-AM** [51].[13] Step (b) can be done using an **AM** protocol for proving approximate lower bounds on the sizes of efficiently recognizable sets [55, 56, 2]. Proving (a) and (b) suffices because on YES instances of VAL, we have

$$|\{r : Z_{d\oplus b}(r) = z\}| \geq 2^m \cdot 2^{-\mathrm{H}(Z_{d\oplus b})-t\Phi},$$

and on NO instances, we have

$$|\{r : Z_{d\oplus b}(r) = z\}| \leq 2^m \cdot 2^{-\mathrm{H}(Z_{d\oplus b})-6\sqrt{t\Phi}\cdot\Phi} < 2^m \cdot 2^{-\mathrm{H}(Z_{d\oplus b})-t\Phi-5}.$$

4. (Zero knowledge) This follows from the zero-knowledge conditions of the sample generation and sample test protocols. Specifically, the simulator $M(x, b)$ chooses a uniformly random (c, r), sets $z = Z(c, r)$ and $d = b \oplus c$, runs the simulator for the sample generation protocol on input $(Z, 3\Phi, t, z)$ to obtain a transcript γ_1, runs the simulator for the sample test protocol on $(Z_c, z, \Phi, t, (c, r))$ to obtain a transcript γ_2, and outputs (γ_1, d, γ_2).
5. (Statistically hiding on YES instances) The only dependence of R_1's view on the bit b is in the value $d = b \oplus c$, where c is selected according to the conditional distribution $C|_{Z_C=z}$, where C is uniform in $\{0, 1\}$. We have seen above that the sample generation protocol generates z according to a distribution that has a statistical difference of at most 2^{-n} from a random sample of $Z \equiv Z_C$. Thus, the pair (z, c) is generated according to a distribution having a statistical difference of at most 2^{-n} from (Z_C, C). In the case of a YES instance, where Z_0 and Z_1 have a statistical difference of at most 2^{-n}, (Z_C, C) has a statistical difference of at most 2^{-n} from (Z_C, C'), where C' is a random bit independent of C. Thus, R_1's view in case $b = 0$ is statistically indistinguishable from R_1's view in case $b = 1$.
6. (Statistically binding on NO instances) Let

$$T = \{z : z \text{ is not } 6\sqrt{t\Phi} \cdot \Phi\text{-light for } Z_0 \text{ or for } Z_1\}.$$

By Lemma 5.3,

$$|T| \leq \frac{2^{\mathrm{H}(Z_0)}}{2^{\ell-6\sqrt{t\Phi}\cdot\Phi}} \leq 2^{-6\sqrt{t\Phi}\cdot\Phi} \cdot 2^{\mathrm{H}(Z)},$$

where the last inequality holds because $\ell = n^{7c} > 12n^{6c+.5} > 12\sqrt{t\Phi} \cdot \Phi$. By the second soundness condition of the sample generation protocol, the probability that the output z is in T is at most $2^{-\Omega(t^2)} < 2^{-n}$. If the output is not in T, then for any d, there is at most one value of b such that z is not $6\sqrt{t\Phi} \cdot \Phi$-light for $Z_{d\oplus b}$. That is, there is at most one value of b such that $(x, (z, d), b) \notin \mathrm{VAL}_N$, as desired. □

Proof of Lemma 4.6. Using Claim 5.10, all that is left to verify is that the protocol is public coin, and the sender is PPT given an **NP** oracle. Both of these follow from the analogous properties of the sample generation and sample test protocols given in Theorems 5.5 and 5.8. □

[13] Indeed, the promise problem ENTROPY APPROXIMATION (EA), where $\mathrm{EA}_Y = \{(X, k) : \mathrm{H}(X) \geq k + 1\}$, $\mathrm{EA}_N = \{(X, k) : \mathrm{H}(X) \leq k\}$, is complete for noninteractive statistical zero knowledge (**NISZK**) [33], and **NISZK** $\subseteq$ **SZK** $\subseteq$ **AM** $\cap$ **co-AM** [18, 1].

6. Putting it together. Now we can put together the results proved in the previous three sections and establish Theorems 1.2, 3.4, 3.6, and 4.2.

THEOREM 6.1 (**ZK** characterization theorem). *For a promise problem* Π*, the following conditions are equivalent:*

1. $\Pi \in \mathbf{HVZK}$.
2. $\Pi \in \mathbf{IP}$*, and* Π *satisfies the* CONDITIONAL PSEUDOENTROPY CONDITION.
3. $\Pi \in \mathbf{IP}$*, and* Π *satisfies the* SZK/OWF CONDITION.
4. $\Pi \in \mathbf{IP}$*, and* Π *satisfies the* INDISTINGUISHABILITY CONDITION.
5. $\Pi \in \mathbf{IP}$*, and* Π *has a public-coin computationally hiding instance-dependent commitment scheme in the sense of Definition* 4.1*. Moreover, the sender is PPT given an* **NP** *oracle.*
6. $\Pi \in \mathbf{ZK}$.
7. Π *has a public-coin computational zero-knowledge proof with a black-box simulator and perfect completeness.*
8. Π *has a public-coin computational zero-knowledge proof with a black-box simulator, where on any input* x*, the prover strategy* P_x *is PPT given an* **NP** *oracle and an oracle for* $\hat{P}_x$*, where* $\hat{P}$ *is the prover in any interactive proof system for* Π*. In particular, if* $\Pi \in \mathbf{NP}$ *(or even* $\Pi \in \mathbf{AM}$*), then* P_x *is PPT given an* **NP** *oracle.*

Proof.

1 ⇒ 2 This follows from Lemma 3.7, together with the trivial inclusion $\mathbf{HVZK} \subseteq \mathbf{IP}$.

2 ⇒ 3 This is Lemma 3.10.

3 ⇒ 5 This is Lemma 4.4.

5 ⇒ 7 Suppose $\Pi \in \mathbf{IP}$, and that Π has a public-coin instance-dependent commitment scheme. By Lemma 4.8, Π has a public-coin honest-verifier zero-knowledge proof. We can convert this into a public-coin proof system with perfect completeness using the transformation of Fürer et al. [19], which preserves honest-verifier zero knowledge. Finally, by Theorem 4.9, this can be converted into a public-coin (cheating-verifier) zero-knowledge proof with a black-box simulator and perfect completeness.

5 ⇒ 8 This is proved the same way as in the previous item, except we omit the transformation of Fürer et al. [19] (which seems to increase the prover complexity beyond $\mathbf{BPP}^{\mathbf{NP}}$). For bounding the prover complexity, we first note that if $\Pi \in \mathbf{IP}$, then a (variant of) the Goldwasser–Sipser [36] transformation converts any interactive proof $(\hat{P}, \hat{V})$ for Π into a public-coin interactive proof, where the prover on input x is PPT given an **NP** oracle and oracle access to $\hat{P}_x$. Then Lemma 4.8 preserves this prover complexity because the sender in the instance-dependent commitment is PPT given with an **NP** oracle. The same holds for Theorem 4.9.

7/8 ⇒ 6 ⇒ 1 These are immediate from the definitions.

2 ⇔ 4 This is by Lemmas 3.13 and 3.14. □

We also prove Theorem 4.3, which we restate here.

THEOREM 6.2 (Theorem 4.3, restated). $\Pi \in \mathbf{SZK}$ *if and only if* $\Pi \in \mathbf{IP}$ *and* Π *has a* statistically *hiding instance-dependent commitment scheme in the sense of Definition* 4.1*.*

Proof.

⇒ This follows from Lemma 4.6, together with the trivial inclusion $\mathbf{SZK} \subseteq \mathbf{IP}$.

⇐ This follows from Lemma 4.8, together with the fact that $\mathbf{HVSZK} = \mathbf{SZK}$ [48, 32]. □

7. Applications and extensions.

7.1. The Ostrovsky–Wigderson theorems. As described in the introduction, the approach of this paper, and in particular the SZK/OWF characterization theorem, are inspired by the work of Ostrovsky and Wigderson [50], who showed that "nontriviality" of **ZK** implies "some form of one-way functions." In this section, we show how our results can be used to give new, more modular proofs of the Ostrovsky–Wigderson theorems. Specifically, we use the SZK/OWF characterization theorem to deduce the Ostrovsky–Wigderson theorems about **ZK** from the earlier (and much simpler) work of Ostrovsky [49] on **SZK**. In fact, we need only our results from section 3, showing that every problem in **HVZK** satisfies the SZK/OWF CONDITION. Our results in the converse direction, from sections 4, 5, and 6, are not needed.

The two Ostrovsky–Wigderson theorems are obtained by two different interpretations of "nontriviality" and "some form of one-way functions." In their first theorem (mentioned in the introduction), both are interpreted in a weak sense as follows.

THEOREM 7.1 (see [50, Thm. 1]). *If* $\mathbf{HVZK} \neq \mathbf{BPP}$*, then there exists a poly-time auxiliary-input family of functions* $\{f_x : \{0,1\}^{p(|x|)} \to \{0,1\}^{q(|x|)}\}$ *that is not "easy to invert." That is, for every PPT* A *and every polynomial* $r(n)$*, there exists an infinite set* $I \subseteq \{0,1\}^*$ *such that*

$$\Pr\left[A(x, f_x(U_{p(|x|)})) \in f_x^{-1}(f_x(U_{p(|x|)}))\right] \leq 1/r(|x|)$$

for all $x \in I$*.*

We point out that the theorem above refers to *uniform* PPT inverters A; to obtain functions that are not easy to invert by nonuniform algorithms, the hypothesis should be replaced with $\mathbf{HVZK} \not\subset \mathbf{P/poly}$.

In their second theorem, both conditions are interpreted in a strong sense as follows.

DEFINITION 7.2. *A promise problem* Π *is* hard on average *if there exists a PPT sampling algorithm* S*, a polynomial* r*, and a constant* $\delta > 0$ *such that for every nonuniform PPT* A*, the following holds for all but finitely many* n*:*

$$\Pr_{x \leftarrow S(1^n)}\left[(x \in \Pi_Y \cup \Pi_N) \wedge (A(x) \neq \chi_\Pi(x)) \wedge |x| \geq n^\delta\right] \geq \frac{1}{r(n)},$$

where χ_Π *is the characteristic function of* Π*, i.e.,* $\chi_\Pi(x) = 1$ *if* $x \in \Pi_Y$*,* $\chi_\Pi(x) = 0$ *if* $x \in \Pi_N$*, and* $\chi_\Pi(x) = \star$ *otherwise.*

THEOREM 7.3 (see [50, Thm. 2]). *If* **HVZK** *contains a hard-on-average promise problem, then (standard) one-way functions exist.*

We begin by observing that the SZK/OWF characterization (Theorem 1.2) immediately implies a stronger form of one-way functions than given by Theorem 7.1 under the stronger (but still worst-case) hypothesis that $\mathbf{HVZK} \neq \mathbf{HVSZK}$.

THEOREM 7.4. *If* $\mathbf{HVZK} \neq \mathbf{HVSZK}$*, then there exists an auxiliary-input one-way function on some infinite set* I*. That is, there is a poly-time auxiliary-input family of functions* $\{f_x : \{0,1\}^{p(|x|)} \to \{0,1\}^{q(|x|)}\}$ *and an infinite set* I *such that for every nonuniform PPT* A *and every polynomial* $r(n)$*, we have*

$$\Pr\left[A(x, f_x(U_{p(|x|)})) \in f_x^{-1}(f_x(U_{p(|x|)}))\right] \leq 1/r(|x|)$$

for all sufficiently long $x \in I$*.*

The key difference between the conclusions of Theorems 7.1 and 7.4 is that the order of quantifiers between the adversary A and the infinite set I is reversed. In

the former, the infinite set of indices x for which the adversary fails to invert f_x can depend on the adversary A, whereas in the latter, there is a fixed infinite set of indices such that f_x is hard for *all* polynomial-time adversaries A.

Recall that $\mathbf{HVSZK} \subseteq \mathbf{AM} \cap \mathbf{co\text{-}AM}$ [18, 1], and thus it is unlikely that $\mathbf{NP} \subseteq \mathbf{HVSZK}$. Thus Theorem 7.4 can be interpreted as further evidence, incomparable to what is given by the Ostrovsky–Wigderson theorems (Theorems 7.1 and 7.3), that one-way functions are necessary to construct zero-knowledge proofs for all of NP (not to mention all of **IP**). (Recall that it is known that one-way functions are *sufficient* to establish that $\mathbf{IP} = \mathbf{ZK}$ [29, 39, 7, 45, 37].)

Proof of Theorem 7.4. Suppose $\mathbf{HVZK} \neq \mathbf{HVSZK}$, and let Π be any promise problem in $\mathbf{HVZK} \setminus \mathbf{HVSZK}$. By Theorem 6.1, Π satisfies the SZK/OWF Condition. That is, there is a set I such that $\Pi' = (\Pi_Y \setminus I, \Pi_N)$ is in **SZK** and there exists an auxiliary-input one-way function on I. We claim that I is infinite (which suffices to complete the proof). Suppose for the sake of contradiction that I is finite. Since $\Pi' \in \mathbf{SZK}$ and Π and Π' differ on only a finite set of inputs, we conclude that $\Pi \in \mathbf{SZK} \subseteq \mathbf{HVSZK}$. (The statistical zero-knowledge proof for Π is the same as the statistical zero-knowledge proof for Π', except we hardwire the set I into the verifier and simulator, have the verifier immediately accept inputs $x \in I$, and have the prover send nothing on such inputs.) This contradicts the choice of Π. ∎

We now give alternate proofs of the Ostrovsky–Wigderson theorems themselves based on the work of Ostrovsky on **SZK**, as captured in the following theorem.

Theorem 7.5 (implicit in Ostrovsky [49]). *For every problem $\Pi \in \mathbf{HVSZK}$, there exists a poly-time auxiliary-input function ensemble $\mathcal{F} = \{f_x : \{0,1\}^{p(|x|)} \to \{0,1\}^{q(|x|)}\}_{x \in \{0,1\}^*}$, a probabilistic polynomial-time oracle machine M, and a negligible function ϵ such that for every $x \in \Pi_Y \cup \Pi_N$, every $t \in \mathbb{N}$, and every function $A : \{0,1\}^{q(|x|} \to \{0,1\}^{p(|x|)}$, we have*

$$\begin{aligned} &\Pr\left[A(f_x(U_{p(|x|)})) \in f_x^{-1}(f_x(U_{p(|x|)}))\right] > \epsilon(|x|) + \frac{1}{t} \\ &\Rightarrow \Pr\left[M^A(x, 1^t) = \chi_\Pi(x)\right] \geq 1 - 2^{-|x|}, \end{aligned}$$

where χ_Π is again the characteristic function of Π.

Note that t, which specifies A's success probability in inverting f_x (up to a negligible term), is given as an input (in unary) to the oracle machine M. Intuitively, for M to take advantage of the fact that A inverts f_x with probability $\approx 1/t$, M must be allowed running time polynomially related to t.

Proof of Theorem 7.1. Suppose that $\mathbf{HVZK} \neq \mathbf{BPP}$. Then either $\mathbf{HVZK} \neq \mathbf{HVSZK}$ or $\mathbf{HVSZK} \neq \mathbf{BPP}$. In the first case, we are done by Theorem 7.4. Thus, we need only show that $\mathbf{HVSZK} \neq \mathbf{BPP}$ implies the existence of an auxiliary-input family of functions that is not easy to invert. This follows readily from Theorem 7.5. Let Π be any promise problem in $\mathbf{HVSZK} \setminus \mathbf{BPP}$, and let $\{f_x\}$ be the family of functions provided by Theorem 7.5. If there is a uniform PPT A inverting f_x with probability at least $1/r(|x|)$, for some polynomial r and all but finitely many x, then by Theorem 7.5, $M^{A(x,\cdot)}(x, 1^{2r(|x|)})$ is a PPT algorithm that decides Π correctly for all but finitely many x.[14] This contradicts the assumption that $\Pi \notin \mathbf{BPP}$. ∎

[14] A minor technicality is that Theorem 7.5 is stated for deterministic oracles A, whereas here A may be probabilistic. However, after a standard error reduction obtained by $O(r(|x|))$ repeated trials, we can ensure that with probability .99 over A's coin tosses w, the deterministic algorithm $A(x, \cdot; w)$ inverts $f_x(U_{p(|x|)})$ with probability $(.75) \cdot (1/r(|x|))$. So we obtain a **BPP** algorithm for Π by randomly choosing w and running $M^{A(x,\cdot,w)}(x, 1^{2r(|x|)})$.

The above proof illustrates why Theorem 7.1 yields only a family of functions that is not easy to invert, rather than the stronger notion of auxiliary-input one-way functions achieved in Theorem 7.4. The reason is that the supposed inverter A for the family of functions is used to construct a **BPP** algorithm for the promise problem $\Pi \in \mathbf{SZK}$. The hypothesis that $\mathbf{SZK} \neq \mathbf{BPP}$ seems to guarantee only that for every inverter A there exists an infinite set I_A of instances on which this procedure fails, not that there exists a fixed infinite set I of "hard" instances on which the procedure fails for any A. For example, an inverter A running in time n^2 may be able to succeed on a larger set of instances than an inverter running in time n, and one running in time n^3 may succeed on an even larger set of instances, and so on. Ultimately, the set of instances which are hard for *all* polynomial-time A may be empty.

How is this difficulty avoided in Theorem 7.4, which relies on the SZK/OWF CONDITION as established in section 3? Intuitively, the reason is that the hardness of inverting the function f_x of the SZK/OWF CONDITION when x is an "OWF instance" is not derived from the intractability of the promise problem Π, which does not make sense for fixed instances x (for the reasons discussed above), but rather is based on the intractability of distinguishing the output of the simulator from the real interaction in an **HVZK** proof system (which makes sense for fixed instances x and indeed is required to hold for every $x \in \Pi_Y$).

Theorem 7.3 gets around this difficulty in a different way by requiring a stronger form of intractability for the problem Π, namely, that it is hard on average. Let us first consider the case that we have a hard-on-average problem $\Pi \in \mathbf{HVSZK}$, following Ostrovsky [49]. Instead of hoping that x's membership in Π will be hard to decide, and thus that f_x from Theorem 7.5 will be hard to invert for particular values of x, we simply can sample a random instance x and be guaranteed, by the definition of "hard on average," that for *any* polynomial-time algorithm A, the instance x will be "hard" for A with at least a fixed nonnegligible probability. Thus $f(x,y) = (x, f_x(y))$ will be hard to invert with a fixed nonnegligible probability for any polynomial-time inverter. Now to handle the more general case of $\Pi \in \mathbf{HVZK}$, we use the SZK/OWF CONDITION, combining Ostrovsky's one-way functions just described, which are hard to invert in the case that x is an "SZK instance," with the one-way functions of the SZK/OWF CONDITION, which are hard to invert in the case that x is an "OWF instance." This yields the following new proof of Theorem 7.3.

Proof of Theorem 7.3. Suppose $\Pi \in \mathbf{HVZK}$ is hard-on-average with respect to the sampling algorithm S. Theorem 6.1 tells us that Π satisfies the SZK/OWF CONDITION, so there is a set $I \subseteq \Pi_Y$ and a poly-time auxiliary-input function ensemble $\mathcal{F} = \{f_x : \{0,1\}^{p(|x|)} \to \{0,1\}^{q(|x|)}\}$ such that $\Pi' = (\Pi_Y \setminus I, \Pi_N)$ is in $\mathbf{SZK} = \mathbf{HVSZK}$ and $\mathcal{F}$ is one-way on I. We now apply Theorem 7.5 to Π' to get another poly-time auxiliary-input function ensemble $\mathcal{F}' = \{f'_x : \{0,1\}^{p'(|x|)} \to \{0,1\}^{q'(|x|)}\}$ such that any inverter for f'_x can be used to decide whether x is a YES or NO instance of Π'.

Now we construct a one-way function g_n, where n is the security parameter, as follows: The input to g_n is a triple (r, w, w'). To compute $g_n(r, w, w')$, we interpret r as coin tosses for the sampling algorithm S, obtaining an instance $x = S(1^n; r)$ of Π, and output $(x, f_x(w), f'_x(w'))$.

We will now argue that g_n is a *weak* one-way function, namely, that no nonuniform PPT algorithm can invert g_n with probability higher than $1 - 1/(4r(n))$, where r is the polynomial in the definition of hard on average. Suppose that there is a nonuniform PPT inverter A such that

$$\Pr\left[A(g_n(R,W,W')) \in g_n^{-1}(g_n(R,W,W'))\right] \geq 1 - 1/(4r(n))$$

for infinitely many n, when R, W, and W' are chosen uniformly at random from the bit-strings of appropriate length. (Since A is nonuniform, we may assume that it is deterministic w.l.o.g.)

First, we note that when $X = S(1^n; R) \in I$, then A has only a negligible probability of inverting over the choice of W, by the one-wayness of f_X. Thus, we have

$$\Pr\left[A(g_n(R, W, W')) \in g_n^{-1}(g_n(R, W, W')) \wedge S(1^n; R) \notin I\right] \geq 1 - 1/(3r(n)). \tag{2}$$

Now we use Theorem 7.5 to convert A into an algorithm B that decides Π', and hence Π, well on average (with respect to the distribution $S(1^n)$). Specifically, on input x, B chooses w uniformly at random and runs $M^{A(x, f_x(w), \cdot)_3}(x, 1^{|x|})$, where δ is the constant in the definition of hard on average and $A(x, f_x(w), \cdot)_3$ denotes the third component of the output of A.

From (2), it follows that with probability at least $1 - 2/(3r(n))$ over the choices of $r \leftarrow R$ and $w \leftarrow W$, we have $x = S(1^n; r) \notin I$ and

$$\begin{aligned} &\Pr\left[A(x, f_x(w), f'_x(W'))_3 \in (f'_x)^{-1}(f'_x(W'))\right] \\ &\geq \Pr\left[A(g_n(r, w, W')) \in g_n^{-1}(g_n(r, w, W'))\right] \geq 1/2, \end{aligned} \tag{3}$$

where the probabilities are taken only over W'. Whenever inequality (3) holds and we have $x \in \Pi'_Y \cup \Pi'_N = (\Pi_Y \cup \Pi_N) \setminus I$, Theorem 7.5 ensures that $M^{A(x, f_x(w), \cdot)_3}(x, 1^{|x|})$ correctly decides whether x is a YES or NO instance of Π' with probability at least $1 - 2^{-|x|}$. Thus, setting $X = S(1^n; R)$, we have

$$\Pr\left[(X \in \Pi_Y \cup \Pi_N) \wedge (B(X) \neq \chi_\Pi(X)) \wedge (|X| \geq n^\delta\right] \leq 2/(3r(n)) + 2^{-n^\delta} < 1/r(n).$$

This contradicts the fact that Π is hard on average with respect to the distribution $S(1^n)$. □

We note that an alternative way to prove a version of Theorem 7.3 is to combine our results with [52, Thm. 5.12], which shows that if a hard-on-average problem satisfies the INDISTINGUISHABILITY CONDITION, then one-way functions exist. However, [52, Thm. 5.12] uses a stronger definition of hard on average than Definition 7.2, requiring that any PPT algorithm has error probability negligibly close to 1/2, rather than just $1/\text{poly}(n)$. In addition, we feel that it is informative to see how the result for **ZK** follows from combining Ostrovsky's work on **SZK** (i.e., Theorem 7.5) with the SZK/OWF
CONDITION.

7.2. Monotone closure. In this section, we use our results to prove closure properties of **ZK**. We begin by noting that the fact that **ZK** is closed under intersection is immediate: To prove that $x \in \Pi_Y \cap \Gamma_Y$ for promise problems $\Pi, \Gamma \in \mathbf{ZK}$, the prover can prove that $x \in \Pi_Y$ using the zero-knowledge proof for Π and then prove that $x \in \Gamma_Y$ using the zero-knowledge proof for Γ, and the verifier accepts only if both proofs are convincing. The analogous approach for union, however, does not work. In particular, proving that $x \in \Pi_Y \cup \Gamma_Y$ seems to require the prover to reveal whether $x \in \Pi_Y$ or $x \in \Gamma_Y$, and thus the proof system may not be zero knowledge.

In this section, we show **ZK** is indeed closed under union. More generally, for every $\Pi \in \mathbf{ZK}$, we give zero-knowledge proofs for arbitrary monotone Boolean formulae over statements about membership in Π, where the formula can even be specified as part of the common input. Such closure properties were previously known for **SZK**

[14, 48, 52].[15] Indeed we prove our results by reduction to the **SZK** case via the SZK/OWF characterization theorem. (An alternative way of proving the results is to mimic the proofs for **SZK**, replacing the STATISTICAL DIFFERENCE in the construction of [52] with the INDISTINGUISHABILITY CONDITION.)

THEOREM 7.6. **ZK** *is closed under union.*

Proof. By Theorem 1.2, a promise problem is in **ZK** if and only if it is in **IP** and it satisfies the SZK/OWF CONDITION. Since **IP** is closed under union, it suffices to show that the class of problems satisfying the SZK/OWF CONDITION is closed under union.

Suppose that Π and Γ satisfy the SZK/OWF CONDITION. Then there are sets I and J and poly-time auxiliary-input families of functions $\{f_x\}, \{g_x\}$ such that $\Pi' = (\Pi_Y \setminus I, \Pi_N)$ and $\Gamma' = (\Gamma_Y \setminus J, \Gamma_N)$ are both in **SZK**, f_x is one-way when $x \in I$, and g_x is one-way when $x \in J$. We claim that the set $K = I \cup J$ of "OWF instances" and the family of functions $\{h_x\}$, where $h_x(y,z) = (f_x(y), g_x(z))$, meet the requirements for showing that $\Pi \cup \Gamma$ satisfies the SZK/OWF CONDITION. Indeed, when $x \in K$, then h_x is one-way because either f_x or g_x is one-way. The promise problem $((\Pi_Y \cup \Gamma_Y) \setminus K, \Pi_N \cap \Gamma_N)$ is in **SZK** because it is a restriction of the promise problem $\Pi' \cup \Gamma' = (\Pi'_Y \cup \Gamma'_Y, \Pi'_N \cap \Gamma'_N)$ (i.e., the YES instances of the former problem are a subset of those of the latter, and the NO instances of both problems are the same), and $\Pi' \cup \Gamma'$ in **SZK** because **SZK** is closed under union [48]. □

We now present some definitions (closely following [52]) to formalize the more general monotone closure properties we will obtain. Specifically, in order to deal with instances of promise problems that violate the promise, we will work with an extension of Boolean algebra that includes an additional "ambiguous" value $\star$.

DEFINITION 7.7. *A* partial assignment *to variables $v_1, \ldots, v_k$ is a k-tuple $\overline{a} = (a_1, \ldots, a_k) \in \{0, 1, \star\}^k$. For a propositional formula (or circuit) ϕ on variables $v_1, \ldots, v_k$, the evaluation $\phi(\overline{a})$ is recursively defined as follows:*

$$v_i(\overline{a}) = a_i, \qquad (\phi \wedge \psi)(\overline{a}) = \begin{cases} 1 & \text{if } \phi(\overline{a}) = 1 \text{ and } \psi(\overline{a}) = 1, \\ 0 & \text{if } \phi(\overline{a}) = 0 \text{ or } \psi(\overline{a}) = 0, \\ \star & \text{otherwise,} \end{cases}$$

$$(\neg\phi)(\overline{a}) = \begin{cases} 1 & \text{if } \phi(\overline{a}) = 0, \\ 0 & \text{if } \phi(\overline{a}) = 1, \\ \star & \text{if } \phi(\overline{a}) = \star, \end{cases} \qquad (\phi \vee \psi)(\overline{a}) = \begin{cases} 1 & \text{if } \phi(\overline{a}) = 1 \text{ or } \psi(\overline{a}) = 1, \\ 0 & \text{if } \phi(\overline{a}) = 0 \text{ and } \psi(\overline{a}) = 0, \\ \star & \text{otherwise.} \end{cases}$$

Note that $\phi(\overline{a})$ equals 1 (resp., 0) for some partial assignment $\overline{a}$; then $\phi(\overline{a}')$ also equals 1 (resp., 0) for every Boolean $\overline{a}'$ obtained by replacing every $\star$ in $\overline{a}$ with either a 0 or 1. The converse, however, is not true: The formula $\phi = v \vee \neg v$ evaluates to 1 on every Boolean assignment, yet is not 1 when evaluated at $\star$. Thus, the "law of excluded middle" $\phi \vee \neg\phi \equiv 1$ no longer holds in this setting. However, other identities in Boolean algebra, such as De Morgan's laws (e.g., $\neg(\phi \vee \psi) \equiv \neg\phi \wedge \neg\psi$), do remain true.

DEFINITION 7.8. *For a promise problem* Π, *the* characteristic function *of* Π *is*

[15] In fact, since **SZK** is closed under complement [48], its closure properties extend even to non-monotone formulae.

the map $\chi_\Pi : \{0,1\}^* \to \{0,1,\star\}$ *given by*

$$\chi_\Pi(x) = \begin{cases} 1 & \text{if } x \in \Pi_Y, \\ 0 & \text{if } x \in \Pi_N, \\ \star & \text{otherwise.} \end{cases}$$

DEFINITION 7.9. *For any promise problem* Π *and constant* $\delta > 0$*, we define a new promise problem* $\mathrm{Mon}_\delta(\Pi)$ *as follows:*

$$\mathrm{Mon}_\delta(\Pi)_Y = \{(\phi, x_1, \ldots, x_k) : \phi(\chi_\Pi(x_1), \ldots, \chi_\Pi(x_k)) = 1 \text{ and } \forall i\ |x_i| \geq n^\delta\},$$
$$\mathrm{Mon}_\delta(\Pi)_N = \{(\phi, x_1, \ldots, x_k) : \phi(\chi_\Pi(x_1), \ldots, \chi_\Pi(x_k)) = 0 \text{ and } \forall i\ |x_i| \geq n^\delta\},$$

where ϕ *is a* monotone *k-ary propositional formula, and* $n = |(\phi, x_1, \ldots, x_k)|$.

The condition $|x_i| \geq n^\delta$ is a technicality due to the fact that the security of zero-knowledge proofs is defined with respect to the input length. Intuitively, we will be constructing zero-knowledge proofs for instances of $\mathrm{Mon}_\delta(\Pi)$ of length $n = |(\phi, x_1, \ldots, x_k)|$, but these will be built by using zero-knowledge proofs (or the resulting SZK/OWF CONDITION) for the individual x_i's. Hence to achieve security in terms of n, we will need the x_i's to be of length polynomially related to n. Naturally, this entire issue disappears if one works with a security-parameterized definition of zero knowledge (cf. remark 5 at the end of section 2.5).

THEOREM 7.10. *For any promise problem* $\Pi \in \mathbf{SZK}$ *and any constant* $\delta > 0$*,* $\mathrm{Mon}_\delta(\Pi) \in \mathbf{SZK}$.

Proof. First, we note that **IP** is closed under $\mathrm{Mon}_\delta(\cdot)$. To prove that $(\phi, x_1, \ldots, x_k)$ is in $\mathrm{Mon}_\delta(\Pi)_Y$, it suffices to prove that a subset of the x_i's is in Π_Y, due to the monotonicity of ϕ. Thus, by Theorem 1.2 we need only show that if Π satisfies the SZK/OWF CONDITION, then $\mathrm{Mon}_\delta(\Pi)$ satisfies the SZK/OWF CONDITION.

Let Π be any promise problem satisfying the SZK/OWF CONDITION, with a corresponding set $I \subseteq \Pi_Y$ and poly-time auxiliary-input functions $\{f_x\}$ such that $\Pi' = (\Pi_Y \setminus I, \Pi_N)$ is in **SZK** and f_x is hard to invert when $x \in I$. Since **SZK** is closed under $\mathrm{Mon}_\delta(\cdot)$ (even for $\delta = 0$) [14, 52], we have that $\mathrm{Mon}_\delta(\Pi') \in \mathbf{SZK}$. Note that $\mathrm{Mon}_\delta(\Pi')$ is identical to $\mathrm{Mon}_\delta(\Pi)$ except on instances $(\phi, x_1, \ldots, x_k)$, where at least one x_i is in I, because then $\chi_\Pi(x_i) = 1$ but $\chi_{\Pi'}(x_i) = \star$. Specifically, since changing a variable's assignment from 1 to $\star$ can change the value of a monotone formula only from 1 to $\star$, we have $\mathrm{Mon}_\delta(\Pi')_N = \mathrm{Mon}_\delta(\Pi)_N$ and $\mathrm{Mon}_\delta(\Pi')_Y = \mathrm{Mon}_\delta(\Pi)_Y \setminus J$, where

$$J = \{(\phi, x_1, \ldots, x_k) \in \mathrm{Mon}_\delta(\Pi)_Y : \exists i\ x_i \in I\}.$$

Thus, to show that $\mathrm{Mon}_\delta(\Pi)$ satisfies the SZK/OWF CONDITION, it suffices to show that we can construct a one-way function from any instance in J. To do this, we simply define

$$g_{(\phi, x_1, \ldots, x_k)}(y_1, \ldots, y_k) = (f_{x_1}(y_1), \ldots, f_{x_k}(y_k)).$$

Then when $x = (\phi, x_1, \ldots, x_k) \in J$, there is at least one f_{x_i} that is hard to invert (by nonuniform PPT algorithms running in time $\mathrm{poly}(|x_i|) = \mathrm{poly}(|x|)$, since $|x| \geq |x_i| \geq |x|^\delta$), implying that g is hard to invert. ∎

Following [52], Theorem 7.10 implies that **ZK** is closed under "$\mathbf{NC}^1$ truth-table reductions" (nonadaptive Cook reductions, where the postcomputation is done by a polynomial-sized formula) and implies that the hierarchy of "computational knowledge complexity in the hint sense" [31] collapses by logarithmic additive terms. Details can be found in our technical report [59].

7.3. Expected polynomial-time simulators and weak-ZK. Recall that, following Goldreich [23], our definitions of zero knowledge (in section 2.5) refer to simulators that run in strict polynomial time. In this section, we extend our results to the original Goldwasser–Micali–Rackoff [35] definition, which allows the simulator to run in expected polynomial time. Indeed, we will prove that the two definitions yield exactly the same class **ZK**; that is, every problem having a zero-knowledge proof with an expected polynomial-time simulator also has one with a strict polynomial-time simulator. In fact, we will consider a further relaxation, captured by the following definitions.

DEFINITION 7.11. *For a function $\varepsilon : \mathbb{N} \to [0,1]$, we say that two auxiliary-input probability ensembles $\{X_x\}$ and $\{Y_x\}$ are* ε-indistinguishable *on $I \subseteq \{0,1\}^*$ if for every nonuniform PPT D, there exists a negligible function μ such that for all $x \in I$,*

$$|\Pr[D(x, X_x) = 1] - \Pr[D(x, Y_x) = 1]| \leq \varepsilon(|x|) + \mu(|x|).$$

DEFINITION 7.12 (weak zero knowledge [15]). *An interactive proof system (P, V) for a promise problem Π is* weak honest-verifier zero knowledge *if for every polynomial p there exists a probabilistic (strict) polynomial-time* simulator *S such that the ensembles $\{\langle P, V\rangle(x)\}_{x \in \Pi_Y}$ and $\{S(x)\}_{x \in \Pi_Y}$ are $(1/p(n))$ indistinguishable.*

weak-HVZK *denotes the class of promise problems having weak honest-verifier zero-knowledge proofs.*

The above definition is more relaxed than allowing expected polynomial-time simulators, because if a simulator S has expected running time $t(n)$, then running it for $p(n) \cdot t(n)$ steps yields a strict polynomial-time simulator whose output distribution is $(1/p(n))$-close to that of S. In particular, if the verifier's view is computationally indistinguishable from the output of S, then it is $(1/p(n))$-indistinguishable from the truncated version of S. (An intermediate notion is that of ε-knowledge [16], where the simulator's running time is required to be bounded by a fixed polynomial in $p(n)$ and $t(n)$.)

We remark that in the past, expected polynomial-time simulators and weak simulators have arisen mainly when considering cheating verifiers (e.g., in [35, 29, 25, 15, 16]); that is, *strict* polynomial-time simulators have always seemed to suffice for simulating the *honest* verifier's view. For such cases, an equivalence between zero knowledge with weak simulators (for cheating verifiers) and zero knowledge with strict polynomial-time simulators has already been established by our result that **HVZK** = **ZK** (Theorem 6.1). However, this does leave open the possibility that weak simulation makes a difference for honest-verifier zero knowledge. We rule out this possibility in the following theorem.

THEOREM 7.13. **weak-HVZK** = **ZK**.

Analogous results were previously known for statistical zero knowledge [34] and noninteractive statistical zero knowledge [33].

By the definitions, **ZK** $\subseteq$ **weak-HVZK**, so we need only show **weak-HVZK** $\subseteq$ **ZK**. We will do this by showing that every problem in **weak-HVZK** satisfies the SZK/OWF CONDITION, and by applying Theorem 6.1. (By definition, **weak-HVZK** $\subseteq$ **IP**.) We will do the former by extending our proof that every problem in **HVZK** satisfies the SZK/OWF CONDITION (from section 3). Intuitively, the "weak" computational indistinguishability in the definition of **weak-HVZK** will translate into obtaining a "weak" one-way function (in the sense that the inversion probability is bounded by, say, 1/2 rather than being negligible), and then we will apply Yao's conversion from weak one-way functions to standard one-way functions (see Goldreich [23, Thm. 2.3.2]).

We begin with an extension of Lemma 3.7.

LEMMA 7.14. *If a promise problem* Π *is in* **weak-HVZK**, *then* Π *satisfies the following* WEAK CONDITIONAL PSEUDOENTROPY CONDITION*: There exists a* fixed *polynomial* m *such that for* every *polynomial* p*, there is a polynomial-time computable function mapping strings* x *to a* samplable joint distribution (X, Y) *on* $\{0,1\}^{m(|x|)} \times \{0,1\}^{m(|x|)}$ *and a parameter* r *such that*

- *if* $x \in \Pi_Y$*, then there exists a (not necessarily samplable) joint distribution* (X', Y') *such that* (X', Y') *is* $(1/p(n))$*-indistinguishable from* (X, Y) *and* $\mathrm{H}(X'|Y') \geq r$*, and*
- *if* $x \in \Pi_N$*, then* $\mathrm{H}(X|Y) \leq r - 1$.

A crucial point is that the output length m of the circuits X and Y does not grow with the level of indistinguishability required (as specified by p).[16] Note, however, that we allow the sizes of the circuits and their input length (i.e., number of coin tosses) to indeed depend on p.

Proof sketch. Recall that the proof of Lemma 3.7 first constructed distributions X and Y as follows:

(X, Y): Select $i \leftarrow \{1, \ldots, \ell(|x|)\}$, choose random coin tosses R for the simulator, and output $(S_{2i}(x; R), S_{2i-1}(x; R))$,

where $\ell = \ell(|x|)$ is the number of rounds in the proof system. Here we do the same, but for any given polynomial p, we take S to be the simulator achieving ε-indistinguishability, where $\varepsilon(|x|) = 1/(\ell(|x|) \cdot p(|x|))$.

As in the proof of Lemma 3.7, when $x \in \Pi_Y$, then (X, Y) is ε-indistinguishable from $(X', Y') = (\langle P, V\rangle_{2I}, \langle P, V\rangle_{2I-1})$, where I denotes a uniform random element of $\{1, \ldots, \ell\}$, and $\mathrm{H}(X'|Y') = r/\ell$. On the other hand, when $x \in \Pi_N$, then $\mathrm{H}(X|Y) \leq (r-1)/\ell$, exactly as in Lemma 3.7.

The final distributions are taken to be $(X_1, \ldots, X_\ell)$ and $(Y_1, \ldots, Y_\ell)$, where each (X_i, Y_i) is an independent copy of (X, Y). This increases the entropy gap to 1 bit as before, and the level of indistinguishability deteriorates to $\ell \cdot \varepsilon < 1/p$. Notice that the output lengths of these distributions depend only on the communication complexity of the proof system (but the circuit sizes and number of random bits required depend on the simulator, which in turn depends on the choice of p). □

Given this lemma, we proceed to reduce the WEAK CONDITIONAL PSEUDOENTROPY CONDITION to the SZK/OWF CONDITION, analogously to Lemma 3.10. In the proof, we will need a weak analogue of the notion of a false entropy generator, as follows.

DEFINITION 7.15. *We say that there is an* auxiliary-input weak false entropy generator on I *if there exists a* fixed *polynomial* m *such that for* every *polynomial* p*, we have samplable auxiliary-input probability ensembles* $\mathcal{D} = \{D_x\}$ *and* $\mathcal{F} = \{F_x\}$ *such that* D_x *and* F_x *take values in* $\{0,1\}^{m(|x|)}$ *and when* $x \in I$*,* D_x*, and* F_x *are* $1/p(|x|)$*-indistinguishable and satisfy* $\mathrm{H}(F_x) \geq \mathrm{H}(D_x) + 1$.

The following generalization of Lemma 3.12, proved in Appendix B, states that such weak false entropy generators also imply one-way functions.

[16] Indeed, otherwise every promise problem would trivially satisfy the WEAK CONDITIONAL PSEUDOENTROPY CONDITION. Let $p = p(n)$ be an arbitrary polynomial, and let $m = p$. Given an input x of length n, let (X, Y) be the distribution that always outputs $(0^m, 0^m)$, let $r = 1$, and let (X', Y') equal $(0^m, 0^m)$ with probability $1 - 1/p$ and equal $(U_m, 0^m)$ with probability $1/p$. Then $\mathrm{H}(X|Y) = 0 \leq r - 1$, (X', Y') is $1/p$-close to (X, Y), and $\mathrm{H}(X'|Y') \geq (1/p) \cdot m = r$.

LEMMA 7.16. *If there is an auxiliary-input weak false entropy generator on I, then there exists an auxiliary-input one-way function on I.*

We now use this to establish the SZK/OWF CONDITION.

LEMMA 7.17. *If a promise problem satisfies the* WEAK CONDITIONAL PSEUDOENTROPY CONDITION*, then it satisfies the* SZK/OWF CONDITION.

Proof. Let Π be a promise problem satisfying the WEAK CONDITIONAL PSEUDOENTROPY CONDITION, with m being the associated fixed polynomial. Then for any given $\varepsilon = \varepsilon(n) = 1/\text{poly}(n)$ and any instance $x \in \{0,1\}^n$, we can efficiently construct two samplable distributions (X, Y) on $\{0,1\}^m \times \{0,1\}^m$ and a parameter r such that if $x \in \Pi_Y$, then $\mathrm{H}(X'|Y') \geq r + 1$ for some (X', Y') that is ε-indistinguishable from (X, Y), and if $x \in \Pi_N$, then $\mathrm{H}(X|Y) \leq r - 1$.

Let I be the set of instances $x \in \Pi_Y$ such that $\mathrm{H}(X|Y) < r$. The argument that $\Pi' = (\Pi_Y \setminus I, \Pi_N)$ is in **SZK** is identical to the argument in the proof of Lemma 3.10.

Thus, we focus on constructing one-way functions on I. The first step of the construction (given in the proof of Lemma 3.10) does not change. We set $k = 4n \cdot (m+n)^2$ and consider the samplable distributions

$$D = (H, Y_1, \ldots, Y_k, H(X_1, \ldots, X_k)),$$
$$F = (H, Y_1, \ldots, Y_k, U_{kr+1}).$$

As in the proof of Lemma 3.10, $\mathrm{H}(F) \geq \mathrm{H}(D) + 1$. The only change is that instead of arguing that D and F are computationally indistinguishable, we claim that they are ε'-indistinguishable from Z for $\varepsilon' = 2k \cdot \varepsilon$. The deterioration by a factor of k comes from applying the hybrid argument to k samples of (X_i, Y_i); this occurs both when relating D to D^* and when relating F to F^* in the proof of Lemma 3.10; hence the additional factor of 2. Recalling that $k = 4n \cdot (m+n)^2$ depends only on n and the output length m, we see that we can still make the level ε' of indistinguishability arbitrarily small (by a suitable choice of ε). Moreover, the output length m' of D and F remains independent of the choice of $\varepsilon' = 1/\text{poly}(n)$. Thus, we have a weak auxiliary-input false entropy generator on I. By Lemma 7.16, we have an auxiliary-input one-way function on I, as needed. ∎

8. Open problems. The following are some results that are known about **ZK** under the assumption that one-way functions exist, but for which we have not given unconditional proofs:

1. **ZK** is closed under complement. (If one-way functions exist, then **ZK** = **IP** = **PSPACE** = **co-PSPACE** [29, 39, 7, 45, 37, 42, 54].)
2. If $\Pi \in \mathbf{ZK} \cap \mathbf{NP}$, then Π has a constant-round zero-knowledge proof with soundness error $1/\text{poly}(n)$ [29, 9]. (Constant-round protocols with negligible soundness error are known under stronger assumptions [25].)
3. If $\Pi \in \mathbf{ZK} \cap \mathbf{NP}$, then Π has a computational zero-knowledge proof, where the prover runs in PPT given an **NP** witness for membership [29]. (In our Theorem 6.1, the prover needs an **NP** *oracle*.)

The only bottleneck for proving the latter two results unconditionally is our instance-dependent commitment scheme for **SZK** (Theorem 4.3), which has polynomially many rounds and a $\mathbf{BPP}^{\mathbf{NP}}$ sender, so any improvement to that commitment scheme in these respects would have an analogous impact on **ZK**. In fact, at the time of this work, the last two items (round complexity and prover efficiency) were open problems for **SZK** as well, and in [44] instance-dependent commitments were proposed as an approach to the question of prover efficiency for **SZK**. Subsequent to this work, in joint work with Nguyen [46], we resolve the prover efficiency question, proving

item 3 unconditionally, as well as its **SZK** analogue. That work does not, however, construct standard instance-dependent commitment schemes with an efficient sender for all of **SZK** and **ZK** (but rather some new variant of such commitment schemes), and this remains an interesting open problem having additional consequences, e.g., for unconditional results on concurrent zero knowledge [43].

Given that we have been able to prove unconditional results about **ZK**, which allows for computational security in the zero-knowledge condition, a natural subsequent project is to try and handle computational security in the *soundness* condition, that is, undertake a similar unconditional study of zero-knowledge *arguments*, as defined in [10, 23].

Appendix A. Lemmas about flat distributions.

LEMMA A.1 (flattening lemma restated). *Let X be a distribution, k be a positive integer, and $\otimes^k X$ denote the distribution composed of k independent copies of X. Suppose that for all x in the support of X it holds that $\Pr[X = x] \geq 2^{-m}$. Then $\otimes^k X$ is $\sqrt{k} \cdot m$-flat.*

Suppose Y is jointly distributed with X, and for all (x, y) in the support of (X, Y) it holds that $\Pr[X = x|Y = y] \geq 2^{-m}$. Then, defining $((X_1, Y_1), \ldots, (X_k, Y_k)) = \otimes^k (X, Y)$, the random variable $(X_1, \ldots, X_k)$ is $\sqrt{k} \cdot m$-flat given $(Y_1, \ldots, Y_k)$.

Proof. For every (x, y) in the support of (X, Y), we define the *weight of x given y* to be $\mathrm{wt}(x|y) = \log(1/\Pr[X = x|Y = y])$. Then $\mathrm{wt}(\cdot)$ maps the support of (X, Y) to $[0, m]$. For every $x_1, \ldots, x_k$ and $y_1, \ldots, y_k$, we have

$$\log \frac{1}{\Pr[(X_1, \ldots, X_k) = (x_1, \ldots, x_k)|(Y_1, \ldots, Y_k) = (y_1, \ldots, y_k)]} = \sum_{i=1}^{k} \mathrm{wt}(x_i|y_i).$$

Thus, if we let $\overline{X} = (X_1, \ldots, X_k)$ and $\overline{Y} = (Y_1, \ldots, Y_k)$, we have

$$\Pr\left[\overline{X} \text{ is not } t\Phi\text{-typical given } \overline{Y}\right] = \Pr\left[\left|\sum_{i=1}^{k} \mathrm{wt}(X_i|Y_i) - \mathrm{H}(\overline{X}|\overline{Y})\right| \geq t\Phi\right].$$

For every i, $\mathrm{E}[\mathrm{wt}(X_i|Y_i)] = \mathrm{H}(X|Y)$ and $\mathrm{H}(\overline{X}|\overline{Y}) = k \cdot \mathrm{H}(X|Y)$, so we are bounding the probability that the average of k independent, identically distributed random variables taking values in $[0, m]$ deviates from its expectation by $t\Phi/k$. By the Hoeffding inequality, this probability is at most

$$2 \cdot \exp\left(\frac{-2 \cdot k \cdot (t\Phi/k)^2}{m^2}\right).$$

For $\Phi = \sqrt{k} \cdot m$ and $t \geq 1$, this bound becomes $2\exp(-2t^2) \leq 2^{-t^2}$, establishing the lemma. □

LEMMA A.2 (Claim 5.9 restated). *Let Z_0 and Z_1 be Φ-flat distributions, for $\Phi \geq 1$. Let $Z = Z_C$, where C is a uniformly chosen random bit. Then Z is 3Φ-flat.*

Proof. We need to show that, for every $t \geq 1$, a random sample $z \leftarrow Z$ is not $t \cdot 3\Phi$-typical for Z with probability at most 2^{-t^2}. For this, it suffices to separately bound the probabilities that z is not $t{\cdot}3\Phi$-light and that z is not $t{\cdot}3\Phi$-heavy. Note that $t{\cdot}3\Phi \geq 2t{\cdot}\Phi{+}1$, so we can bound the probabilities with respect to a lightness/heaviness threshold of $2t \cdot \Phi + 1$ instead.

Bounding the lightness probability is relatively straightforward because z being light for Z implies that it is light for both Z_0 and Z_1. Specifically, for any z that is

$(2t \cdot \Phi + 1)$-light for Z, we have

$$\Pr[Z_0 = z] \leq 2 \cdot \Pr[Z = z] \leq 2 \cdot 2^{-(2t\Phi+1)} \cdot 2^{-\mathrm{H}(Z)} \leq 2^{-2t\Phi} \cdot 2^{-\mathrm{H}(Z_0)}.$$

The analogous bound holds for Z_1. Therefore any such z is also $2t\Phi$-light for Z_0 and Z_1. Hence, if $z \leftarrow Z$, then z is $(2t \cdot \Phi + 1)$-light for Z with probability at most $2^{-(2t)^2}$.

The heaviness probability is a bit more subtle because z being heavy for Z implies only that it is heavy for either Z_0 or Z_1; specifically, if z is $(2t \cdot \Phi + 1)$-heavy for Z, then

$$\max\{\Pr[Z_0 = z], \Pr[Z_1 = z]\} \geq \Pr[Z = z] \geq 2^{2t\Phi+1} \cdot 2^{-\mathrm{H}(Z)} \geq 2^{2t\Phi+1} \cdot 2^{-(\mathrm{H}(Z_0)+1)}.$$

Thus, any such z is $2t\Phi$-heavy for either Z_0 or Z_1. W.l.o.g. say that z is heavy for Z_0. The probability that Z_0 outputs a string that is $2t\Phi$-heavy (for Z_0) is at most $2^{-(2t)^2}$, by Φ-flatness. However we also need to bound the probability that Z_1 outputs such a string. Let H_0 be the set of strings that are $2t\Phi$-heavy for Z_0. The total probability mass of H_0 under Z_0 is at least $|H_0| \cdot 2^{-\mathrm{H}(Z_0)+2t\Phi}$ and at most $2^{-(2t)^2}$ by Φ-flatness. Thus, $|H_0| \leq 2^{-(2t)^2} \cdot 2^{\mathrm{H}(Z_0)-2t\Phi}$. Then

$$\Pr[Z_1 \in H_0] \leq \Pr[Z_1 \text{ is } 2t\Phi\text{-heavy}] + |H_0| \cdot 2^{-\mathrm{H}(Z_1)+2t\Phi} \leq 2^{-(2t)^2} + 2^{-(2t)^2}.$$

We can perform an identical analysis for the strings H_1 that are $2t\Phi$-heavy for Z_1. Then

$$\begin{aligned}\Pr[Z \in H_0 \cup H_1] &= \frac{1}{2}\left(\Pr[Z_0 \in H_0] + \Pr[Z_1 \in H_0] + \Pr[Z_0 \in H_1] + \Pr[Z_1 \in H_1]\right) \\ &\leq \frac{1}{2}\left(2^{-(2t)^2} + 2 \cdot 2^{-(2t)^2} + 2 \cdot 2^{-(2t)^2} + 2^{-(2t)^2}\right) \\ &= 3 \cdot 2^{-(2t)^2}.\end{aligned}$$

In total, we see that the probability that a random sample of Z is not $(2t\Phi+1)$-typical for Z is at most $2^{-(2t)^2} + 3 \cdot 2^{-(2t)^2} \leq 2^{-t^2}$, for $t \geq 1$. ∎

Appendix B. Weak false entropy generators imply one-way functions. We recall the definition of a weak false entropy generator.

DEFINITION B.1 (Definition 7.15 restated). *We say that there is an* auxiliary-input weak false entropy generator on I *if there exists a* fixed *polynomial m such that for* every *polynomial p, we have samplable auxiliary-input probability ensembles $\mathcal{D} = \{D_x\}$ and $\mathcal{F} = \{F_x\}$ such that D_x and F_x take values in $\{0,1\}^{m(|x|)}$ and when $x \in I$, D_x, and F_x are $1/p(|x|)$-indistinguishable and satisfy $\mathrm{H}(F_x) \geq \mathrm{H}(D_x) + 1$.*

The following generalizes Lemma 3.12 (due to [37]). Intuitively, the weakness of the false entropy generator translates to constructing only a weak one-way function (where the inversion probability is at most, say, 1/2), which is known to imply standard one-way functions [60] (cf. [23]).

LEMMA B.2 (Lemma 7.16 restated). *If there is an auxiliary-input weak false entropy generator on I, then there exists an auxiliary-input one-way function on I.*

Proof. Let $x \in I$, $n = |x|$, and let $D = D_x$ and $F = F_x$ be the samplable auxiliary-input probability ensembles on $\{0,1\}^m$ that are ε-indistinguishable. Recall that the definition of auxiliary-input weak false entropy generators gives us a fixed polynomial $m = m(n)$ such that we can take $\varepsilon = 1/p(n)$ for any desired polynomial p (which we will choose later in the proof). To construct an auxiliary-input one-way

function, we will essentially follow the construction of Håstad et al. [37] which converts a "false entropy generator" to a "pseudoentropy generator"—where the output is indistinguishable from a distribution whose min-entropy is higher than the seed-length of the generator. However, since we are starting from only a weak false entropy generator D, we need to ensure that the level of indistinguishability deteriorates only as a function of the output length m of D and the security parameter (but not with the number of random bits used to generate D).

This part of the construction depends on "guess" e for (an approximation to) the entropy of D. (At the end we will enumerate over all choices for e.) Specifically, set $k = 256n \cdot (m+n)^2$, let q be the number of random bits used to generate D, let G be a random universal hash function mapping $\{0,1\}^{kq}$ to $\{0,1\}^{kq-ke-k/8}$, and consider the following samplable distributions:

$$W_e = (D(R_1), \ldots, D(R_k), G, G(R_1, \ldots, R_k)),$$
$$W'_e = (F_1, \ldots, F_k, G, U_{kq-ke-k/8}),$$

where $R_1, \ldots, R_k$ are independent copies of U_q, and $F_1, \ldots, F_k$ are independent copies of F.

Claim B.3. *For* $\mathrm{H}(D) \le e \le \mathrm{H}(D) + 1/2$*, we have that*

1. W_e *and* W'_e *are* $k\varepsilon$*-indistinguishable.*
2. $\Pr[W'_e \in \mathrm{Supp}(W_e)] \le (k+2) \cdot 2^{-n}$.

Before proving the claim, we describe how it completes the proof of the lemma. Specifically, we argue that the circuit generating W_e defines a (weak) one-way function. Any algorithm that inverts W_e with probability at least δ can be used to distinguish between W_e and W'_e with an advantage of at least $\delta - (k+2) \cdot 2^{-n}$ (because by item 2 it is information-theoretically impossible to find a W_e-preimage of a random sample of W'_e, except with probability $(k+2) \cdot 2^{-n}$). By item 1, we conclude that W_e can be inverted with probability at most $\delta = (k+2) \cdot 2^{-n} + k\varepsilon \le 1/2$, for a sufficiently large choice of the polynomial p (recalling that $\varepsilon = 1/p$), and is thus a weak one-way function. Since we do not know the value of $\mathrm{H}(D)$, we consider the function $f_x(y_1, \ldots, y_{2m}) = (W_{1/2}(y_1), W_1(y_2), \ldots, W_{m-1/2}(y_{2m-1}), W_m(y_{2m}))$, which is a weak one-way function because one of its components is a weak one-way function (and the others are independent). Applying the standard reduction from weak one-way functions to standard one-way functions [60] (cf. [23]) completes the proof. Thus, all that remains is to establish Claim B.3.

Proof of Claim B.3. It will first be useful to remove low-probability samples from both D and F, analogously to Lemma 2.2. Let

$$L = \{z : \Pr[D = z] \le 2^{-n} \cdot 2^{-m}\}.$$

By a union bound, $\Pr[D \in L] \le 2^{-n}$. Then $\hat{D} = D|_{D \notin L}$ is 2^{-n}-close to D and, moreover, for every $z \in \mathrm{Supp}(\hat{D})$,

$$\Pr\left[\hat{D} = z\right] \ge \Pr[D = z] \ge 1/2^{m+n}.$$

By Lemma 2.1, we have $|\mathrm{H}(\hat{D}) - \mathrm{H}(D)| \le 2^{-n} \cdot m + \mathrm{H}_2(2^{-n})$, which is negligible. By the flattening lemma, $\otimes^k \hat{D}$ is Φ-flat for $\Phi = \sqrt{k} \cdot (m+n)$. Analogously, using F we can define a set L' of light samples and obtain an $\hat{F}$ satisfying the same conclusions.

The Φ-flatness of $\otimes^k \hat{D}$ implies that with probability at least $1 - 2^{-n}$ over $\overline{z} = (z_1, \ldots, z_k) \leftarrow \otimes^k \hat{D}$, we have

$$\Pr[\otimes^k \hat{D} = \overline{z}] \geq 2^{-\sqrt{n}\cdot\Phi} \cdot 2^{-k\cdot \mathrm{H}(\hat{D})}.$$

Since $\otimes^k D$ and $\otimes^k \hat{D}$ are $k\cdot 2^{-n}$-close (by Lemma 2.3), the same holds with probability at least $1 - (k+1)\cdot 2^{-n}$ over $\overline{z} \leftarrow \otimes^k D$. For any such $\overline{z}$, we have

$$\begin{aligned}
&\#\{(r_1, \ldots, r_k) : \forall i\ D(r_i) = z_i\} \\
&= 2^{kq} \cdot \Pr[\otimes^k D = \overline{z}] \\
&\geq 2^{kq} \cdot \Pr[\otimes^k D = \overline{z} | \otimes^k D \in (L^c)^k] \cdot \Pr\left[\otimes^k D \in (L^c)^k\right] \qquad (\text{where } L^c = \{z : z \notin L\}) \\
&\geq 2^{kq} \cdot \Pr[\otimes^k \hat{D} = \overline{z}] \cdot (1 - k \cdot 2^{-n}) \\
&\geq 2^{kq} \cdot 2^{-\sqrt{n}\cdot\Phi - k\cdot \mathrm{H}(\hat{D})} \cdot (1 - k \cdot 2^{-n}) \\
&\geq 2^{kq - ke - k/8 + 2n},
\end{aligned}$$

where L^c denotes the complement of L and in the last inequality we use the facts that $\mathrm{H}(\hat{D}) \leq \mathrm{H}(D) + \mathrm{neg}(n) \leq e + \mathrm{neg}(n)$ and $\sqrt{n}\cdot\Phi = k/16$, $2n + 1 \leq k/16$ for sufficiently large n. This implies that conditioned on $(D(R_1), \ldots, D(R_k)) = \overline{z}$, the min-entropy of $(R_1, \ldots, R_k)$ is at least $kq - ke - k/8 + 2n$. Thus, by the leftover hash lemma (Lemma 2.7), $(G, G(R_1, \ldots, R_k))$ is (2^{-n})-close to $(G, U_{kq-ke-k/8})$. We conclude that W_e is statistically indistinguishable from

$$V = (D_1, \ldots, D_k, G, U_{kq-ke-k/8}),$$

where $D_1, \ldots, D_k$ are independent copies of D. Since D is ε-indistinguishable from F, it follows that V is $(k\varepsilon)$-indistinguishable from W'_e. Therefore, W_e and W'_e are $(k\varepsilon)$-indistinguishable, as desired.

Now we proceed to item 2. First, we bound $|\mathrm{Supp}(W_e)|$. Let g be the number of random bits to generate G. Then the number of random bits used to generate W_e is at most $kq + g$. Hence $|\mathrm{Supp}(W_e)| \leq 2^{kq+g}$. Next, we show that W'_e is statistically indistinguishable from a distribution with min-entropy significantly higher than $kq+g$. This amounts to lower bounding the min-entropy of $(F_1, \ldots, F_k) = \otimes^k F$, since the remaining components of the W'_e are independent and have min-entropy $g + kq - ke - k/8$. As above, instead of F, we consider $\hat{F}$. Recall that $\otimes^k \hat{F}$ is $(k2^{-n})$-close to $\otimes^k F$ and is Φ-flat. By Φ-flatness, $\otimes^k \hat{F}$ is (2^{-n})-close to a distribution with min-entropy $k \cdot \mathrm{H}(\hat{F}) - \sqrt{n}\Phi \geq k \cdot (e + 1/2 - \mathrm{neg}(n)) - k/16 \geq ke + k/4$ for sufficiently large n. Therefore, W'_e is $(k+1)\cdot(2^{-n})$-close to a distribution with min-entropy at least

$$(g + kq - ke - k/8) + (ke + k/4) > kq + g + n$$

for sufficiently large n. A distribution of min-entropy at least $w = kq + g + n$ can land in $\mathrm{Supp}(W_e)$ with probability at most $2^{-w} \cdot |\mathrm{Supp}(W_e)| \leq 2^{-n}$. Therefore W'_e lands in $\mathrm{Supp}(W_e)$ with probability at most $2^{-n} + (k+1)\cdot 2^{-n}$, as desired. □

Acknowledgments. I am grateful to Emanuele Viola for an inspiring conversation about pseudoentropy and [37] that prompted me to revisit the questions addressed in the present paper. I thank Oded Goldreich, Shafi Goldwasser, and Shien Jin Ong for clarifying discussions. Their comments, as well as those of the anonymous referees, also improved the presentation significantly. Finally, I thank Danny Gutfreund, Madhu Sudan, and Luca Trevisan for some past conversations that have influenced this work.

REFERENCES

[1] W. Aiello and J. Håstad, *Statistical zero-knowledge languages can be recognized in two rounds*, J. Comput. System Sci., 42 (1991), pp. 327–345.

[2] L. Babai and S. Moran, *Arthur-Merlin games: A randomized proof system and a hierarchy of complexity classes*, J. Comput. System Sci., 36 (1988), pp. 254–276.

[3] B. Barak, *How to go beyond the black-box simulation barrier*, in Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (Las Vegas, NV), 2001, pp. 106–115.

[4] B. Barak, Y. Lindell, and S. Vadhan, *Lower bounds for non-black-box zero knowledge*, J. Comput. System Sci., 72 (2006), pp. 321–391.

[5] M. Bellare, O. Goldreich, and E. Petrank, *Uniform generation of NP-witnesses using an NP-oracle*, Inform. and Comput., 163 (2000), pp. 510–526.

[6] M. Bellare, S. Micali, and R. Ostrovsky, *Perfect zero-knowledge in constant rounds*, in Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing (Baltimore, MD), 1990, pp. 482–493.

[7] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway, *Everything provable is provable in zero-knowledge*, in Advances in Cryptology—CRYPTO '88, Lecture Notes in Comput. Sci. 403, S. Goldwasser, ed., Springer-Verlag, Berlin, 1990, pp. 37–56.

[8] C. H. Bennett, G. Brassard, and J.-M. Robert, *Privacy amplification by public discussion*, SIAM J. Comput., 17 (1988), pp. 210–229.

[9] M. Blum, *How to prove a theorem so no one else can claim it*, in Proceedings of the International Congress of Mathematicians (Berkeley, CA), AMS, Providence, RI, 1987, pp. 1444–1451.

[10] G. Brassard, D. Chaum, and C. Crépeau, *Minimum disclosure proofs of knowledge*, J. Comput. System Sci., 37 (1988), pp. 156–189.

[11] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed., Wiley Series in Telecommunications, John Wiley & Sons, New York, 1991.

[12] I. B. Damgård, *On the existence of bit-commitment schemes and zero-knowledge proofs*, in Advances in Cryptology—CRYPTO '89, Lecture Notes in Comput. Sci. 435, G. Brassard, ed., Springer-Verlag, Berlin, 1990, pp. 17–29.

[13] I. B. Damgård, *Interactive hashing can simplify zero-knowledge protocol design without computational assumptions (extended abstract)*, in Advances in Cryptology—CRYPTO '93, Lecture Notes in Comput. Sci. 773, D. R. Stinson, ed., Springer-Verlag, Berlin, pp. 100–109.

[14] A. De Santis, G. Di Crescenzo, G. Persiano, and M. Yung, *Image density is complete for non-interactive-SZK*, in Proceedings of the 25th International Colloquium on Automata, Languages and Programming (Aalborg, Denmark), Lecture Notes in Comput. Sci., Springer-Verlag, Berlin, pp. 784–795.

[15] G. Di Crescenzo, T. Okamoto, and M. Yung, *Keeping the SZK-verifier honest unconditionally*, in Advances in Cryptology—CRYPTO '97, Lecture Notes in Comput. Sci. 1294, B. S. Kaliski, Jr., ed., Springer-Verlag, Berlin, pp. 31–45.

[16] C. Dwork, M. Naor, and A. Sahai, *Concurrent zero-knowledge*, J. ACM, 51 (2004), pp. 851–898.

[17] S. Even, A. L. Selman, and Y. Yacobi, *The complexity of promise problems with applications to public-key cryptography*, Inform. Control, 61 (1984), pp. 159–173.

[18] L. Fortnow, *The complexity of perfect zero-knowledge*, in Advances in Computing Research, Vol. 5, S. Micali, ed., JAI Press, Greenwich, CT, 1989, pp. 327–343.

[19] M. Fürer, O. Goldreich, Y. Mansour, M. Sipser, and S. Zachos, *On completeness and soundness in interactive proof systems*, in Advances in Computing Research, Vol. 5, S. Micali, ed., JAI Press, Greenwich, CT, 1989, pp. 429–442.

[20] O. Goldreich, *A note on computational indistinguishability*, Inform. Process. Lett., 34 (1990), pp. 277–281.

[21] O. Goldreich, *A uniform-complexity treatment of encryption and zero-knowledge*, J. Cryptology, 6 (1993), pp. 21–53.

[22] O. Goldreich, *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, Algorithms and Combin. 17, Springer-Verlag, Berlin, 1999.

[23] O. Goldreich, *Foundations of Cryptography: Basic Tools*, Cambridge University Press, Cambridge, UK, 2001.

[24] O. Goldreich, *On Promise Problems (A Survey in Memory of Shimon Even* [1935–2004]*)*, Tech. Report TR05–018, Electronic Colloquium on Computational Complexity, 2005. Available online at http://eccc.hpi-web.de/.

[25] O. Goldreich and A. Kahan, *How to construct constant-round zero-knowledge proof systems for NP*, J. Cryptology, 9 (1996), pp. 167–190.
[26] O. Goldreich and H. Krawczyk, *Sparse pseudorandom distributions*, Random Structures Algorithms, 3 (1992), pp. 163–174.
[27] O. Goldreich and H. Krawczyk, *On the composition of zero-knowledge proof systems*, SIAM J. Comput., 25 (1996), pp. 169–192.
[28] O. Goldreich and E. Kushilevitz, *A perfect zero-knowledge proof system for a problem equivalent to the discrete logarithm*, J. Cryptology, 6 (1993), pp. 97–116.
[29] O. Goldreich, S. Micali, and A. Wigderson, *Proofs that yield nothing but their validity, or all languages in NP have zero-knowledge proof systems*, J. Assoc. Comput. Mach., 38 (1991), pp. 691–729.
[30] O. Goldreich and Y. Oren, *Definitions and properties of zero-knowledge proof systems*, J. Cryptology, 7 (1994), pp. 1–32.
[31] O. Goldreich and E. Petrank, *Quantifying knowledge complexity*, Comput. Complex., 8 (1999), pp. 50–98.
[32] O. Goldreich, A. Sahai, and S. Vadhan, *Honest verifier statistical zero-knowledge equals general statistical zero-knowledge*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing (Dallas, TX), 1998, pp. 399–408.
[33] O. Goldreich, A. Sahai, and S. Vadhan, *Can statistical zero-knowledge be made non-interactive?, or On the relationship of SZK and NISZK*, in Advances in Cryptology—CRYPTO '99, Lecture Notes in Comput. Sci. 1666, M. Wiener, ed., Springer-Verlag, Berlin, pp. 467–484.
[34] O. Goldreich and S. Vadhan, *Comparing entropies in statistical zero-knowledge with applications to the structure of SZK*, in Proceedings of the Fourteenth Annual IEEE Conference on Computational Complexity (Atlanta, GA), 1999, pp. 54–73.
[35] S. Goldwasser, S. Micali, and C. Rackoff, *The knowledge complexity of interactive proof systems*, SIAM J. Comput., 18 (1989), pp. 186–208.
[36] S. Goldwasser and M. Sipser, *Private coins versus public coins in interactive proof systems*, in Advances in Computing Research, Vol. 5, S. Micali, ed., JAI Press, Greenwich, CT, 1989, pp. 73–90.
[37] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby, *A pseudorandom generator from any one-way function*, SIAM J. Comput., 28 (1999), pp. 1364–1396.
[38] R. Impagliazzo, L. A. Levin, and M. Luby, *Pseudo-random generation from one-way functions (extended abstract)*, in Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing (Seattle, WA), 1989, pp. 12–24.
[39] R. Impagliazzo and M. Yung, *Direct minimum-knowledge computations (extended abstract)*, in Advances in Cryptology—CRYPTO '87 Lecture Notes in Comput. Sci. 293, C. Pomerance, ed., Springer-Verlag, Berlin, 1988, pp. 40–51.
[40] T. Itoh, Y. Ohta, and H. Shizuya, *A language-dependent cryptographic primitive*, J. Cryptology, 10 (1997), pp. 37–49.
[41] M. R. Jerrum, L. G. Valiant, and V. V. Vazirani, *Random generation of combinatorial structures from a uniform distribution*, Theoret. Comput. Sci., 43 (1986), pp. 169–188.
[42] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, *Algebraic methods for interactive proof systems*, J. Assoc. Comput. Mach., 39 (1992), pp. 859–868.
[43] D. Micciancio, S. J. Ong, A. Sahai, and S. Vadhan, *Concurrent zero knowledge without complexity assumptions*, in Proceedings of the Third Theory of Cryptography Conference (TCC '06), Lecture Notes in Comput. Sci. 3876, S. Halevi and T. Rabin, eds., Springer-Verlag, Berlin, pp. 1–20.
[44] D. Micciancio and S. Vadhan, *Statistical zero-knowledge proofs with efficient provers: Lattice problems and more*, in Advances in Cryptology—CRYPTO '03, Lecture Notes in Comput. Sci. 2729, D. Boneh, ed., Springer-Verlag, Berlin, pp. 282–298.
[45] M. Naor, *Bit commitment using pseudorandomness*, J. Cryptology, 4 (1991), pp. 151–158.
[46] M. Nguyen and S. Vadhan, *Zero knowledge with efficient provers*, in Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC '06), Seattle, WA, 2006, pp. 287–295.
[47] N. Nisan and A. Ta-Shma, *Extracting randomness: A survey and new constructions*, J. Comput. System Sci., 58 (1999), pp. 148–173.
[48] T. Okamoto, *On relationships between statistical zero-knowledge proofs*, J. Comput. System Sci., 60 (2000), pp. 47–108.
[49] R. Ostrovsky, *One-way functions, hard on average problems, and statistical zero-knowledge proofs*, in Proceedings of the Sixth Annual Structure in Complexity Theory Conference (Chicago, IL), 1991, pp. 133–138.

[50] R. Ostrovsky and A. Wigderson, *One-way functions are essential for non-trivial zero-knowledge*, in Proceedings of the Second Israel Symposium on Theory of Computing and Systems, IEEE Press, Los Alamitos, CA, 1993, pp. 3–17.

[51] E. Petrank and G. Tardos, *On the knowledge complexity of $\mathcal{NP}$*, in Proceedings of the 37th Annual Symposium on Foundations of Computer Science (Burlington, VT), 1996, pp. 494–503.

[52] A. Sahai and S. Vadhan, *A complete problem for statistical zero knowledge*, J. ACM, 50 (2003), pp. 196–249.

[53] R. Shaltiel, *Recent developments in explicit constructions of extractors*, in Current Trends in Theoretical Computer Science: The Challenge of the New Century, Vol. I: Algorithms and Complexity, G. Paun, G. Rozenberg, and A. Salomaa, eds., World Scientific, River Edge, NJ, 2004, pp. 189–228.

[54] A. Shamir, *IP = PSPACE*, J. Assoc. Comput. Mach., 39 (1992), pp. 869–877.

[55] M. Sipser, *A complexity theoretic approach to randomness*, in Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing (Boston, MA), 1983, pp. 330–335.

[56] L. Stockmeyer, *On approximation algorithms for* #P, SIAM J. Comput., 14 (1985), pp. 849–861.

[57] S. P. Vadhan, *A Study of Statistical Zero-Knowledge Proofs*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, 1999. Available from author's Web page, http://eccs.harvard.edu/∼salil.

[58] S. P. Vadhan, *An unconditional study of computational zero knowledge*, in Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (Rome, Italy), 2004, pp. 176–185.

[59] S. P. Vadhan, *An Unconditional Study of Computational Zero Knowledge*, Tech. Report TR06-056, Electronic Colloquium on Computational Complexity, 2006. Available online from http://eccc.hpi-web.de/.

[60] A. C. Yao, *Theory and applications of trapdoor functions (extended abstract)*, in Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science (Chicago, IL), 1982, pp. 80–91.

SIAM J. COMPUT.
Vol. 36, No. 4, pp. 1215–1230

DERANDOMIZING HOMOMORPHISM TESTING IN GENERAL GROUPS*

AMIR SHPILKA† AND AVI WIGDERSON‡

Abstract. The main result of this paper is a near-optimal derandomization of the *affine* homomorphism test of Blum, Luby, and Rubinfeld [*J. Comput. System Sci.,* 47 (1993), pp. 549–595].

We show that for any groups G and Γ, and any *expanding* generating set S of G, the natural deramdomized version of the BLR test in which we pick an element x randomly from G and y randomly from S and test whether $f(x)\cdot f(y) = f(x\cdot y)$, performs nearly as well (depending of course on the expansion) as the original test. Moreover, we show that the underlying homomorphism can be found by the natural local "belief propagation decoding."

We note that the original BLR test uses $2\log_2|G|$ random bits, whereas the derandomized test uses only $(1+o(1))\log_2|G|$ random bits. This factor of 2 savings in the randomness complexity translates to a near quadratic savings in the length of the tables in the related locally testable codes (and possibly probabilistically checkable proofs which may use them).

Our result is a significant generalization of recent results that either refer to the special case of the groups $G = Z_p^m$ and $\Gamma = Z_p$ or are nonconstructive. We use simple combinatorial arguments and the transitivity of Cayley graphs (and this analysis gives optimal results up to constant factors). Previous techniques used the Fourier transform, a method which seems unextendable to general groups (and furthermore gives suboptimal bounds).

Finally, we provide a polynomial time (in $|G|$) construction of a (somewhat) small ($|G|^\epsilon$) set of expanding generators for *every* group G, which yield efficient testers of randomness $(1+\epsilon)\log|G|$ for G. This result follows from a simple derandomization of a known probabilistic construction.

Key words. derandomization, linearity testing, homomorphism testing

AMS subject classification. 68Q99

DOI. 10.1137/S009753970444658X

1. Introduction.

1.1. Property testers and randomness complexity. Let F be the family of all functions (from a given domain to a given range), and P a subset of these functions (those with property "P"). A tester T is a probabilistic algorithm that receives as input a (black box for) function $f \in F$, evaluates f on a set of points in the domain, and uses this information to accept or reject the input function f. Roughly speaking, T is a tester for the property P if every f in P is accepted with high probability, and every f which is "far" from P (in Hamming distance) is rejected with high probability. This is the basic set up of property testing, by now a very large field dealing with many other objects than functions, such as strings, distributions, graphs, etc. (see excellent surveys by Goldreich [19] and by Ron [30]). A central theme in this field is relating $error(T)$, the probability that our tester fails to give the correct output, to its "complexity" $query(T)$, measuring the number of domain samples it used, and its "accuracy" $\mathrm{dist}(T)$, which is how far from P are the functions it rejects. The importance of this field for various applications follow numerous results giving testers

*Received by the editors November 29, 2004; accepted for publication (in revised form) July 1, 2005; published electronically December 15, 2006.

http://www.siam.org/journals/sicomp/36-4/44658.html

†Technion, Haifa, Israel (shpilka@cs.technion.ac.il). The work of this author was supported by the Koshland fellowship.

‡Institute for Advanced Study, Princeton, NJ (avi@ias.edu). The work of this author was partially supported by NSF grant CCR-0324906.

for a variety of properties P, in which both *query* and *distance* depend only on *error* (and not on the size of the domain of the functions).

Central applications of this area are (the related) locally testable codes (LTCs) and probabilistically checkable proofs (PCPs). In these, the answers to all possible sets of queries are explicitly written down, and it is a major concern to minimize their length. This length can be seen to be directly related to (indeed, an exponential of) the number of random bits $random(T)$ used by the tester T, and so this parameter and its tradeoffs with the others have been investigated as well. Related are the "derandomized" amplification of hardness results [23, 33] which lead to optimal derandomization of BPP.

A recent paper of Goldreich and Sudan [20] addresses the minimization of $random(T)$ for two important testers: the homomorphism tester of Blum, Luby, and Rubinfeld [12] (which was the first and motivating example of property testing of functions), and the "point vs. lines" low-degree tester of Rubinfeld and Sudan [31] (which was central in the proof of the PCP theorem). Both testers use randomness to name *two* random domain queries, which is related to having *quadratic* proof/code length (as a function of the length of the appropriate input). They note that in order to reduce this length to near *linear*, one must use only randomness which is sufficient for only *one* query. Moreover, [20] showed that *nonuniformly* such a saving is possible (the arguments of [20] can achieve similar savings in much more general contexts of multiprover systems, but we will restrict our discussion from this point on only to the first tester for homomorphism, which is the subject of our paper). Indeed Ben-Sasson et al. [13] were able to minimize $random(T)$ for the special case of testing homomorphism between the groups Z_p^m and Z_p.

1.2. Affine homomorphism testing. Given two finite groups G, Γ, a homomorphism is a function $f : G \to \Gamma$ such that for every $g_1, g_2 \in G$ we have that $f(g_1 \cdot g_2) = f(g_1) \cdot f(g_2)$. When the groups are abelian it is customary to use "$+$" instead of "$\cdot$," so a homomorphism is a function that for every $g_1, g_2 \in G$ satisfies $f(g_1 + g_2) = f(g_1) + f(g_2)$. This is the reason that in abelian groups homomorphisms are referred to as linear functions. In particular the famous paper of Blum et al. [12] analyze homomorphism testing (which they call linearity testing) for abelian groups. An *affine* homomorphism between G and Γ is a function f such that $f(1)^{-1} \cdot f$ is a homomorphism (in the case of abelian groups this is sometimes denoted as $f - f(0)$). The BLR linearity testing can be slightly changed to yield an affine version of their linearity test.

Let G and Γ be finite groups. Let F be all functions from G to Γ, let $P_{hom} \subseteq F$ be the set of all homomorphisms from G to Γ, and $P_{aff} \subseteq F$ be the set of all affine homomorphisms from G to Γ. For two functions $f, h \in F$ we have the normalized Hamming distance $\text{dist}(f, h) = Prob[f(x) \neq h(x)]$ for a uniform element $x \in G$.

Fix a subset E of $G \times G$ (which may be viewed as a directed graph on G), and consider the following tester T_E. It picks uniformly a random pair $(x, y) \in E$; evaluates the input function f on the three (related) elements x, y, and $x^{-1}y$; and accepts if and only if it satisfies the equation $f(x)f(x^{-1}y)f(y)^{-1} = 1$. It is easy to see that if f is a homomorphism, then T_E will accept f with probability one. The interesting direction is showing that if the error of the test is small, then f is close to a homomorphism (or an affine homomorphism). We say that T_E is a (δ, ϵ)-test if every function that passes the test with probability at least $1 - \delta$ is at most ϵ far from having the property (either P_{hom} or P_{aff}).

The well known BLR linearity tester [12] uses (in this notation) $E = G \times G$. They

proved that $T_{G\times G}$ is a $(\delta, 9\delta/2)$-test. However, their analysis wasn't tight and was later improved by [9, 8, 7]. Ben-Or et al. [10] extended the BLR result and showed that the test with $E = G \times G$ works for general groups as well. The proof of Ben-Or et al. is similar to the proof of [12].

THEOREM 1.1 (see [12, 10, 8, 9, 7]). *Let G, Γ be groups. Consider the test $T_{G\times G}$ described above. There the test picks uniformly at random two elements $x, y \in G$ and accepts if $f(x) \cdot f(y) = f(x \cdot y)$. For every $\delta > 0$, if f passes the test with probability $> 1-\delta$, then there exist a homomorphism $h \in P_{hom}$ such that $\mathrm{dist}(f, h) \leq \delta/3+O(\delta^2)$. In other words, $T_{G\times G}$ is a $\left(\delta, \delta/3 + O(\delta^2)\right)$-test for P_{hom}.*

To save on randomness, [20] suggested to use sparser graphs E. The tester T_E obviously has $random(T_E) = \log |E|$ (all logs are to base 2). The value attained by the BLR test, $random(T_{G\times G}) = 2\log |G|$. It is also easy to see that any nontrivial tester (giving any dependence between error and distance) must satisfy $random(T_E) \geq \log |G| - O(1)$. Goldreich and Sudan [20] showed that this lower bound can essentially be matched, and at negligible cost to the dependence of distance on error.

THEOREM 1.2 (see [20]). *For all but $exp(-|G|)$ fraction of all possible graphs E of size $C|G| \log |\Gamma|$ (with C an absolute constant) the following holds. For every $\delta > 0$, T_E is a $\left(\delta, \delta/3 + O(\delta^2) + exp(-|G|)\right)$-test for P_{hom}.*

On the one hand, notice that for this size of E, we have $random(T_E) = \log |G| + \log\log |\Gamma| + O(1)$. This gives $(1 + o(1)) \log |G|$ for all interesting cases ($|\Gamma| \leq |G|$). It gives the optimal $\log |G| + O(1)$ when Γ is of fixed size, which includes the important special case of linearity testing in which $\Gamma = Z_p$ for a fixed prime p and $G = Z_p^m$ for a large m.

On the other hand, the proof of [20] is not explicit. It uses a probabilistic argument in choosing E, which gives no clue to which graphs induce good testers. This is a major problem if one wants to use such testers in objects like PCPs. This raises a natural "derandomization" problem (which Goldreich and Sudan [20] raise in their paper), of explicitly constructing good testers E, or at least characterize good testers E.

This problem was answered for a special case of affine linear testing (i.e., for the property P_{aff}), by Ben-Sasson et al. [13] who proved the following theorem.

THEOREM 1.3 (see [13]). *Fix any $\lambda > 0$. Let S be a λ-biased set in $G = Z_p^m$, and let E denote all pairs (x, xs) for all $x \in G$ and $s \in S$. Then for every $\delta > 0$, T_E is a $\left(\delta, O\left(p^2(\delta + \lambda)\right)\right)$-test [1] for P_{aff}.*

λ-biased sets of size $\mathrm{poly}(m/\lambda)$ can be explicitly constructed for these groups [3, 4, 16, 24, 29], which gives explicit testers T_E with near optimal randomness $random(T_E) \leq \log |G| + O(\log(m/\lambda))$.[2]

Ben-Sasson et al. [13] note that the resulting graphs E are precisely Cayley graphs over G with generating set S whose second (normalized) eigenvalue is bounded by λ. In short, Cayley expanders are good tests. This fact, as well as the fact that most graphs in Theorem 1.2 are expanders, may cause one to be tempted to conjecture that *any* expander leads to a good homomorphism test for *any* group G. However, [13] caution that their proof works only due to the link between the algebra of the test and the algebraic structure of the graph, which needs to be a Cayley graph over the same group GZ_p^m.

Indeed, the insight that one needs a specific expander rather than an arbitrary

[1] Observe that this bound is useless unless both δ and λ are below $1/p^2$.

[2] Note that the second term is only $O(1)$ for the case where p is a fixed prime in the existential result of Theorem 1.2.

one comes from Goldreich [18] who designed a counter example for $m = 2$. Goldreich introduced a function which is very far from any linear function from Z_p^2 to Z_p, and yet passes with high probability the test defined by the Margulis graph [27, 17] (which is a Schrier graph of some group action on $G = Z_p^2$, but is *not* a Cayley graph).

Thus the question of which graphs are good testers for general groups G (and Γ) seem more subtle. Moreover, the techniques of [13] use Fourier transforms and seem to work only for abelian groups. We make significant progress for characterizing good testers for general groups, which we describe next.

1.3. Our results. In brief, we show that for every domain group G, *all* expanding Cayley graphs E on the group G are good testers for (affine) homomorphism. Since any group G has an expanding generating set of size $O(\log |G|)$ [6], our result immediately gives a nonuniform test with a near-optimal $randomness(T_E) = \log |G| + O(\log \log |G|)$. Moreover, we derandomize [6] to give a polynomial time algorithm (in $|G|$) to generate, for every group G, an expanding set of generators of size $|G|^\epsilon$, giving the randomness $(1+\epsilon)\log |G|$ explicitly and uniformly. We also note that we can find in quasi-polynomial time an expanding generating set of size $O(\log |G|)$, which implies a test with a near-optimal $randomness(T_E) = \log |G| + O(\log \log |G|)$.

We note that even our nonexplicit result is much stronger than [20], as one can efficiently verify whether a given Cayley graph is an expander and therefore good as a test graph, while Goldreich and Sudan cannot tell which of their random graphs are good. We note again that Goldreich gave an example showing that not every expander is good. We include this example in section 5.

Our testing result depends on two parameters: λ, which is the (normalized) second largest eigenvalue (in absolute value) of the Cayley graph of G with the generating set S, and δ, which is the error of the test ($\delta = error(T_{G\times S})$). We show that if S is expanding (i.e., $\lambda < 1$), then $\mathrm{dist}(T_{G\times S}) = O(\delta)$.

THEOREM 1.4. *For every G, Γ and a subset S of G, the tester that picks uniformly at random an edge $(x, xs) \in Cay(G; S)$ and checks whether $f(x)\cdot f(s) = f(xs)$ surely accepts any homomorphism $f : G \to \Gamma$, and rejects with probability at least δ any $f : G \to \Gamma$, which is $4\delta/(1-\lambda)$ far from being an affine homomorphism, provided that $\frac{12\delta}{1-\lambda} < 1$.*

Note that it follows that if f is at least $\frac{1}{3}$-far from any affine homomorphism, then f is rejected with probability at least $\frac{1-\lambda}{12}$. We also note that the test accepts any homomorphism, but rejects any function that is far from any *affine* homomorphism (rather than any function that is far from any homomorphism). It is still open to derandomize the homomorphism test of BLR. By a slight modification to the tester $T_{G\times S}$ we can get a tester that accepts any *affine* homomorphism and rejects any function that is far from any *affine* homomorphism.

THEOREM 1.5. *For every G, Γ and a subset S of G, the tester that picks uniformly at random an edge $(x, xs) \in Cay(G; S)$ and checks whether $f(x)\cdot f(1)^{-1}\cdot f(s) = f(xs)$ surely accepts any affine homomorphism $f : G \to \Gamma$, and rejects with probability at least δ any $f : G \to \Gamma$ which is $4\delta/(1-\lambda)$ far from being an affine homomorphism, given that $\frac{12\delta}{1-\lambda} < 1$. In other words, this tester is a $(\delta, \frac{4\delta}{1-\lambda})$-test for P_{aff}.*

Note that the tester of Theorem 1.5 makes 4 queries whereas the tester of Theorem 1.4 makes only 3 queries.

The proof of Theorem 1.4 uses a simple combinatorial argument together with the transitivity of groups (the proof of Theorem 1.5 is by a reduction to Theorem 1.4). Recent analysis of (variants of) the BLR test [7, 13] use some sort of Fourier transform on abelian groups. As we deal with nonabelian groups as well, we cannot use this

approach, and so rather study what may be a natural analog—the correlation of shifts of the given function with itself. It is interesting to note that the close homomorphism is defined globally, despite the fact that the tester makes only local (neighbor) tests. We also stress that our analysis avoids what seems to be an inherent problem in the Fourier approach, i.e., the relation between the Fourier coefficients and the distance to linearity is not tight (not even up to a constant factor), resulting in the suboptimal bounds of Theorem 1.3 of [13]. We note, however, that the Fourier approach has the advantage that it extends to the case where the error is relatively large, as in [13] (list decoding regime)—something we cannot do in general groups.

Our bounds are independent of the groups at hand, and thus are meaningful for constants δ and λ (in contrast to the bounds of [13]). As a consequence of our proof we get that the natural decoding procedure in which group elements correct their values according to the majority of their neighbors' values, converges to a homomorphism. It is interesting that this local decoding proof needs the global consistency in homomorphism testing. In contrast to derandomized "low degree" testers, Ben-Sasson et al. [13] derive the global consistency via iterated local decoding. It is interesting if their result has a different proof that goes along the lines of the current paper.

Our testers require expanding generators for the groups at hand. As mentioned, for abelian groups such small explicit sets were known. We next provide the first nontrivial explicit construction of expanding generating sets in every group. It is fairly weak; improving it to approach the existential bound of Alon and Roichman [6] is very interesting.

THEOREM 1.6. *For every $\epsilon > 0$ there is a polynomial time algorithm which, on input a group G, given by its multiplication table, produces a set S of size $|G|^{\epsilon}$ of expanding generators. More precisely,*

$$\lambda(Cay(G;S)) \leq O\left(|G|^{-\epsilon/8}\right).$$

Finally, combining the two theorems we have the following corollary.

COROLLARY 1.7. *For every $\epsilon > 0$ there is a polynomial time algorithm, which given any two groups G, Γ, produces a tester of randomness complexity $(1+\epsilon)\log|G|$. This tester accepts every affine homomorphism between G and Γ with probability one, and for every $\beta > 0$, rejects every function which is β-far from any such affine homomorphism, with probability $\geq 1 - \beta/5$.*

An alternative way to view our test is that we accept with probability 1 any homomorphism and reject with high probability any function that is far from any *affine* homomorphism.

2. Preliminaries.

DEFINITION 2.1 (affine homomorphism). *Let G, Γ be finite groups. A homomorphism $\phi : G \to \Gamma$ is a function with the property that for every $g, h \in G$ we have that $\phi(g \cdot h) = \phi(g) \cdot \phi(h)$. We say that ϕ is an affine homomorphism if there exists an element $\gamma \in \Gamma$ such that $\gamma \cdot \phi$ is an homomorphism. Note that in this case $\phi \cdot \gamma\gamma^{-1} \cdot (\gamma \cdot \phi) \cdot \gamma$ is also an homomorphism.*

For two functions $f_1, f_2 : G \to \Gamma$ we define

$$\mathrm{dist}(f_1, f_2) = \mathrm{Pr}_{g \in_R G}[f_1(g) \neq f_2(g)].$$

2.1. Expander graphs. Let $\mathcal{G} = (V, E)$ be a graph on n vertices. Let $A_{\mathcal{G}}$ be its adjacency matrix. For two sets $A, B \subset V$ denote

$$E(A, B) = \{(u, v) \mid u \in A \text{ and } v \in B\}.$$

Let $e(A,B) = |E(A,B)|$. Denote with $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ the eigenvalues of $A_{\mathcal{G}}$. In case that $\mathcal{G}$ is a d-regular graph we get that $\lambda_1 = d$. Denote

$$\lambda[\mathcal{G}] = \frac{1}{d} \cdot \max(\lambda_2, |\lambda_n|);$$

we sometimes use λ instead of $\lambda[\mathcal{G}]$ when $\mathcal{G}$ is clear from the context.

The next lemma due to [1] relates the edge expansion of $\mathcal{G}$ to λ.

LEMMA 2.2 (expander mixing lemma; see [1]). *For any two sets $A, B \subset V$ we have that*

$$\left| e(A,B) - d \cdot \frac{|A| \cdot |B|}{n} \right| \leq \lambda \cdot d \cdot \sqrt{|A| \cdot |B|};$$

in case $A = B^c$, we get a stronger result [34, 5].

LEMMA 2.3 (analog of the Cheeger constant; see [34, 5]).

$$\frac{1-\lambda}{2} \cdot d \leq \min_{|A| \leq n/2} \frac{e(A, A^c)}{|A|} \leq 2\sqrt{1-\lambda} \cdot d.$$

In particular we obtain the following corollary.

COROLLARY 2.4.

- *For any A we have that*

$$\min(|A|, n - |A|) \leq \frac{2}{1-\lambda} \cdot \frac{e(A, A^c)}{d}.$$

- *If we remove $2\delta dn < \frac{1-\lambda}{6} \cdot dn$ edges from the graph, then there is a connected component of size at least $\left(1 - \frac{4\delta}{1-\lambda}\right) \cdot n$.*

Proof. The first part follows immediately from Lemma 2.3. For the second part notice that if A and A^c are disconnected after the removal of the edges, then $e(A, A^c) \leq 2\delta dn$. Thus if $|A| \leq \frac{n}{2}$, then by the first part we get that

$$|A| \leq \frac{2}{1-\lambda} \cdot \frac{2\delta dn}{d} \frac{4\delta}{1-\lambda} \cdot n < n/3. \tag{1}$$

Therefore, after the removal of the edges, if we take the union of two components smaller than $\frac{n}{2}$, then the size of the union is smaller than $2n/3 < \left(1 - \frac{4\delta}{1-\lambda}\right) \cdot n$. Thus the complement of the union has size larger than $\frac{4\delta}{1-\lambda} \cdot n$ and therefore must be of size at least $\frac{n}{2}$ (because otherwise, by (1), its size is $\leq \frac{4\delta}{1-\lambda} \cdot n$). It follows that the size of the union is smaller than $\frac{4\delta}{1-\lambda} \cdot n$. By induction we get that the union of all components of size smaller than $\frac{n}{2}$ has size at most $\frac{4\delta}{1-\lambda} \cdot n$. Hence there is a large component of size $\left(1 - \frac{4\delta}{1-\lambda}\right) \cdot n$. ∎

Next we describe a simple dynamical process on graphs, which converges quickly in every (good enough) expander. The constants below are just some parameters which suffice for our purposes—clearly one can state a more general result along the same lines.

DEFINITION 2.5 (the infection process). *Let $\mathcal{G} = (V, E)$ be a d-regular graph on n vertices. Assume that initially an adversary "infects" a subset B_0 of the vertices V.*

At every subsequent time step t the infected set B_t is determined to be exactly those vertices which have at least $1/3$ fraction of their neighbors in B_{t-1}. A graph is healthy if for every initial subset B_0 of size at most $n/4$, after a finite number T of steps we have $B_T = \emptyset$.

The following is an easy consequence of the expander mixing Lemma 2.2.

COROLLARY 2.6. *Assume that $\lambda[\mathcal{G}] < 1/13$. Then $\mathcal{G}$ is healthy. Moreover, the convergence time T is at most $O(\log n)$.*

Proof. We will show that for every t $|B_t| \leq 0.9|B_{t-1}|$. By definition, the number of edges between B_t and B_{t-1} is lower bounded by $e(B_t, B_{t-1}) \geq d|B_t|/3$. Applying the expander mixing lemma to these two sets gives

$$d|B_t|/3 \leq e(B_t, B_{t-1}) \leq d|B_t| \cdot |B_{t-1}|/n + \lambda d\sqrt{|B_t| \cdot |B_{t-1}|}.$$

As $\lambda < 1/13$ and (by induction) $|B_{t-1}| \leq n/4$, we get that

$$|B_t| \leq \left(\frac{12}{13}\right)^2 |B_{t-1}| < 0.9|B_{t-1}|.$$

Iterating $O(\log n)$ times shrinks the infected set to a number smaller than 1, hence zero. ∎

2.2. Expanding Cayley graphs. Let G be a group. Let S be a generating set for G. That is, G is the minimal subgroup of G that contains all the elements of S. S is called symmetric if $s \in S \Leftrightarrow s^{-1} \in S$. We now define the Cayley graph of G with respect to a symmetric set of generators S.

DEFINITION 2.7. *Let G be a group and S a symmetric generating set for G. We define the graph* $\mathrm{Cay}(G; S)$ *as follows. The vertices are the elements of G. For every $g \in G$ and $s \in S$ we put an edge (labeled s) from g to gs.*

The definition describes $\mathrm{Cay}(G; S)$ as a directed graph, which will be one useful view of it, e.g., for describing the testers. However, since S is symmetric, if there is an edge from g to h labeled s, then there is an edge from h to g labeled s^{-1}, and both can be thought of as one undirected edge. Thus $\mathrm{Cay}(G; S)$ can also be viewed as an undirected graph (which is regular of degree $|S|$), which will be used for studying its spectral and connectivity properties.

A very nice property of Cayley graphs is their transitivity. That is, if (g_1, g_2) is an edge, then so does (gg_1, gg_2) for every g.

An interesting open problem is to deterministically find, for a given group G, a *small* symmetric generating set S, such that $\mathrm{Cay}(G; S)$ is a good expander, in time $\mathrm{poly}(|G|)$. For $G = \mathbb{Z}_2^n$ it is easy to verify that $\mathrm{Cay}(G; S)$ is an expander with second eigenvalue λ if and only if S is a λ-biased set (see [3]). However, except for some special groups [26, 27, 28] it is not known in general how to deterministically find such an S. The following result of Alon and Roichman [6] guarantees that if we pick a large enough S at random, then almost surely the associated Cayley graph is an expander.

THEOREM 2.8 (see [6]). *For every $\eta > 0$ there is a constant $c(\eta) > 0$ such that the following holds. Let G be a group of order n and let S be a symmetric set of $c(\eta) \log n$ random elements of G then, with probability at least $1 - \eta$,*

$$\lambda[\mathrm{Cay}(G; S)] < \eta.$$

This theorem assures us that we can always find an S of size $O(\log |G|)$ such that $\mathrm{Cay}(G;S)$ is an expander.

We will show that a simple "derandomization" of this argument leads to a deterministic construction of expanding generating sets of size $O(|G|^{\epsilon})$ for every group G and $\epsilon > 0$. For this, the following well known estimate via the trace formula will be very useful.

DEFINITION 2.9. *Fix G. For a (symmetric) set $S \subseteq G$ and integer m, let P_{2m} be the probability that a random word of length $2m$ in the elements of S evaluates to the identity in G.*

PROPOSITION 2.10. *For every m,*

$$\lambda[\mathrm{Cay}(G;S)]^{2m} \leq nP_{2m} - 1.$$

Proof. Let A be the adjacency matrix of $\mathrm{Cay}(G;S)$. Let $\lambda_1 \geq \cdots \geq \lambda_n$ be its eigenvalues. Let $\lambda = \lambda[\mathrm{Cay}(G;S)]$. Note that $\forall\ 1 \leq i \leq n$, $P_{2m} = \left((\frac{1}{d}A)^{2m}\right)_{i,i}$. Thus

$$nP_{2m} = \mathrm{trace}\left(\left(\frac{1}{d}A\right)^{2m}\right) = \sum_{i=1}^{n}\left(\frac{\lambda_i}{d}\right)^{2m} \geq 1 + \lambda^{2m}. \qquad \square$$

3. Derandomized homomorphism testers. We first prove Theorem 1.4. Then we show that the natural local decoding procedure (namely belief propagation) converges to a homomorphism. As it is easy to verify that every homomorphism passes the test with probability 1, we focus on the other direction—showing that any function that passed the test is close to an affine homomorphism.

3.1. Proof of Theorem 1.4. As it is easy to verify that every homomorphism passes the test with probability 1, we focus on the other direction—showing that any function that passed the test is close to an affine homomorphism.

Let G, Γ be groups such that $|G| = n$. Let f be a given function from G to Γ. Fix a symmetric $S \subseteq G$ of size $|S| = d$ and let $\lambda = \lambda[\mathrm{Cay}(G;S)]$. Consider the test that picks a random $g \in G$ and a random $s \in S$ and accepts if $f(g)f(s) = f(gs)$. Let δ be the rejection probability, i.e.,

$$\delta = \Pr_{y \in G,\ s \in S}[f(y)f(s) \neq f(ys)]. \tag{2}$$

Also assume that

$$\frac{12\delta}{1-\lambda} < 1.$$

Define the function

$$\phi(x) = \mathrm{Plurality}_{y \in G} f(xy)f(y)^{-1}.$$

We will prove that, for every x, almost all y agree on the value of $\phi(x)$ (i.e., satisfy $f(xy)f(y)^{-1} = \phi(x)$), then prove that ϕ is a homomorphism, and finally that it is close to an affine shift of f. The first of these tasks, proved in the next lemma, is perhaps the most surprising, as the test guarantees (near) local consistency, and we show it implies (near) global consistency. This claim is the main technical contribution of our proof and the rest of it follows the footsteps of the proof of [10].

LEMMA 3.1. *For every $x \in G$ we have that*

$$\Pr_{y \in G}[f(xy)f(y)^{-1} = \phi(x)] \geq 1 - \frac{4\delta}{1-\lambda}.$$

Proof. Fix $x \in G$. Note that some constructs in this proof will depend on x and later we will use them for all values of x. From (2) we have for random $y \in G$

$$\delta = \Pr_{y \in G,\, s \in S}[f(xy)f(s) \neq f(xys)]. \tag{3}$$

We now construct a subgraph of $\mathrm{Cay}(G;S)$. Call the edge (y, ys) bad for x if either $f(y)f(s) \neq f(ys)$ or $f(xy)f(s) \neq f(xys)$. By (2) and (3) the number of bad edges is at most $2\delta dn$. Consider the subgraph H_x obtained by removing all (undirected) edges that are bad for x. By the expansion of $\mathrm{Cay}(G;S)$, and since we remove only $2\delta dn$ edges, we get by Corollary 2.4 that H_x contains a connected component C_x of size at least $\left(1 - \frac{4\delta}{1-\lambda}\right) n$.

By connectivity and the fact that the remaining edges satisfy the test, we get that for all y in that component the value $f(xy)f(y)^{-1}$ is constant. We prove it formally as it is a bit subtle.

PROPOSITION 3.2. *For every two distinct elements $v, u \in C_x$ we have $f(xv)f(v)^{-1} = f(xu)f(u)^{-1}$.*

Proof. Let $v = v_1, v_2, \ldots, v_t = u$ be a path between v and u in C_x. Let $s_i = v_i^{-1}v_{i+1}$ be the generator labeling the ith edge (i.e., the ith edge is $(v_i, v_i s_i)$). For every i, the existence of the edge (v_i, v_{i+1}) in H_x implies by definition the existence of the edge (xv_i, xv_{i+1}) in H_x as well. Since all these edges are good for x, it follows that

$$f(v_i)^{-1}f(v_{i+1}) = f(s_i) = f(xv_i)^{-1}f(xv_{i+1})$$

for all i. Thus

$$f(v)^{-1}f(u) = f(v_1)^{-1}f(v_t) = \prod_{i=1}^{t-1} f(v_i)^{-1}f(v_{i+1}) = \prod_{i=1}^{t-1} f(s_i)$$

$$= \prod_{i=1}^{t-1} f(xv_i)^{-1}f(xv_{i+1}) = f(xv_1)^{-1}f(xv_t) = f(xv)^{-1}f(xu).$$

By changing sides we get that $f(xv)f(v)^{-1} = f(xu)f(u)^{-1}$ as required. ∎

Thus, $f(xy)f(y)^{-1}$ is the same for all $y \in C_x$. As $|C_x| > |G|/2$, and we have defined ϕ using plurality over y, we get $\phi(x) = f(xy)f(y)^{-1}$ for every $y \in C_x$. Since $|C_x| \geq (1 - \frac{4\delta}{1-\lambda})n$, Lemma 3.1 follows. ∎

LEMMA 3.3. *ϕ is a homomorphism.*

Proof. We need to show that for every $x, y \in G$ we have that $\phi(x)\phi(y) = \phi(xy)$. Consider (like [10]) arbitrary $x, y \in G$ and the probability over $h \in G$

$$\Pr_{h \in G}[\phi(x)\phi(y) = \phi(xy)] \tag{4}$$

which is independent of h, and thus is either 0 or 1. We prove that this probability is positive and therefore 1. We lower bound it by the probability of the intersection of three events over the same random variable h chosen uniformly in G

$$\Pr_{h \in G}[\phi(x)\phi(y) = \phi(xy)] \geq \Pr_{h \in G}\left[\begin{array}{c} \phi(x) = f(xh)f(h)^{-1} \quad \text{and} \\ \phi(y) = f(h)f(y^{-1}h)^{-1} \quad \text{and} \\ \phi(xy) = f(xh)f(y^{-1}h)^{-1} \end{array}\right]$$

since in this case $\phi(x)\phi(y) = f(xh)f(h)^{-1}f(h)f(y^{-1}h)^{-1} = f(xh)f(y^{-1}h)^{-1} = \phi(xy)$. Now

- By Lemma 3.1 $\Pr_{h\in G}[\phi(x) = f(xh)f(h)^{-1}] \geq 1 - \frac{4\delta}{1-\lambda}$.
- Notice that

$$\Pr_{h\in G}[\phi(y) = f(h)f(y^{-1}h)^{-1}] = \Pr_{h\in G}[\phi(y) = f(y\cdot(y^{-1}h))f(y^{-1}h)^{-1}].$$

  As h is random this probability equals

$$\Pr_{h'\in G}[\phi(y) = f(yh')f(h')^{-1}]$$

  which by Lemma 3.1 is at least $1 - \frac{4\delta}{1-\lambda}$.
- Similarly we get that

$$\Pr_{h\in G}[\phi(xy) = f(xh)f(y^{-1}h)^{-1}] = \Pr_{h\in G}[\phi(xy) = f((xy)\cdot(y^{-1}h))f(y^{-1}h)^{-1}]$$

$$\geq 1 - \frac{4\delta}{1-\lambda}.$$

Note that each of these events have probability of at least $1 - \frac{4\delta}{1-\lambda}$, and so the probability of their intersection is at least $1 - \frac{12\delta}{1-\lambda}$ which is strictly positive and so must be 1. □

Finally we show that f is close to some affine shift of ϕ.

LEMMA 3.4. *There exists $\gamma \in \Gamma$ such that*

$$\Pr_{x\in G}[\phi(x)f(x)\cdot\gamma] \geq 1 - \frac{4\delta}{1-\lambda}.$$

Proof. For every $x \in G$ denote with G_x the set of ("good") y's satisfying $\phi(x) = f(xy)f(y)^{-1}$ (note that the set G_x contains the set C_x defined in the proof of Lemma 3.1, but may in fact be even larger). By Lemma 3.1, for every x it holds that $|G_x| \geq (1 - \frac{4\delta}{1-\lambda})|G|$. It follows by averaging that there exist $y \in G$ such that

$$|\{x \,:\, y \in G_x\}| \geq \left(1 - \frac{4\delta}{1-\lambda}\right)|G|.$$

For this y we have that

$$\Pr_{x\in G}[\phi(x) = f(xy)f(y)^{-1}] \geq 1 - \frac{4\delta}{1-\lambda}.$$

Therefore

$$\Pr_{x'\in G}[\phi(x'y^{-1}) = f(x')f(y)^{-1}] \geq 1 - \frac{4\delta}{1-\lambda}.$$

As ϕ is a homomorphism we get that

$$\Pr_{x'\in G}[\phi(x') = f(x')f(y)^{-1}\phi(y)] \geq 1 - \frac{4\delta}{1-\lambda}.$$

The lemma follows by defining $\gamma = f(y)^{-1}\phi(y)$.

This completes the proof of Theorem 1.4. □

3.2. Proof of Theorem 1.5. Again it is easy to see that if f is an affine homomorphism, then f passes the test with probability 1. To prove the other direction we define a new function $f' : G \to \Gamma$ in the following way:

$$f'(x) \stackrel{\triangle}{=} f(1)^{-1} \cdot f(x).$$

It is obvious that $f'(x) \cdot f'(y) = f'(xy)$ if and only if $f(x) \cdot f(1)^{-1} \cdot f(y) = f(xy)$. In particular the probability of success of the tester of Theorem 1.5 on f equals the probability of success of the tester of Theorem 1.4 on f'. Assume that f passed the test with probability $\geq \delta$ for δ as in the statement of the theorem. Theorem 1.4 implies that f' is $\frac{4\delta}{1-\lambda}$ close to some affine homomorphism $\gamma \cdot \phi$. Hence f is $\frac{4\delta}{1-\lambda}$ close to the affine homomorphism $f(1) \cdot \gamma \cdot \phi$. □

3.3. Iterated local majority decoding. Recall that the close homomorphism in the proof above was defined according to a *global* majority: every group element x chose the value $\phi(x)$ according to the plurality of $f(xy)f(y)^{-1}$ over *all* group elements $y \in G$. We show that iterated *local* majority decoding, where (in each phase) every group element x updates its value according to the plurality of $f(xs)f(s^{-1})$ over its neighbors in the Cayley graph, converges to (the same) global homomorphism ϕ.

DEFINITION 3.5 (iterated majority decoding). *Let G, Γ be groups, S a subset of G and $f : G \to \Gamma$ any function. Set $f = f_0$ and for every integer t define f_t by*

$$f_t(x) = \text{Plurality}_{s \in S} f_{t-1}(xs) f_0(s^{-1}).^3$$

THEOREM 3.6. *Let $G, \Gamma, S, \lambda, \delta, \gamma$ be as in Theorem 1.4 and further assume that $\lambda, \delta \leq 1/17$. Let $f : G \to \Gamma$ be such that the tester $\text{Cay}(G; S)$ accepts f with probability of at least $1 - \delta$. Then the iterated decoding procedure above converges to a homomorphism $\phi : G \to \Gamma$ in $O(\log |G|)$ steps. Moreover, ϕ is a conjugate of the homomorphism defined in the proof of Theorem 1.4, and is at most $4\delta/(1-\lambda)$-far from $\gamma \cdot f$.*

Proof. By Theorem 1.4 we get that there is a homomorphism ϕ such that[4] $\text{dist}(\phi, \gamma \cdot f) \leq \frac{4\delta}{1-\lambda}$ for some $\gamma \in \Gamma$. Let f_t be the sequence of functions defined by the iterated majority decoding procedure above, and let D_t denote the set of group elements on which $\gamma \cdot f_t$ and ϕ disagree. By our choice of parameters,

$$|D_0| \leq \frac{4\delta}{1-\lambda} |G| < |G|/4.$$

We reduced the analysis of the local decoding to that of the infection process in 2.6 at the end of section 2.1.

Set $B_0 = D_0$ and apply the infection process to it, to obtain a sequence B_t. We show by induction that for every t, we have $D_t \subseteq B_t$ so the theorem follows from Corollary 2.6. Assume for the moment that for all but 1/6 fraction of the $s \in S$ we have $f(s) = \phi(s)$. Then every element x in step t which has a 2/3 of its neighbors in the complement of B_{t-1}, gets the same value from at least $\frac{2}{3} - \frac{1}{6} = \frac{1}{2}$ of them (as $D_{t-1} \subset B_{t-1}$), and this value agrees with ϕ, namely $\gamma \cdot f_t(x) = \phi(x)$.

[3] Note that we keep using the *initial* values on S in all iterations.

[4] In the proof of Theorem 1.4 we found ϕ that was close to $f \cdot \gamma$, this implies that $\phi'(x) \stackrel{\triangle}{=} \gamma \cdot \phi(x) \cdot \gamma^{-1}$ is close to $\gamma \cdot f$.

We now argue that for at most 1/6 fraction of $s \in S$ it is the case that $f(s) \neq \phi(s)$. Fix any "bad" s for which $f(s) \neq \phi(s)$. Since $|D_0| < |G|/4$, for at least 1/2 of all the elements $x \in G$ we have both $\gamma \cdot f(x) = \phi(x)$ and $\gamma \cdot f(xs) = \phi(xs)$. All these pairs x, s are rejected by the tester, and since it rejects only a δ fraction of all such pairs, the number of bad s is at most $2\delta < 1/6$. □

Note that the proof relies on the fact that f passed the test with high probability. It is not sufficient that f is close to an homomorphism: consider the constant function $f = \gamma$ for some $\gamma \neq 1$. It is clear that f is an affine homomorphism and that f does not pass the test $T_{G \times S}$. We get that $f_t = \gamma^{t+1}$, where f_t is defined by the iterative process above. Clearly f_t does not converge to 1_Γ (the identity element of Γ), which is the homomorphism close to $\gamma^{-1} \cdot f$.

4. Explicit expanding generators–Proof of Theorem 1.6. In this section we give a polynomial time algorithm to find a relatively small expanding generating set in every group. We state the main technical result, which is nearly identical to Theorem 2.8 of Alon and Roichman, except adding the condition that the choices of the generators need not be fully independent. The proof remains identical to their proof, only we'll need it with different parameters. We give the proof for completeness.

DEFINITION 4.1. *A set $\mathcal{A} \subset [n]^d$ is a k-wise independent sample space if for any subset $I \subset [d]$ of size $|I| = k$, and any sequence $(g_1, ..., g_k) \in [n]^k$, we have that*

$$\Pr_{a \in_R \mathcal{A}}[a_I = (g_1, ..., g_k)] = \frac{1}{n^k},$$

where a_I denotes the restriction of the d-tuple a to the set of coordinates I.

There are many works showing how to construct k-wise independent sample spaces efficiently [2, 14, 16].

THEOREM 4.2 (see [2, 14, 16]). *There is a deterministic algorithm, which on input n, d, k outputs a k-wise independent sample space in time $\max(n, d)^{O(k)}$ (this also implies that the size of the set is $\max(n, d)^{O(k)}$).*

For the rest of the section we fix a group G of size n.

THEOREM 4.3. *Fix any integer $m \geq 2$. Consider the following distribution on Cayley graphs on G. Let $\mathcal{A}$ be a $2m$-wise independent sample space of d-tuples from G^d. Draw a random sample $(g_1, \ldots, g_d)$ from $\mathcal{A}$ to form a (multi)set $T = \{g_1, ..., g_d\}$, and let $S = T \cup T^{-1}$. Then the expectation of $\lambda(\mathrm{Cay}(G; S))$ is*

$$E[\lambda(\mathrm{Cay}(G; S))] < (2n)^{1/2m}(16m/d)^{1/4}.$$

Proof. We repeat the essentials of the proof of [6], with the only difference being the limited independence of the generators. This turns out not to change the analysis. We skip easy proofs which can be obtained from their paper.

By Proposition 2.10 and Jensen's inequality we get that

$$E[\lambda(\mathrm{Cay}(G; S))] < \left(nE[P_{2m}] - 1\right)^{1/(2m)},$$

where P_m was defined in Definition 2.9. Thus, it suffices to prove that $E[P_{2m}] \leq 1/n + 2(16m/d)^{m/2}$. In order to bound P_{2m} we construct a random word of length $2m$ in three steps.

- Pick a random word W' of length $2m$ in the alphabet $\{a_1, a_1^{-1}, \ldots, a_d, a_d^{-1}\}$.
- Reduce the word over the free group on d generators to obtain the word W.
- Replace every a_i by the associated random g_i from T.

The upper bound on the expectation of P_{2m} will follow from the following three probability estimates.

LEMMA 4.4.

$$\Pr[|W| < m] \leq (32/d)^{m/2}.$$

LEMMA 4.5. *Call W bad if none of the d letters*[5] *appears exactly once in W. Condition on $|W| \geq m$. Then*

$$\Pr[W \text{ bad}] \leq (16m/d)^{m/2}.$$

LEMMA 4.6. *Fix any good w, and replace each a_i by g_i as above to generate the word $w(T)$ in G. Then*

$$\Pr[w(T) = 1_G] = 1/n.$$

We prove only the last lemma, since this is the only point where the limited independence of T could make a difference. The first two lemmas follow from [6] after an adjustment of the parameters.

Proof. Since w is good, there is some generator, say a_1 w.l.o.g., which occurs exactly once in w. There are at most $2m-1$ other generators a_i in w. For each of these, expose their g_i value. Now the probability in question is the probability that g_1 equals a fixed group element determined by the exposed g_i's and w. But g_1 is completely uniform given these choices, and so that probability is precisely $1/n$. ∎

The proof now follows as the expectation of P_{2m} is bounded by the sum of the probabilities of the events in the lemmas above. This concludes the proof of Theorem 4.3. ∎

COROLLARY 4.7. *Take $d = n^{4/m}$. Then*

$$E[\lambda(\mathrm{Cay}(G;S))] < 3m^{1/4}d^{-1/8}.$$

Finally we show how to choose a set of generators deterministically, establishing Theorem 1.6. Given $\epsilon > 0$, we set $m = 4/\epsilon$, and $d = n^{4/m} = n^{\epsilon}$. Apply Theorem 4.2 to construct a sample space of size at most $poly(n^m)$ of d-tuples over G which are $(2m)$-wise independent. This takes polynomial time in n^m (remember that m is a constant). For each such tuple T compute (again in polynomial time, as all we need is a reasonable approximation) the associated $\lambda(\mathrm{Cay}(G;S))$, and returns the set S for which this eigenvalue is smallest.

This concludes the proof of Theorem 1.6. ∎

5. Not every expander is good. In this section we present a construction, due to Goldreich, of an expander graph on a group (but not a Cayley graph), for which the natural tester fails miserably.

Let p be a prime and consider the (Schreier) graph H_p describing the action of the group $SL_2(p)$ on the vector space Z_p^2, with generators S being the two matrices

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

and their inverses.

[5] a_i and a_i^{-1} are considered the same letter.

More concretely, the vertices of H_p are $(x, y) \in Z_p \times Z_p$, and the four neighbors of (x, y) are $(x, y \pm x)$ and $(x \pm y, y)$. Note that H_p has two connected components—the vertex $(0, 0)$ and the rest. Thus H_p has two eigenvalues of value 1, and we denote here by $\lambda(H_p)$ the maximum absolute value of any of the other eigenvalues.

The graph H_p is a variant of the famous Margulis graph—the first explicit expander. The expansion of (the large component of) H_p follows directly from the expansion of the Cayley graph $\mathrm{Cay}(SL_2(p); S)$, which follows from Selberg's celebrated 3/16 Theorem (see Lubotzky [25] for details).

THEOREM 5.1 (see [25]). *For every p, $\lambda(H_p) \leq 13/16$.*

We will consider functions from Z_p^2 to Z_p. Since the groups are abelian, we will write them additively.

For defining the tester (and the function), it will be convenient to view each undirected edge as directed "positively." In other words every vertex $v = (x, y)$ has two directed edges emanating from it: to $u = (x, y + x)$ (labeled by $u - v = (0, x)$) and to $w = (x + y, y)$, (labeled by $w - v = (y, 0)$).

Observe that in our graph, labels of edges always have 0 in one of their components. Also note that there are roughly $2p$ distinct labels, despite the graph having degree 4—this is very different from a Cayley graph (in which the number of labels is the degree).

In this notation, the tester associated to this graph, picks uniformly at random a (directed edge) from v to u and tests if $f(u) - f(v) = f(u - v)$.

We now present the example that beats this tester. It will be very far from any affine homomorphism, but will pass the test with probability close to 1.

Consider the function $f : Z_p \times Z_p \to Z_p$ defined as follows. $f(x, y) = x^2$ if $y = 0$, $f(x, y) = y^2$ if $x = 0$, and $f(x, y) = x \cdot y$ otherwise (with all arithmetic in Z_p).

THEOREM 5.2. *For the function f defined above we have that*

- *f is $(1 - 4/p)$-far from any affine homomorphism.*
- *f passes the test with probability $1 - 4/p$.*

Proof. First we prove the first item in the theorem. Every affine homomorphism g from Z_p^2 to Z_p looks like $g(x, y) \to ax + by + c$ for some constants $a, b, c \in Z_p$. Consider only pairs $x, y \neq 0$, as only $2/p$ of the pairs (x, y) are not of this form. We want to count the number of possible solutions to the equation $xy = ax + by + c$. When $x = b$ there can be p solutions. For every other possible value of x we get a (different) nonconstant linear equation in y, which has at most one solution. So for every possible affine homomorphism g we have $\mathrm{dist}(f, g) \geq 1 - 4/p$, as required.

Now we prove the second item in the theorem. Only $8p - 4$ of the $2p^2$ directed edges have a 0 component in either of their endpoints. Thus with probability of at least $1 - 4/p$ the chosen neighboring vertices v, u have no zero component. We show that all these edges pass the test. Let $v = (x, y)$. There are 2 similar cases. First take $u = (x, y + x)$. Then

$$f(u) - f(v) = x(y + x) - xy = x^2 = f(u - v).$$

Now take $w = (x + y, y)$. Then

$$f(w) - f(v) = (x + y)y - xy = y^2 = f(w - v). \qquad \square$$

Acknowledgments. We thank Eli Ben-Sasson and Salil Vadhan for many illuminating discussions and for reading and commenting earlier versions of the manuscript. We thank Oded Goldreich for many valuable discussions and for suggestions that improved the presentation of the results. We also thank Oded for kindly allowing us

to include his counterexample here. We thank the anonymous referees for their comments.

REFERENCES

[1] N. Alon and F. R. K. Chung, *Explicit construction of linear sized tolerant networks*, Discrete Math., 72 (1988), pp. 15–19.

[2] N. Alon, L. Babai, and A. Itai, *A fast and simple randomized parallel algorithm for the maximal independent set problem*, J. Algorithms, 7 (1986), pp. 567–583.

[3] N. Alon, O. Goldreich, J. Hastad, and R. Peralta, *Simple constructions of almost k-wise independent random variables*, Random Structures Algorithms, 3 (1992), pp. 289–304.

[4] N. Alon and Y. Mansour, *ϵ-Discrepancy sets and their applications for interpolation of sparse polynomials*, Inform. Process. Lett., 54 (1995), pp. 337–342.

[5] N. Alon and V. D. Milman, *Eigenvalues, Expanders, and Superconcentrators (extended abstract)*, in Proceedings of the 25th Annual Symposium on Foundations of Computer Science, Singer Island, FL, 1984, pp. 320–322.

[6] N. Alon and Y. Roichman, *Random Cayley graphs and expanders*, Random Structures Algorithms, 5 (1994), pp. 271–284.

[7] M. Bellare, D. Coppersmith, J. Håstad, M. Kiwi, and M. Sudan, *Linearity testing over characteristic two. Codes and Complexity*, IEEE Trans. Inform. Theory, 42 (1996), pp. 1781–1795.

[8] M. Bellare, S. Goldwasser, C. Lund, and A. Russell, *Efficient probabilistic checkable proofs and applications to approximation*, in Proceedings of the 25th Annual ACM Symposium on the Theory of Computing, San Diego, CA, 1993, pp. 294–304.

[9] M. Bellare and M. Sudan, *Improved non-approximability results*, in Proceedings of the 26th Annual ACM Symposium on the Theory of Computing, Montreal, Quebec, Canada, 1994, pp. 184–193.

[10] M. Ben-Or, D. Coppersmith, M. Luby, and R. Rubinfeld, *Nonabelian homomorphism testing*, in Approximation, Randomization, and Combinatorial Optimization—Algorithms and Techniques (Random-Approx 2004), Lecture Notes in Comput. Sci. 3122, Cambridge, MA, 2004, pp. 273–285.

[11] M. Bellare, O. Goldreich, and M. Sudan, *Free bits, PCP, and nonapproximability—towards tight results*, SIAM J. Comput., 27 (1998), pp. 804–915.

[12] M. Blum, M. Luby, and R. Rubinfeld, *Self-testing/correcting with applications to numerical problems*, J. Comput. System Sci., 47 (1993), pp. 549–595.

[13] E. Ben-Sasson, M. Sudan, S. Vadhan, and A. Wigderson, *Randomness-efficient low degree tests and short PCPs via epsilon-biased sets*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, San Diego, CA, 2003, pp. 612–621.

[14] B. Chor and O. Goldreich, *On the power of two-point based sampling*, J. Complexity, 5 (1989), pp. 96–106.

[15] F. R. K. Chung, *Diameters and eigenvalues*, J. AMS, 2 (1989), pp. 187–196.

[16] G. Even, O. Goldreich, M. Luby, N. Nisan, and B. Velickovic, *Approximations of general independent distributions*, in Proceedings of the 24th Annual ACM Symposium on the Theory of Computing, Victoria, BC, Canada, 1992, pp. 10–16.

[17] O. Gabber and Z. Galil, *Explicit constructions of linear size superconcentrators*, in Proceedings of the 20th Annual Symposium on Foundations of Computing Science, New York, 1979, pp. 364–370.

[18] O. Goldreich, *Private communication*, 2002.

[19] O. Goldreich, *Combinatorial Property Testing*, DIMACS Ser. Discrete Math. Theoret. Comput. Sci., Vol. 43, AMS, 1999, pp. 45–59.

[20] O. Goldreich and M. Sudan, *Locally testable codes and PCPs of almost-linear length*, in Proceedings of the 43rd Annual Symposium on Foundations of Computer Science, Vancouver, BC, Canada, 2002, pp. 13–22.

[21] J. Håstad, *Some optimal inapproximability results*, J. ACM, 48 (2001), pp. 798–859.

[22] J. Håstad and A. Wigderson, *Simple analysis of graph tests for linearity and PCP*, Random Structures Algorithms, 22 (2003), pp. 139–160.

[23] R. Impagliazzo and A. Wigderson, *P=BPP unless E has subexponential circuits: Derandomizing the XOR lemma*, in Proceedings of the 29th Annual ACM Symposium on the Theory of Computing, El Paso, TX, 1997, pp. 220–229.

[24] N. M. Katz, *An estimate for character sums*, J. AMS, 2 (1989), pp. 197–199.

[25] A. Lubotzky, *Discrete Groups, Expanding Graphs, and Invariant Measures*, Progress in

Math. 125, Birkhaüser Verlag, Basel, Switzerland, 1994.
[26] A. Lubotzky, R. Phillips, and P. Sarnak, *Ramanujan graphs*, Combinatorica, 8 (1988), pp. 261–277.
[27] G. A. Margulis, *Explicit constructions of concentrators*, Problemy Peredači Informacii, (1973), pp. 71–80.
[28] M. Morgenstern, *Existence and explicit constructions of $q+1$ regular Ramanujan graphs for every prime power q*, J. Combin. Theory Ser. B, 62 (1994), pp. 44–62.
[29] J. Naor and M. Naor, *Small-bias probability spaces: Efficient constructions and applications*, SIAM J. Comput., 22 (1993), pp. 838–856.
[30] D. Ron, *Property Testing*, Handbook of Randomized Computing, S. Rajasekaran, P. M. Pardalos, J. H. Reif, and J. D. P. Rolim, eds., Kluwer Press, Dordrecht, The Netherlands, 2001.
[31] R. Rubinfeld and M. Sudan, *Robust characterization of polynomials with applications to program testing*, SIAM J. Comput., 25 (1996), pp. 252–271.
[32] A. Samorodnitsky and L. Trevisan, *A PCP characterization of NP with optimal amortized query complexity*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, Portland, OR, 2000, pp. 191–199.
[33] M. Sudan, L. Trevisan, and S. Vadhan, *Pseudorandom generators without the XOR lemma*, J. Comput. System Sci., 62 (2001), pp. 236–266.
[34] R. M. Tanner, *Explicit concentrators from generalized N-gons*, SIAM J. Algebraic Discrete Methods, 5 (1984), pp. 287–293.

SIAM J. COMPUT.
Vol. 36, No. 5, pp. 1231–1247

DETERMINISTIC EXTRACTORS FOR BIT-FIXING SOURCES AND EXPOSURE-RESILIENT CRYPTOGRAPHY*

JESSE KAMP† AND DAVID ZUCKERMAN‡

Abstract. We give an efficient deterministic algorithm that extracts $\Omega(n^{2\gamma})$ almost-random bits from sources where $n^{\frac{1}{2}+\gamma}$ of the n bits are uniformly random and the rest are fixed in advance. This improves upon previous constructions, which required that at least $n/2$ of the bits be random in order to extract many bits. Our construction also has applications in exposure-resilient cryptography, giving explicit adaptive exposure-resilient functions and, in turn, adaptive all-or-nothing transforms. For sources where instead of bits the values are chosen from $[d]$, for $d > 2$, we give an algorithm that extracts a constant fraction of the randomness. We also give bounds on extracting randomness for sources where the fixed bits can depend on the random bits.

Key words. extractors, randomness, deterministic, bit-fixing sources, exposure-resilient, cryptography, resilient function, random walks

AMS subject classifications. 68Q10, 94A60, 68W20

DOI. 10.1137/S0097539705446846

1. Introduction.

True randomness is needed for many applications, such as cryptography. However, most physical sources of randomness are not even close to being truly random, and may in fact seem quite weak in that they can have substantial biases and correlations. A natural approach to dealing with the problem of weak physical sources is to apply a *randomness extractor*—a function that transforms a weak random source into an almost uniformly random source. For certain natural notions of such random sources, it has been shown that it is impossible to devise a single function that extracts even one bit of randomness [32]. One way to combat this problem is to allow the use of a small number of uniformly random bits as a catalyst in addition to the bits from the weak random source. Objects constructed in this manner, known as seeded extractors [28], have been shown to extract almost all of the randomness from general weak random sources (see [33] for a recent survey).

However, we would like to eliminate the need for the random catalyst by restricting the class of weak random sources for which we need our function to work. Following the lead of Trevisan and Vadhan [34], we call such functions deterministic extractors for the given class of sources. More formally, we say that a function is an ϵ-extractor for a class of sources if the output of the function is ϵ-close to uniform (in variation distance) for all sources in the class.

1.1. Bit-fixing and symbol-fixing sources.

The particular class of sources that we are interested in are bit-fixing sources, in which some subset of the bits are fixed and the rest are chosen at random. There are two classes of bit-fixing sources,

*Received by the editors January 19, 2005; accepted for publication (in revised form) March 12, 2006; published electronically December 21, 2006. A preliminary version of this paper has appeared in *IEEE Symposium on Foundations of Computer Science*, 2003, pp. 92–101.
http://www.siam.org/journals/sicomp/36-5/44684.html

†Department of Computer Science, University of Texas, Austin, TX 78712 (kamp@cs.utexas.edu).
The research of this author was supported in part by NSF grants CCR-9912428 and CCR-0310960.

‡Department of Computer Science, University of Texas, Austin, TX 78712 (diz@cs.utexas.edu).
The research of this author was supported in part by a David and Lucile Packard Fellowship for Science and Engineering, NSF grants CCR-9912428 and CCR-0310960, a Radcliffe Institute Fellowship, and a Guggenheim Fellowship.

depending on whether the fixed bits are chosen before or after the random bits are determined, known respectively as oblivious and nonoblivious bit-fixing sources. We will construct extractors for both classes.

Extractors for oblivious bit-fixing sources were first studied in [11], in which they considered the case of exactly uniform output. They proved that at least $n/3$ random bits are needed to extract even two bits from an input of length n. Friedman generalized this result to obtain bounds on the number of random bits needed for longer outputs [16]. The large amount of randomness needed to obtain exactly uniform resilient functions led to the consideration of relaxing this restriction to allow for almost uniform output. We note that even when we allow the extractor to have small error, the best previous constructions still required that at least half of the bits be random [23, 4].

We are able to improve on these constructions by outputting $\Omega(n^{2\gamma})$ bits when the input has at least $n^{\frac{1}{2}+\gamma}$ random bits.

THEOREM 1.1. *For any $\gamma > 0$ and any constant $c > 0$, there exists an ϵ-extractor $f : \{0,1\}^n \to \{0,1\}^m$ for the set of oblivious bit-fixing sources with $n^{\frac{1}{2}+\gamma}$ random bits, where $m = \Omega(n^{2\gamma})$ and $\epsilon = 2^{-cm}$. This extractor is computable in a linear number of arithmetic operations on m-bit strings.*

We can even extract some bits when there are fewer random bits, although we get a much shorter output.

THEOREM 1.2. *There exists an ϵ-extractor $f : \{0,1\}^n \to \{0,1\}^{\frac{1}{4}\log k}$, for the set of oblivious bit-fixing sources with k random bits, where $\epsilon = \frac{1}{2}k^{\frac{1}{4}}\exp(-\frac{\pi^2\sqrt{k}}{2})$. This extractor is computable in a linear number of arithmetic operations on $\frac{1}{4}\log k$ bits.*

In addition to studying oblivious bit-fixing sources, we introduce the related model of d-ary oblivious symbol-fixing sources (SF sources). Such a source consists of a string of symbols over a d symbol alphabet where k of the symbols are random and the rest are fixed. This model is somewhat more restricted than the bit-fixing model. For example, for $d = 2$, this model is the same as the oblivious bit-fixing model, and for $d = 4$, it corresponds to oblivious bit-fixing sources where the fixed and random bits have to come in pairs. However, it is still an extremely natural and interesting model.

For SF sources with $d > 2$, we get much better results than for oblivious bit-fixing sources. We extract a constant fraction of the randomness for sources with any number of random symbols, with the constant depending on d. In particular, as d grows large we can extract almost all of the randomness.

THEOREM 1.3. *For every $d > 2$ there exists a $c_d > 0$ such that for every n and k, there exists an ϵ-extractor $f : [d]^n \to [d]^m$ for the set of d-ary SF sources with k random symbols that outputs $m = c_d\ k - O(\log_d(1/\epsilon))$ symbols, where $c_d \to 1$ as $d \to \infty$. This extractor is computable in a linear number of arithmetic operations on m-symbol strings.*

Another interesting related class of sources for which deterministic extraction is possible are nonoblivious bit-fixing sources [3, 21]. In such sources, the fixed bits can depend on the random bits chosen. This problem was originally studied in the context of collective coin flipping [3], which can be viewed as extraction of a single bit. For the single bit case, nearly optimal lower [21] and upper [2] bounds are known, though the upper bound is not completely constructive. However, little attention has previously been given to generalizing these results to the case of multiple output bits. We give bounds for this case. If $\ell = n - k$ is the number of fixed bits in the source, we show that at most n/ℓ bits can be extracted from these sources, which is likely to be nearly optimal. We also give a construction of an ϵ-extractor for nonoblivious

bit-fixing sources which outputs $\Omega((\epsilon/\ell)^{\log_2 3} \cdot n)$ bits.

1.2. Exposure-resilient cryptography. Our work has applications in cryptography. In traditional cryptography, secret keys are required to remain secret. Most cryptographic schemes have no security guarantees even when an adversary learns only a small part of the secret key. Is it possible to achieve security even when the adversary learns most of the secret key? The class of mappings known as all-or-nothing transforms (AONT), introduced by Rivest [30], address this issue. An AONT is an efficient randomized mapping that is easy to invert given the entire output, but where an adversary would gain "no information" about the input even if it could see almost the entire output of the AONT. Various important applications of the AONT have been discovered, such as the previously mentioned application of protecting against almost complete exposure of secret keys [10], and increasing the efficiency of block ciphers [27, 20, 5].

Boyko used the random-oracle model to give the first formalizations and constructions of the AONT [9]. Canetti et al. gave the first constructions in the standard computational model [10]. For their construction, they introduced a new, related primitive known as an exposure-resilient function (ERF). An ERF is an efficiently computable deterministic function where the output looks random even if the adversary obtains almost all of the bits of a randomly chosen input. They then reduced the task of constructing an AONT to constructing an equivalent ERF. This work was extended by Dodis, Sahai, and Smith [15] to the adaptive setting, where the adversary can decide which bits to look at based on the bits he has already seen. This setting is applicable to the problem of partial key exposure, where it is likely that the adversary would be adaptive.

An important idea used in both [10] and [15] is that we can construct ERF's in the computational setting by first constructing ERF's in the statistical setting and then applying a pseudorandom generator to the output. This allows us to get longer output lengths, which is useful for applications. Because of this observation, we can restrict our attention to constructing ERF's in the statistical setting, where the output must be statistically close to the uniform distribution. However, though [15] gives a probabilistic construction of adaptive statistical ERF's, the problem of giving an explicit construction was left open (see also [14]).

We address this problem by giving an explicit construction of efficient adaptive ERF's in the statistical setting, which in turn gives an explicit construction of adaptive AONT's. Our construction actually gives a stronger function, known as an almost-perfect resilient function (APRF), introduced in [23]. An APRF is like an ERF, except it works for even the case where the adversary can fix some bits of the input instead of merely looking at them. The connection between APRF's and exposure resilient cryptography was shown in [15], where it was proved that APRF's are also adaptive ERF's. In fact, it is easy to see that APRF's are essentially the same as deterministic extractors for oblivious bit-fixing sources. So by constructing extractors for oblivious bit-fixing sources, we will also get APRF's and thus adaptive statistical ERF's and AONT's.

1.3. Overview of our constructions. We now give an overview of our various extractor constructions along with an outline of the rest of the paper.

Our extractor for d-ary SF sources involves using the input symbols to take a random walk on a d-regular expander graph, starting from an arbitrary start vertex. The extractor then outputs the label of the final vertex on the walk. We show that even though we allow some of the steps to be fixed in advance, corresponding to the

fixed bits of the source, these steps will not hurt us. Therefore the random walk behaves essentially like a random walk on the random steps only. Because of the rapid mixing properties of expanders, this output will be close to uniform, and we can extract a linear fraction of the entropy, thus proving Theorem 1.3. For $d = 2$, we cannot use an expander graph since expanders only exist for degree $d > 2$, but we show that if we take a random walk on a cycle we can still extract some bits, proving Theorem 1.2; we give these constructions in section 3.1. We also note that similar types of random walks have been used in previous pseudorandomness constructions [1, 12, 19].

For oblivious bit-fixing sources, we show that we can extract even more bits by first converting the sources into sources that are close to SF sources, which we call approximate symbol-fixing (approx-SF) sources, and then applying the expander walk extractor. This gives the extractor from Theorem 1.1. We show in section 3.2 that our extractor for SF sources also works for approx-SF sources. To convert the oblivious bit-fixing source into a d-ary approx-SF source, we partition the input into blocks. For each block, we take a random walk on the d-cycle and output the label of the final vertex. Enough of the blocks will have enough random bits so that enough of the symbols are almost random. We note that the symbols in the output source have constant error, so we can't just add the errors from the almost random steps since they are too large. Because of this conversion step, we "lose" some of the randomness, which is why we require that the number of random bits be greater than $\sqrt{n}$ in Theorem 1.1. In section 4, we show how to do the conversion and prove that the extractor works.

In section 5, we show the relation between our extractors for oblivious bit-fixing sources and exposure-resilient cryptography.

We give our results for nonoblivious bit-fixing sources in section 6. For such sources, let $\ell = n - k$ be the number of fixed bits. We show that at most n/ℓ bits can be extracted from these sources using a generalization of the edge isoperimetric inequality on the cube. This is likely to be nearly optimal, as it almost corresponds to applying known single bit functions to blocks of the input. In particular, we can use any function with low "influence" [3]. Our best explicit construction uses the iterated majority function of Ben-Or and Linial [3] and outputs $\Omega((\epsilon/\ell)^{\log_2 3} \cdot n)$ bits. However, there are nonexplicit constructions that give bounds within a polylogarithmic factor of our edge isoperimetric bound [2].

1.4. Subsequent work. Since this paper first appeared, Gabizon, Raz, and Shaltiel [18] have improved upon our constructions of extractors for oblivious bit-fixing sources. Using our extractors as building blocks, they are able to extract almost all of the randomness from oblivious bit-fixing sources. Unfortunately, however, the error they achieve is not good enough for our application of constructing adaptive ERF's.

2. Preliminaries. For ease of notation, we sometimes assign noninteger values to integer variables when we mean to round off the values. It is easy to observe that any errors introduced in this manner do not affect our results.

We frequently write our definitions in terms of a single function f, though we really mean for f to represent a family of functions over all input lengths, so asymptotic notions make sense.

2.1. Probability definitions. We need some standard definitions for probability distributions. First, we express our probability distributions as probability vectors

$p = (p_1, ..., p_n)$ with $\sum_i p_i = 1$. Unless otherwise stated, π represents the uniform probability vector (of the appropriate length). The *variation (statistical) distance* $|p - q|$ between two distributions with probability vectors p and q is half the ℓ_1 distance, so $|p - q| = \frac{1}{2}\sum_i |p_i - q_i|$. Also, we use $\|.\|$ to represent the standard ℓ_2 norm for vectors. It is well known that $|p - q| \leq \frac{1}{2}\sqrt{n}\|p - q\|$.

A *source* is a family of probability distributions (a probability ensemble). For ease of notation, we usually refer to a source as a single probability distribution.

2.2. Extractor definitions. Trevisan and Vadhan studied what would happen if you removed the random catalyst from ordinary extractors, and they called such functions deterministic extractors [34]. Deterministic extractors for general weak sources are impossible, and they're even impossible for semirandom sources [32]. However, if we restrict our attention to certain classes of weak sources, then the problem becomes tractable. The following definition of a deterministic extractor is taken from [14], which is implicit in the definitions of [34].

DEFINITION 2.1. *An efficiently computable function $f : \{0,1\}^n \to \{0,1\}^m$ is an ϵ-extractor for a set of random sources $\mathcal{X}$, if for every $X \in \mathcal{X}$, $f(X)$ is within variation distance ϵ of uniform.*

The sets of sources we use are the sets of oblivious bit-fixing [11], symbol-fixing, and nonoblivious bit-fixing sources [3]. Oblivious bit-fixing sources are the easiest to handle, since the fixed bits do not depend on the random bits.

DEFINITION 2.2 (see [11]). *An (n, k) oblivious bit-fixing source X is a source with n bits, of which all but k are fixed and the rest are then chosen uniformly at random.*

DEFINITION 2.3. *An (n, k, d) oblivious SF source X is a source with n independent symbols each taken from $[d]$, of which all but k are fixed and the rest are then chosen uniformly at random.*

Note that for $d = 2^t$, SF sources can be viewed as a special case of bit-fixing sources where the bits are divided up into blocks of size t and each block is either fixed or random.

Nonoblivious bit-fixing sources are more difficult to handle, since the fixed bits can depend on the random bits.

DEFINITION 2.4 (see [3]). *An (n, k) nonoblivious bit-fixing source X is a source with n bits, of which k are chosen uniformly at random and then the remaining $n - k$ bits are chosen, possibly depending on the random bits.*

We will need a slightly weaker notion of symbol-fixing sources when converting bit-fixing sources to symbol-fixing sources.

DEFINITION 2.5. *An (n, k, d, ϵ) approximate oblivious symbol-fixing (approx-SF) source X is a source with n symbols independently chosen from $[d]$, of which k have distributions within an ℓ_2 distance of ϵ of uniform.*

2.3. Graph definitions. We define some standard notions used when studying random walks on graphs. Transition matrices indicate the probability of following any edge in a random walk. A (general) *transition matrix* P for a graph $G = (V, E)$ with n vertices is an $n \times n$ matrix with entries $p_{ij} \geq 0$ if $(i, j) \in E$ and $p_{ij} = 0$ otherwise, and $\sum_{j=1}^{n} p_{ij} = 1$ for all rows i. The *uniform transition matrix* P of a d-regular graph $G = (V, E)$ has all nonzero entries equal to $1/d$. The way to view these definitions is that the probability of choosing edge (i, j) if we are currently at vertex i corresponds to p_{ij}. The *stationary probability vector* π for a random walk with transition matrix P is the vector such that $\pi P = \pi$, and is well defined for connected graphs. In the cases we will look at, π corresponds to the uniform distribution on the vertices.

For each random walk, the input is a string of values, each of which can take on any value in $[d]$, where d is the degree of the graph. A directed edge (u, v) is *labeled* i if (u, v) is the edge taken when the random walk is at u and receives input value i.

One property that we need in our graphs is that the error shouldn't accumulate in any of the vertices. In order for our graphs to have this property, we require that no vertex has two incoming edges with the same label. Such a graph is said to be *consistently labeled*. All of our results apply only to consistently labeled graphs.

An expander graph is a graph that has low degree, but is well connected, so that random walks on expanders converge quickly to the uniform distribution. For a given matrix P, let $\lambda(P)$ denote the second largest eigenvalue in absolute value. Here we define expanders in terms of $\lambda(P)$.

DEFINITION 2.6. *A family of expander graphs is an infinite set of regular graphs G with uniform transition matrix P that have $\lambda(P) = 1 - \epsilon$ for some constant $\epsilon > 0$.*

We will need all of our expander graphs that we use to be efficiently constructible, that is, we should find the neighbors of any vertex in polynomial time in the length of the vertex label. There are various constructions that give infinite families of constant-degree consistently labeled expander graphs that are efficiently computable; see, e.g., [17, 25, 26, 29]. Though these constructions don't work for every degree, we can always construct an expander for a given degree by adding an appropriate number of self loops to an existing expander. It is easy to see that doing so maintains the eigenvalue separation. We also should note that there are expander constructions that work for degrees as small as 3.

3. Constructing extractors for SF and approx-SF sources. In this section, we first show how to construct deterministic extractors for SF sources. We will then show how this construction can be extended to extract from approx-SF sources. We will use the construction for approx-SF sources in the next section to show how we can extract from oblivious bit-fixing sources.

3.1. Extracting from SF sources. In this section, we prove the following generalization of Theorem 1.3 to show that we can extract a constant fraction of the randomness from SF sources.

THEOREM 3.1. *For any $k = k(n)$, ϵ and $d > 2$, if there exists an efficiently computable d-regular expander with $\lambda(P) \leq d^{-\alpha}$ on d^m vertices, for $m \leq 2\alpha k - \frac{2}{\log d} \log \frac{1}{2\epsilon}$, then there exists an efficiently computable ϵ-extractor for the set of (n, k, d) SF sources which outputs m symbols.*

The extractor works by taking a walk on an expander with d^m vertices starting at a fixed vertex and using the input symbols as steps. The output is the label of the final vertex.

We get extractors with the longest output length when we use Ramanujan expanders, for which $\lambda(P) = 2\sqrt{(d-1)}/d$. For certain parameters, there exist efficiently computable Ramanujan graphs [26, 25]. Note that for Ramanujan graphs, as d grows large, α approaches $1/2$, so the output length approaches k.

For $d = 2$, we can't use an expander, but we can use the symbols to take a walk on the cycle to get an extractor for oblivious bit-fixing sources that extracts a small number of bits from any source regardless of k. Note that we're restricted to using odd size cycles here, since random walks on even cycles don't converge to uniform, as they alternate between the even and odd vertices.

THEOREM 3.2. *For odd d, there exists an ϵ-extractor $f : \{0,1\}^n \to [d]$, for the set of (n, k) oblivious bit-fixing sources, where $\epsilon = \frac{1}{2}\sqrt{d}\exp(-\frac{\pi^2 k}{2d^2})$. This extractor is*

computable in a linear number of arithmetic operations on $\log d$ *bits.*

Note that for this extractor to be useful, we must have $\log d < \frac{1}{2}\log k$, which shows that we can output only a small amount of the original randomness with this technique. In particular, if we take $d = k^{\frac{1}{4}}$, we get Theorem 1.2.

Both Theorems 3.1 and 3.2 arise from the following key lemma.

LEMMA 3.3. *Let* P *be a uniform transition matrix with stationary distribution* π *for an undirected nonbipartite* d*-regular graph* G *on* M *vertices. Consider an* n *step walk on* G*, with the steps taken according to the symbols from an* (n, k, d) *SF source* X*. For any initial probability distribution* $p = v + \pi$*, the distance from uniform at the end of the walk is bounded by*

$$\left| p \prod_{i=1}^{n} P_i - \pi \right| \leq \frac{1}{2} \| p \prod_{i=1}^{n} P_i - \pi \| \sqrt{M} \leq \frac{1}{2} \lambda(P)^k \sqrt{M}.$$

To prove this lemma, we show that the random symbols from the source bring us closer to uniform and also that the fixed symbols don't bring us any further away.

For the random steps, it is well known that the distance can be bounded in terms of $\lambda(P)$. This gives the following lemma, a proof of which can be found in [24].

LEMMA 3.4. *Let* P *be a uniform transition matrix for an undirected,* d*-regular graph* G*. Then for any probability vector* $p = v + \pi$*,*

$$\| pP - \pi \| \leq \lambda(P) \| v \|.$$

In our case, most of the steps in our random walks will be fixed. The consistent labeling property ensures that the transition matrix for these fixed steps will be a permutation matrix. Thus these steps leave the distance from uniform unchanged, and so we get the following lemma.

LEMMA 3.5. *Let* P *be a transition matrix for a fixed step on an undirected,* d*-regular graph* G*. Then for any probability vector* $p = v + \pi$*,*

$$\| pP - \pi \| = \| v \|.$$

Now, using the previous two lemmas, we can prove Lemma 3.3.

Proof of Lemma 3.3. For the random symbols we can apply Lemma 3.4. Since there are k random symbols, this gives us the $\lambda(P)^k$ factor. We also use that by Lemma 3.5 the steps corresponding to the fixed symbols don't increase the distance from uniform. Combining both the random and the fixed steps together with the relation between the variation and ℓ_2 distance and the fact that the $\|v\| \leq 1$, we get the stated bound. □

Now we can use Lemma 3.3 to prove Theorem 3.1.

Proof of Theorem 3.1. We can apply Lemma 3.3, where in this case $\lambda(P) \leq d^{-\alpha}$ and $M = d^m$. Thus the error $\epsilon \leq \frac{1}{2} d^{-\alpha k + (m/2)}$. Taking logarithms and solving for m, we get the stated bound on m. □

Now, using Lemma 3.3, we can prove Theorem 3.2. We first separate out the following lemma which will be useful later.

LEMMA 3.6. *Let* P *be a uniform transition matrix for the random walk on the* d*-cycle for* d *odd. Suppose the length of the walk is* n*, with the steps taken according to the symbols from an* (n, k) *oblivious bit fixing source* X*. For any initial probability distribution* $p = v + \pi$*, the distance from uniform at the end of the walk is bounded by*

$$\left| p \prod_{i=1}^{n} P_i - \pi \right| \leq \frac{1}{2} \| p \prod_{i=1}^{n} P_i - \pi \| \sqrt{d} \leq \frac{1}{2} (\cos(\pi/d))^k \sqrt{d}.$$

Proof. The lemma follows from Lemma 3.3 and the fact that the d-cycle has $\lambda(P) = \cos(\pi/d)$ (see [13]). □

Proof of Theorem 3.2. The extractor outputs the result of a random walk on the d-cycle. By Lemma 3.6, this will be within $\frac{1}{2}\sqrt{d}(\cos(\pi/d))^k$ of uniform. Since $\cos(\pi/d) \leq \exp(-\frac{\pi^2}{2d^2})$ (see [13, p. 26]), we get the desired error. □

There is one slight difficulty, since we may want to use a family of expander graphs (or cycles) that includes graphs that don't have exactly 2^m vertices. (In fact, in the cycle case, we can't use any even sized cycle.) This difficulty can be overcome by outputting the result of the random walk on a much larger graph modulo 2^m. The following lemma shows that doing so has little impact on the error.

LEMMA 3.7. *If a random variable X is within ϵ of uniform over $[N]$, then the random variable $Y = X \mod M$ is within $\epsilon + 1/r$ of uniform over $[M]$, where $r = \lfloor N/M \rfloor$.*

Proof. Divide the $y \in [M]$ up into two classes, those corresponding to r different $x \in [N]$ with $y = x \mod M$ and those corresponding to $r+1$ different $x \in [N]$. The probability that Y assigns to each y is then either r/N or $(r+1)/N$, plus the corresponding part of the original error ϵ. Since $r/N \leq 1/M \leq (r+1)/N$, the additional error introduced for each y when going from X to Y is at most $1/N$. So the total additional error introduced is at most $M/N \leq 1/r$. □

3.2. Extracting from approx-SF sources. We now show how the previous construction can be extended to handle the case of approx-SF sources. Our main result in this section is the following variant of Lemma 3.3 for approx-SF sources.

LEMMA 3.8. *Let P be a uniform transition matrix with stationary distribution π for an undirected nonbipartite d-regular graph G on M vertices. Suppose we take a walk on G for n steps, with the steps taken according to the symbols from an (n, k, d, ϵ) approx-SF source X. For any initial probability distribution $p = v + \pi$, the distance from uniform at the end of the walk is bounded by*

$$\left| p \prod_{i=1}^{n} P_i - \pi \right| \leq \frac{1}{2} \| p \prod_{i=1}^{n} P_i - \pi \| \sqrt{M} \leq \frac{1}{2} (\lambda(P) + \epsilon\sqrt{d})^k \sqrt{M}.$$

In the case of approx-SF sources, the random steps in our random walk will be only almost uniformly random. This introduces some small amount of error into our transition matrix. We can separate out the error terms by dividing up our new transition matrix P' into the uniform transition matrix P and an error matrix E, which is defined as follows.

DEFINITION 3.9. *An ϵ-error matrix E for a d-regular graph G is a matrix with the following properties. If $|E_{ij}| > 0$, then (i, j) is an edge in G; all of the columns of E sum to 0; and the ℓ_2 norm of each column of E is at most ϵ.*

For slightly nonuniform random steps, we can modify the bound from Lemma 3.4 slightly to get the following lemma.

LEMMA 3.10. *Let P be a uniform transition matrix for an undirected, d-regular graph G. Let E be an ϵ-error matrix for G. Now let $P' = P + E$ be our modified transition matrix. Then P' has the same stationary distribution π as P and for any probability vector $p = v + \pi$,*

$$\|pP' - \pi\| \leq (\lambda(P) + \epsilon\sqrt{d})\|v\|.$$

Proof. Because π is uniform and because each of the columns of E sum to 0 by definition, $\pi E = 0$. Thus $\pi P' = \pi P + \pi E = \pi$ by the above observation combined with the stationarity of π with respect to P. Thus P' has stationary distribution π.

Now we bound $\|pP' - \pi\|$. We first observe that $\|pP' - \pi\| = \|vP' + \pi P' - \pi\| = \|vP'\|$ since we know from above that π is stationary. Now we can focus on bounding $\|vP'\|$. By the triangle inequality $\|vP'\| \leq \|vP\| + \|vE\|$. We know that $\|vP\| \leq \lambda(P)\|v\|$. Letting e_{ij} denote the entries of E, we get

$$\begin{aligned}\|vE\| &= \left(\sum_j \left(\sum_{i;e_{ij}\neq 0} e_{ij}v_i\right)^2\right)^{\frac{1}{2}} \\ &\leq \left(\sum_j \left(\sum_{i;e_{ij}\neq 0} e_{ij}^2\right)\left(\sum_{i;e_{ij}\neq 0} v_i^2\right)\right)^{\frac{1}{2}} \\ &\leq \epsilon \left(\sum_j \sum_{i;e_{ij}\neq 0} v_i^2\right)^{\frac{1}{2}} \leq \epsilon\sqrt{d}\|v\|,\end{aligned}$$

where the first line is simply from the definition, and noting that we only need to sum over all nonzero e_{ij}. The second line follows from the Cauchy–Schwarz inequality. The third line follows from the fact that the sum of the square of the errors e_{ij}^2 over any column is at most ϵ^2. The final inequality comes from the fact that e_{ij} can only be nonzero when ij corresponds to an edge in G. Since there are d edges adjacent to i, we will have at most d v_i^2 terms in the sum for each i.

Putting everything together we get the desired bound on $\|pP' - \pi\|$. ∎

Unlike in the case of SF sources, the nonrandom steps may not be fixed, but may simply not have enough randomness in them. However, we would still like to show that these steps do not take us further from the uniform distribution. The following lemma shows that since any step chosen according to a symbol from a d-ary source is a convex combination of permutations, the nonrandom steps in our random walk don't increase the distance from uniform. Note that this result depends on our assumption that the graph G is consistently labeled.

LEMMA 3.11. *Let P be a transition matrix for a step chosen according to a symbol X_j from a d-ary source X. Then P is a convex combination of permutation matrices and for any probability vector $p = v + \pi$, $\pi P = P$, and $\|pP - \pi\| \leq \|v\|$.*

Proof. First we show that P is a convex combination of permutation matrices. Every possible value from $i \in [d]$ for x gives a permutation matrix P_i. If X_j is distributed with probabilities α_i for each $i \in [d]$, then $P = \sum_{i=0}^{d-1} \alpha_i P_i$, which is a convex combination of permutation matrices.

Then note that since any permutation of π is still uniform, we have $\pi P_i = \pi$ and thus $\pi P = P$. This gives us $\|pP - \pi\| = \|vP\|$. We bound $\|vP\|$ by the triangle inequality as $\|vP\| \leq \sum_i \alpha_i \|vP_i\| = \sum_i \alpha_i \|v\| = \|v\|$, where the second inequality follows from the fact that since P_i is a permutation, $\|vP_i\| = \|v\|$. ∎

Using the previous two lemmas, we can prove Lemma 3.8.

Proof. Let P_i be the transition matrix of the random walk at the ith step. By Lemma 3.11 P_i is a convex combination of permutation matrices and $\pi P_i = \pi$. This gives us $\pi \prod_{i=1}^n P_i = \pi$, so $p\prod_{i=1}^n P_i - \pi = v\prod_{i=1}^n P_i$.

Let $v_j = \prod_{i=1}^j P_i$. Then $v_j = v_{j-1}P_j$, and $v_0 = v$. For k of the steps, the symbols are within an ℓ_2 distance of ϵ from uniform, which implies $P_j = P + E_j$, where every column of E_j has ℓ_2 norm at most ϵ. Since G is consistently labeled, the sum of each column of E_j is equal to 0, so E_j is indeed an error matrix. So for these steps, by

Lemma 3.10, $\|v_{j-1}P_j\| \leq (\lambda(P) + \epsilon\sqrt{d})\|v_{j-1}\|$. For the other steps, we still have by Lemma 3.11 that $\|v_{j-1}P_j\| \leq \|v_{j-1}\|$. So for k steps the ℓ_2 norm is reduced while for the rest of the steps it, at worst, remains the same. Thus

$$\left\| p\prod_{i=1}^{n} P_i - \pi \right\| = \left\| v\prod_{i=1}^{n} P_i \right\| \leq (\lambda(P) + \epsilon\sqrt{d})^k \|v\|.$$

Now apply the bound relating the ℓ_2 norm and variation distance and $\|v\| \leq 1$. ∎

4. From SF sources to oblivious bit-fixing sources. In this section, we show how to extend our results for SF sources to oblivious bit-fixing sources to get the following theorem, which is basically a restatement of Theorem 1.1. Though we state the theorem for general values of δ, we have in mind the case $\delta n = n^{\frac{1}{2}+\gamma}$.

THEOREM 4.1. *For any positive $\delta = \delta(n) \leq 1$ and any constant $c > 0$, there exists an ϵ-extractor $f : \{0,1\}^n \to \{0,1\}^m$, for the set of $(n, \delta n)$ oblivious bit-fixing sources, where $m = \Omega(\delta^2 n)$ and $\epsilon = 2^{-cm}$. This extractor is computable in a linear number of arithmetic operations on m-bit strings.*

There are two main steps in the extractor construction. First, we transform the source into an approx-SF source by dividing it into blocks. For each block we take a random walk on the cycle and output the label of the final vertex on the walk. The approx-SF source is then the concatenation of these outputs. Then we use the expander walk extractor from the previous section to extract from the approx-SF source.

We start by applying Lemma 3.6 to our degree 2 walks on the d-cycle for each of the blocks. We will show that enough of the blocks mix to within ϵ' of the uniform distribution, for some ϵ'. This process gives us an approx-SF source.

LEMMA 4.2. *For any odd d, any $(n, \delta n)$ oblivious bit-fixing source can be deterministically converted into a $(\frac{\delta n}{2t}, \frac{\delta^2 n}{4t}, d, \epsilon)$ approx-SF source, where $t = \lceil \frac{\log \epsilon}{\log(\cos(\pi/d))} \rceil$.*

The almost random symbols in the approx-SF source correspond to blocks where we have "enough" random bits. Using a Markov-like argument, we can quantify how many such blocks we will have, as shown in the following lemma.

LEMMA 4.3. *Suppose we have n bits from an (n, k) oblivious bit-fixing source, where $k = \delta n$. For any partition of the n bits into $\delta n/2t$ blocks of size $2t/\delta$, the number r of blocks with at least t random bits satisfies $r > \frac{\delta^2 n}{4t}$.*

Proof. We know that in the r blocks with at least t random bits there are at most $2t/\delta$ random bits. In the remaining blocks there are less than t random bits. Combining these two facts we get that the total number of random bits $k < 2rt/\delta + t((n\delta/2t) - r)$, which after a simple calculation gives the desired result. ∎

Using this lemma, we can now prove Lemma 4.2.

Proof of Lemma 4.2. Divide the input r up into $\delta n/2t$ blocks of size $2t/\delta$. Then take a random walk on a d-cycle using the bits from each block and output the vertex label of the end vertex for each walk. These vertex labels are the symbols for our approx-SF source. We call a block good if this random walk reaches within an ℓ_2 distance of ϵ from uniform, which means the corresponding symbol is good for our source. By Lemma 3.6, if there are at least t random bits in the block the ℓ_2 distance from uniform is at most $(\cos(\pi/d))^t \leq \epsilon$, which means all such blocks are good. Then by Lemma 4.3, the number of good blocks r satisfies $r > \frac{\delta^2 n}{4t}$. Thus the output source is an approx-SF source with the appropriate parameters. ∎

The symbols from the approx-SF source then correspond to our almost random steps in the expander graph, so we can apply Lemma 3.8 to the expander walk to get

that the final distribution is close to uniform.

Proof of Theorem 4.1. If $\delta = O(1/\sqrt{n})$, we can take f to be the parity function, since in this case outputting a single bit is enough. Otherwise, let G be a d-regular expander graph on 2^m vertices with uniform transition matrix P. Choose ϵ' so that $\lambda_{\epsilon'} = \lambda(P) + \epsilon'\sqrt{d} < 1$. Then use the procedure in Lemma 4.2 to convert the $(n, \delta n)$ oblivious bit-fixing source to a $(\frac{\delta n}{2t}, \frac{\delta^2 n}{4t}, d, \epsilon')$ approx-SF source, where $t = \lceil (\log \epsilon')/(\log(\cos(\pi/d))) \rceil$.

Now we use the approx-SF source to take a random walk on G. We take the label of the final vertex of the walk on G as the output $f(r)$. Then we can apply Lemma 3.8, which states that the variation distance from uniform of $f(r)$ is at most

$$\frac{1}{2}\lambda_{\epsilon'}^r 2^{m/2} < \lambda_{\epsilon'}^{\frac{\delta^2 n}{4t}} 2^{m/2}.$$

We want this to be at most $\epsilon = 2^{-cm}$, so setting $m = b\delta^2 n$ for some constant $b > 0$ and taking the logarithm, we get $\frac{1}{4t}\log\frac{1}{\lambda_{\epsilon'}} \geq b\left(c + \frac{1}{2}\right)$. The left-hand side of this inequality is just some positive constant, so for any given value of c we can select b so that the inequality is satisfied. These constants give the desired output length and the desired error ϵ.

Since there are a linear number of expander steps and there exist expanders that take a constant number of arithmetic operations per step, f is computable in a linear number of arithmetic operations on m-bit strings. □

Note that in the last proof we only needed a bound on the ℓ_2 distance, which from the proof of Lemma 3.8 is tighter than the bound on the variation distance, but this difference only affects the constants in the theorem.

5. Exposure-resilient cryptography. We now discuss the needed background from exposure-resilient cryptography and how our extractor for oblivious bit-fixing sources can be used to get better statistical adaptive ERF's and AONT's.

There are a few different types of resilient functions that we define, taken from [15], each of which involve making the output look random given an adversary with certain abilities. For all of these definitions, f is a polynomial time computable function $f : \{0,1\}^n \to \{0,1\}^m$. Also, there is a computationally unbounded adversary $\mathcal{A}$ that has to distinguish the output of f from a uniformly random string $R \in \{0,1\}^m$. A function $\epsilon(n)$ is said to be *negligible* if $\epsilon(n) = O(\frac{1}{n^c})$ for all constants c.

Adaptive k-ERFs are defined as functions that remain indistinguishable from uniform even by adversaries that can adaptively read most of the input.

DEFINITION 5.1 (see [15]). *An adaptive k-ERF is a function f where, for a random input r, when $\mathcal{A}$ can adaptively read all of r except for k bits, $|\Pr[\mathcal{A}^r(f(r)) = 1] - \Pr[\mathcal{A}^r(R) = 1]| \leq \epsilon(n)$ for some negligible function $\epsilon(n)$.*

Our goal is to construct adaptive ERF's. We might first think that any $\epsilon(n)$-extractor for oblivious bit-fixing sources would work as long as $\epsilon(n)$ is negligible. However, [15] show that there are functions that are oblivious bit-fixing extractors but not adaptive ERF's. To solve this problem, they use a stronger condition which they show is sufficient. This condition is that every single output value has to occur with almost uniform probability. Functions that satisfy this stronger condition are the APRFs (first stated in section 1.2), introduced by Kurosawa, Johansson, and Stinson [23].

DEFINITION 5.2 (see [23]). *A $k = k(n)$ APRF is a function f where, for any setting of $n - k$ bits of the input r to any fixed values, the probability vector p of the*

output $f(r)$ over the random choices for the k remaining bits satisfies $|p_i - 2^{-m}| < 2^{-m}\epsilon(n)$ for all i and for some negligible function $\epsilon(n)$.

THEOREM 5.3 (see [15]). *If f is a k-APRF, then f is an adaptive k-ERF.*

The following lemma shows that any extractor for oblivious bit-fixing sources with small enough error is also an APRF. We use this lemma to show that the extractor we constructed earlier is also an APRF, and hence an adaptive k-ERF.

LEMMA 5.4. *Any $2^{-m}\epsilon(n)$-extractor $f : \{0,1\}^n \to \{0,1\}^m$ for the set of (n,k) oblivious bit-fixing sources, where $\epsilon(n)$ is negligible, is also a k-APRF.*

Proof. Since f is an extractor, the total variation distance from uniform of the output of f when $n-k$ bits of the input are fixed is within $2^{-m}\epsilon(n)$. Thus the distance of any possible output from uniform must also be within $2^{-m}\epsilon(n)$, and the APRF property is satisfied. ∎

Now using this lemma we get the following theorem.

THEOREM 5.5. *For any positive constant $\gamma \leq 1/2$, there exists an explicit k-APRF $f : \{0,1\}^n \to \{0,1\}^m$, computable in a linear number of arithmetic operations on m-bit strings, with $m = \Omega(n^{2\gamma})$ and $k = n^{\frac{1}{2}+\gamma}$.*

Proof. Apply Lemma 5.4 to the extractor from Theorem 4.1, choosing $c > 1$. ∎

We can use adaptive ERFs to construct AONTs, which were introduced by Rivest [30] and extended to adaptive adversaries by Dodis, Sahai, and Smith [15]. We first give a formal definition of AONTs. There are two parts to the definition. First, the AONT is an efficient randomized mapping that is easily invertible given the entire output. Second, an adversary gains negligible information about the input to the AONT even when it can read almost the entire output. This is formalized by the adversary not being able to distinguish between any two distinct inputs. Note that the output of the AONT has two parts. We call the first part of the output the secret part and the second part of the output the public part.

DEFINITION 5.6 (see [15]). *A polynomial time randomized transformation $T : \{0,1\}^m \to \{0,1\}^s \times \{0,1\}^p$ is a statistical adaptive k-AONT if*

1. *T is invertible in polynomial time.*
2. *For any adversary $\mathcal{A}$ who has oracle access to string $y = (y_s, y_p)$ and is required not to read at least k bits of y_s, and for any $x_0, x_1 \in \{0,1\}^m$ and some negligible function $\epsilon(s+p)$:*

$$|\Pr[\mathcal{A}^{T(x_o)}(x_0,x_1) = 1] - \Pr[\mathcal{A}^{T(x_1)}(x_0,x_1) = 1]| \leq \epsilon(s+p).$$

The following lemma from [15] relates adaptive k-ERF's to adaptive k-AONT's, and shows that our construction gives adaptive k-AONT's.

THEOREM 5.7 (see [15]). *If $f : \{0,1\}^n \to \{0,1\}^m$ is an adaptive k-ERF, then $T(x) = \langle r, x \oplus f(r)\rangle$ is a statistical adaptive k-AONT with secret part r and public part $x \oplus f(r)$.*

By combining Theorem 5.7 with Theorem 5.5, we get the following theorem.

THEOREM 5.8. *For any positive constant $\gamma \leq 1/2$, there exists an explicit function $f : \{0,1\}^n \to \{0,1\}^m$ computable in a linear number of arithmetic operations on m-bit strings, with $m = \Omega(n^{2\gamma})$, such that $T(x) = \langle r, x \oplus f(r)\rangle$ is a statistical adaptive k-AONT with secret part r and public part $x \oplus f(r)$.*

6. Extracting from nonoblivious bit-fixing sources. In this section, we switch our focus to nonoblivious bit-fixing sources, where the fixed bits can depend on the random bits. We give upper and lower bounds for extracting from such sources.

Previous bounds on nonoblivious bit-fixing sources have been defined in terms of the "influence" of variables on a function [3]. The influence of a set of variables S on a

function f, denoted $I_f(S)$, is the probability that if the variables not in S are chosen randomly, the function remains undetermined. The following two lemmas show that the influence of a function is related to the variation distance of the function from uniform when the input comes from a nonoblivious bit-fixing source. The first lemma shows that having low influence for all sets of a given size implies that a function is an extractor, while the second lemma shows that a function that has a set with high influence cannot be an extractor.

LEMMA 6.1. *Suppose $f : \{0,1\}^n \to \{0,1\}^m$ maps the uniform distribution U_n to U_m and $I_f(S) \leq \epsilon$ for all sets S of ℓ variables. Then f is an ϵ-extractor for the set of $(n, n-\ell)$ nonoblivious bit-fixing sources.*

Proof. Let X be an $(n, n-\ell)$ nonoblivious bit-fixing source and let S denote the set of fixed variables of X. Since $I_f(S) \leq \epsilon$, for all but an ϵ fraction of the choices for the random bits in X, f has the same distribution regardless of whether the rest of the bits are chosen according to X or according to U_n. Thus the variation distance is at most ϵ. □

LEMMA 6.2. *Let S be a set of ℓ variables. If, for some $\epsilon > 0$, $I_f(S) = \epsilon$, then there exists an $(n, n-\ell)$ nonoblivious bit-fixing source X with set of fixed variables S so that*

$$|f(X) - U_m| \geq \epsilon/4.$$

Proof. View the possible outputs as vertices of a hypergraph on 2^m vertices. Look at all possible values of the $n-\ell$ bits not in S. Since $I_f(S) = \epsilon$, we know that an ϵ fraction of these values leave f undetermined. For each such value, place a hyperedge between all possible output values of f (when going over all possible values for the bits in S).

Eliminate all of the vertices with no edges. Now divide all of the remaining vertices at random into two sets of equal size, A and B. The expected number of hyperedges in the cut between A and B is at least half the total number of hyperedges, so there exists a pair of sets with at least this many hyperedges. Consider such A and B, and look at only the hyperedges in the cut. Now each of these hyperedges corresponds to a setting of the $n-\ell$ bits not in S. So we define two $(n, n-\ell)$ nonoblivious bit-fixing sources X_A and X_B based on how the values of the bits in S are set for each cut hyperedge. Define X_A (X_B) by setting the bits in S for each cut hyperedge so that the output of f lies in A (B). Since these hyperedges have total probability at least $\epsilon/2$, these sources will differ by at least $\epsilon/2$. Thus at least one of them will differ by at least $\epsilon/4$ from the uniform distribution. □

Using Lemma 6.1, we immediately see that known constructions of Boolean functions with low influence [3, 2] are extractors. To get longer output length, we show that we can construct an extractor that extracts several bits from any Boolean function with small influence. The extractor simply works by applying the low influence function to blocks of the input and concatenating the resulting output bits.

LEMMA 6.3. *Suppose there exists a function $g : \{0,1\}^s \to \{0,1\}$, with expectation $1/2$, and a value $r(s)$ such that any set S of $\ell(s,\epsilon) = \epsilon r(s)$ variables has $I_g(S) \leq \epsilon$ for all $\epsilon > 0$. Then there exists an ϵ-extractor $f : \{0,1\}^n \to \{0,1\}^m$ for the set of $(n, n-\ell(s,\epsilon))$ nonoblivious bit-fixing sources that extracts $m = n/s$ bits.*

Proof. Divide the input into $m = n/s$ blocks of size s. The jth output bit of f will be g applied to the jth block. Fix a set S. By Lemma 6.1 we need to show that f has $I_f(S) \leq \epsilon$ for all sets S of $\ell = \ell(s,\epsilon)$ variables. Let ℓ_i be the number of bits in S in block i and set $\epsilon_i = \ell_i / r(s)$. The influence for each output bit is

then at most ϵ_i. Now we note that since the random bits for each of these functions are chosen independently, the total influence is at most the sum of the influences for each of these Boolean functions. Thus, since $\sum_{i=1}^{m} \epsilon_i = (\sum_{i=1}^{m} \ell_i)/r(s) = \ell/r(s) = \epsilon$, $I_f(S) \leq \epsilon$. □

We can apply this lemma to the iterated majority function of Ben-Or and Linial [3] to get an explicit extractor for nonoblivious bit-fixing sources.

THEOREM 6.4 (see [3]). *For every s, there is an explicit construction of functions $g : \{0,1\}^s \to \{0,1\}$, with expectation $1/2$, where any set S of $\ell(s,\epsilon) = \epsilon\left(\frac{s}{3}\right)^{\alpha}$ variables has $I_g(S) \leq \epsilon$ for every $\epsilon > 0$, where $\alpha = \log_3 2$.*

THEOREM 6.5. *For every n, we can construct an ϵ-extractor $f : \{0,1\}^n \to \{0,1\}^m$ for the set of $(n, n-\ell)$ nonoblivious bit-fixing sources that extracts $m = \frac{1}{3}(\epsilon/\ell)^{1/\alpha} n$ bits, where $\alpha = \log_3 2$.*

Proof. Apply Lemma 6.3 using the function from Theorem 6.4. □

Ajtai and Linial [2] give hope for improvement since their functions allow $\Omega(s/\log^2 s)$ fixed bits. However, their construction is nonexplicit, and a bound like that in Lemma 6.3 is only known to hold for $\epsilon \geq 1/\operatorname{polylog}(s)$ [31].

In the other direction, we now show that at most n/ℓ bits can be extracted from nonoblivious bit-fixing sources. To do so, we generalize the edge-isoperimetric bound from [3].

LEMMA 6.6. *For every function $f : \{0,1\}^n \to \{0,1\}^m$ with output within ϵ of uniform on uniform input, the expected influence over all sets of variables S of size ℓ is at least*

$$1 - 2\frac{\binom{n-m+1}{l}}{\binom{n}{l}} - 2\epsilon.$$

Proof. View all 2^n possible inputs as vertices of the n dimensional cube. Color the vertices of the cube with 2^m colors, where the color of x corresponds to $f(x)$. Now for each possible set S of size ℓ and setting of the remaining $n-\ell$ random variables, there is a corresponding subcube of dimension ℓ in the cube. Note that f is undetermined over such a subcube if and only if the subcube is not monochromatic. So the average influence over all possible S is the probability that a randomly chosen ℓ dimensional subcube is not monochromatic. We divide the set of colors into two classes, those with at most 2^{n-m+1} vertices and those with more, which we call "small" and "large."

First, we handle the large colors. Let t be the number of large colors. Each of these t colors contributes at least 2^{-m} to the error ϵ of f with uniform input, so $t \leq \epsilon 2^m$. Since the distance from uniform is at most ϵ, the total number of vertices with large colors is at most $\epsilon 2^n + t2^{n-m} \leq 2\epsilon 2^n$. The probability that a subcube is monochromatic for a large color is at most the probability that the subcube lies completely within this set of vertices, which is at most the probability that any given vertex in the subcube is in this set. Thus, the probability that a subcube is monochromatic for a large color is at most 2ϵ.

Second, we handle the small colors. Each small color has at most 2^{n-m+1} vertices. By a generalization of the edge-isoperimetric inequality, the set of vertices of size 2^{n-m+1} with the most monochromatic subcubes of dimension ℓ corresponds to a subcube of dimension $n-m+1$ [7, 6]. This larger subcube contains $\binom{n-m+1}{\ell}2^{n-m+1-\ell}$ subcubes of dimension ℓ. Since there are at most 2^m small colors, the total number of monochromatic subcubes with small colors is at most $2^{n+1-\ell}\binom{n-m+1}{\ell}$. Since there are $2^{n-\ell}\binom{n}{\ell}$ subcubes total, the probability of a randomly chosen subcube being monochromatic for a small color is at most $2\frac{\binom{n-m+1}{l}}{\binom{n}{l}}$.

Thus, the probability of a randomly chosen subcube being not monochromatic is at least $1 - 2\frac{\binom{n-m+1}{l}}{\binom{n}{l}} - 2\epsilon$, which means that the average influence is at least this much. ∎

Note that due to the tightness of the isoperimetric bounds, this bound is essentially the best that can be achieved using an averaging argument. Using Lemmas 6.6 and 6.2, we're able to prove the following theorem. Note that the theorem says that if $m > n/\ell$, then we can't even extract with error a small constant.

THEOREM 6.7. *No function $f : \{0,1\}^n \to \{0,1\}^m$ is an ϵ-extractor for $(n, n-\ell)$ nonoblivious bit-fixing sources for any $\epsilon \leq \frac{1}{10}\min\{\frac{\ell\cdot(m-1)}{n}, 1\}$.*

Proof. Suppose f is an ϵ-extractor. First note that f must be within ϵ of uniform on uniform input. So by Lemma 6.6, there is a set of variables S of size ℓ with

$$\begin{aligned} I_f(S) &\geq 1 - 2\frac{\binom{n-m+1}{l}}{\binom{n}{l}} - 2\epsilon \\ &\geq 1 - 2\left(1 - \frac{m-1}{n}\right)^{\ell} - 2\epsilon \\ &\geq 1 - e^{-\ell\cdot(m-1)/n} - 2\epsilon. \end{aligned}$$

By Lemma 6.2, there is an $(n, n-\ell)$ nonoblivious bit-fixing source X so that $f(X)$ is of distance at least $I_f(S)/4$ from uniform, so $\epsilon > I_f(S)/4$. Thus $\epsilon > (1 - e^{-\ell\cdot(m-1)/n})/6$. If $\ell \cdot (m-1)/n \geq 1$, then $\epsilon > (1-e^{-1})/6 > 1/10$. If $\ell \cdot (m-1)/n < 1$, then $e^{-\ell\cdot(m-1)/n} < 1 - (1-e^{-1})\frac{\ell\cdot(m-1)}{n}$, so $\epsilon > \frac{(1-e^{-1})}{6}\frac{\ell\cdot(m-1)}{n} > \frac{1}{10}\frac{\ell\cdot(m-1)}{n}$. ∎

7. Open questions. There remains some work to be done in order to get truly optimal deterministic extractors for oblivious bit-fixing sources. Though we can get nearly optimal results for the d-ary case, for $d > 2$, we lose a factor of δ in the binary case because of the need to take the random walks on the cycle. Ideally, we would like to improve the output length from $\Omega(\delta^2 n)$ to $\Omega(\delta n)$, to match the number of random bits in the input. The extractor of [18] is able to extract almost all of the randomness; however, the error is not as good. In particular, their extractor is not useful for the application to exposure-resilient cryptography. Can we construct an extractor that extracts a linear fraction of the randomness and has small error?

For nonoblivious bit-fixing sources, there also remains more work to be done. It would be nice to eliminate some of the difference between the lower and upper bounds. For the single bit case, Kahn, Kalai, and Linial [21] give a lower bound that improves upon the edge isoperimetric bound by a factor of $\log n$ using a harmonic analysis argument. Perhaps similar techniques could be applied to the general case of many output bits. Also, we could get better extractors if we could modify the construction of Ajtai and Linial [2] to work for smaller error and make it explicit.

Another interesting future direction would be to identify additional classes of sources that have deterministic extractors. One interesting possibility is the set of affine sources, where k bits are chosen uniformly at random and the n bits of the source are affine combinations of these bits. Affine sources are a special case of nonoblivious bit-fixing sources, so our constructions apply to affine sources as well. Other methods allow us to extract when $k > n/2$, but it would be interesting to construct extractors for affine sources that work for $k \leq n/2$. Recently, Bourgain [8] has overcome this barrier by constructing extractors that work for affine sources with $k = \delta n$ for any constant δ. However, there is still room for improvement,

since probabilistic arguments show that affine source extractors exist even when k is logarithmic in n.

Another interesting model is sources generated using a small amount of space. Recently, in joint work with Rao and Vadhan [22], we have given the first explicit constructions of deterministic extractors for such sources.

8. Acknowledgments. We thank Peter Bro Miltersen for suggesting the problem of extractors for oblivious bit-fixing sources and Anindya Patthak and Vladimir Trifonov for helpful discussions.

REFERENCES

[1] M. Ajtai, J. Komlós, and E. Szemerédi, *Deterministic simulation in Logspace*, in the 19th ACM Symposium on Theory of Computing, New York, NY, 1987, pp. 132–140.

[2] M. Ajtai and N. Linial, *The influence of large coalitions*, Combinatorica, 13 (1993), pp. 129–145.

[3] M. Ben-Or and N. Linial, *Collective coin flipping*, in Randomness and Computation, S. Micali, ed., Academic Press, New York, 1990, pp. 91–115.

[4] J Bierbrauer and H. Schellwat, *Almost independent and weakly biased arrays: Efficient constructions and cryptologic applications*, in Advances in Cryptology—CRYPTO 2000, Lecture Notes in Comput. Sci. 1880, Springer-Verlag, Berlin, 2000, pp. 531–543.

[5] M. Blaze, *High-bandwidth encryption with low-bandwidth smartcards*, in Fast Software Encryption, Third International Workshop, Cambridge, UK, Lecture Notes in Comput. Sci 1039, Springer-Verlag, Berlin, 1996, pp. 33–40.

[6] B. Bollobás and I. Leader, *Exact edge-isoperimetric inequalities*, European J. Combin., 11 (1990), pp. 335–340.

[7] B. Bollobás and A. J. Radcliffe, *Isoperimetric inequalities for faces of the cube and the grid*, European J. Combin., 11 (1990), pp. 323–333.

[8] J. Bourgain, *On the Construction of Affine Extractors*, Geom. Funct. Anal., to appear.

[9] V. Boyko, *On the security properties of the oaep as an all-or-nothing transform*, in Advances in Cryptology—CRYPTO 1999, M. Wiener, ed., Lecture Notes in Comput. Sci. 1666, Springer-Verlag, Berlin, 1999, pp. 503–518.

[10] R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, and A. Sahai, *Exposure-resilient functions and all-or-nothing transforms*, in Advances in Cryptology—EUROCRYPT 2000, B. Preneel, ed., Lecture Notes in Comput. Sci. 1807, Springer-Verlag, Berlin, 2000, pp. 453–469.

[11] B. Chor, J. Friedman, O. Goldreich, J. Håstad, S. Rudich, and R. Smolensky, *The bit extraction problem or t–resilient functions*, in 26th Annual Symposium on Foundations of Computer Science, Portland, OR, 1985, pp. 396–407.

[12] A. Cohen and A. Wigderson, *Dispersers, deterministic amplification, and weak random sources*, in 30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, NC, 1989, pp. 14–19.

[13] P. Diaconis, *Group Representations in Probability and Statistics*, Lecture Notes—Monograph Series 11, Institute of Mathematical Statistics, Hayward, CA, 1988.

[14] Y. Dodis, *Exposure-Resilient Cryptography*, Ph.D. thesis, MIT, Cambridge, MA, 2000.

[15] Y. Dodis, A. Sahai, and A. Smith, *On perfect and adaptive security in exposure-resilient cryptography*, in Advances in Cryptology—EUROCRYPT 2001, Birgit Pfitzmann, ed., Lecture Notes in Computer Sci. 2045, Springer-Verlag, 2001, pp. 301–324.

[16] J. Friedman, *On the bit extraction problem*, in 33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, PA, 1992, pp. 314–319.

[17] O. Gabber and Z. Galil, *Explicit construction of linear sized superconcentrators*, J. Comput. System Sci., 22 (1981), pp. 407–420.

[18] A. Gabizon, R. Raz, and R. Shaltiel, *Deterministic extractors for bit-fixing sources by obtaining an independent seed*, in 45th Annual Symposium on Foundations of Computer Science, Rome, Italy, 2004, pp. 394–403.

[19] R. Impagliazzo and D. Zuckerman, *How to recycle random bits*, in the 30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, NC, 1989, pp. 248–253.

[20] M. Jakobsson, J. P. Stern, and M. Yung, *Scramble all, encrypt small*, Lecture Notes in Comput. Sci. 1636 (1999), pp. 95–111.

[21] J. Kahn, G. Kalai, and N. Linial, *The influence of variables on Boolean functions*, in the 29th Annual Symposium on Foundations of Computer Science, White Plains, NY, 1988,

pp. 68–80.

[22] J. Kamp, A. Rao, S. Vadhan, and D. Zuckerman, *Deterministic extractors for small space sources*, in the 38th ACM Symposium on Theory of Computing, Seattle, WA, 2006, pp. 691–700.

[23] K. Kurosawa, T. Johansson, and D. R. Stinson, *Almost k-wise independent sample spaces and their cryptologic applications*, J. Cryptology, 14 (2001), pp. 231–253.

[24] L. Lovász, *Random walks on graphs: A survey*, in Combinatorics, Paul Erdős is Eighty, Vol. 2, D. Miklós, V. T. Sós, and T. Szőnyi, eds., János Bolyai Math. Soc., Budapest, 1996, pp. 353–398.

[25] A. Lubotzky, *Discrete Groups, Expanding Graphs and Invariant Measures*, Birkhäuser-Verlag, Basel, Switzerland, 1994.

[26] A. Lubotzky, R. Philips, and P. Sarnak, *Ramanujan graphs*, Combinatorica, 8 (1988), pp. 261–277.

[27] S. Matyas, M. Peyravian, and A. Roginsky, *Encryption of long blocks using a short-block encryption procedure*. http://grouper.ieee.org/groups/1363/P1363a/LongBlock.html.

[28] N. Nisan and D. Zuckerman, *Randomness is linear in space*, J. Comput. System Sci., 52 (1996), pp. 43–52.

[29] O. Reingold, S. Vadhan, and A. Wigderson, *Entropy waves, the zig-zag product, and new constant-degree expanders and extractors*, Ann. of Math. (2), 155 (2002), pp. 155–187.

[30] R. L. Rivest, *All-or-nothing encryption and the package transform*, Lecture Notes in Comput. Sci., 1267 (1997), pp. 210–218.

[31] A. Russell and D. Zuckerman, *Perfect-information leader election in* $\log^* n + O(1)$ *rounds*, J. Comput. System Sci., 63 (2001), pp. 612–626.

[32] M. Santha and U. V. Vazirani, *Generating quasi-random sequences from semi-random sources*, J. Comput. System Sci., 33 (1986), pp. 75–87.

[33] R. Shaltiel, *Recent developments in explicit constructions of extractors*, Bull. Eu. Assoc. Theor. Comput. Sci., (2002), pp. 67–95.

[34] L. Trevisan and S. P. Vadhan, *Extracting randomness from samplable distributions*, in the 41st Annual Symposium on Foundations of Computer Science, Redondo Beach, CA, IEEE Comput. Soc. Press, Los Alamitos, CA, 2000, pp. 32–42.

SIAM J. COMPUT.
Vol. 36, No. 5, pp. 1248–1263

LINEAR UPPER BOUNDS FOR RANDOM WALK ON SMALL DENSITY RANDOM 3-CNFs*

MIKHAIL ALEKHNOVICH† AND ELI BEN-SASSON‡

In memory of Mikhail (Misha) Alekhnovich—friend, colleague and brilliant mind

Abstract. We analyze the efficiency of the random walk algorithm on random 3-CNF instances and prove *linear* upper bounds on the running time of this algorithm for small clause density, less than 1.63. This is the first subexponential upper bound on the running time of a *local improvement* algorithm on random instances. Our proof introduces a simple, yet powerful tool for analyzing such algorithms, which may be of further use. This object, called a *terminator*, is a weighted satisfying assignment. We show that any CNF having a good (small weight) terminator is assured to be solved quickly by the random walk algorithm. This raises the natural question of the *terminator threshold* which is the maximal clause density for which such assignments exist (with high probability). We use the analysis of the pure literal heuristic presented by Broder, Frieze, and Upfal [*Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1993, pp. 322–330] and Luby, Mitzenmacher, and Shokrollahi [*Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1998, pp. 364–373] and show that for small clause densities good terminators exist. Thus we show that the pure literal threshold ($\approx$1.63) is a lower bound on the terminator threshold. (We conjecture the terminator threshold to be in fact higher.) One nice property of terminators is that they can be found efficiently via linear programming. This makes tractable the future investigation of the terminator threshold and also provides an efficiently computable certificate for short running time of the simple random walk heuristic.

Key words. SAT solving, random CNF, SAT heuristics, random walk algorithm

AMS subject classifications. 68Q25, 68W20, 68W40

DOI. 10.1137/S0097539704440107

1. Introduction.

The phenomena we seek to explain is best described by Figure 1.

`RWalkSAT`, originally introduced by Papadimitriou [35], tries to find a satisfying assignment for a CNF $\mathcal{C}$ by the following method. We start with a random assignment, and as long as the assignment at hand does not satisfy the CNF, an unsatisfied clause $C \in \mathcal{C}$ is picked, and the assignment to a random literal in this clause is flipped. The new assignment satisfies C but may "ruin" the satisfiability of other clauses. We repeat this process (of flipping a bit in the current assignment according to some unsatisfied clause) until either a satisfying assignment is found (success) or we get tired and give up (failure).

The lower batch in Figure 1 (plus sign) was obtained by selecting 810 random 3-CNF formulas[1] with a clause density (i.e., clause/variable ratio) of 1.6 and running `RWalkSAT` on each instance. The y-axis records the number of assignments used before finding a satisfying one. In particular, the algorithm found an assignment in all

*Received by the editors January 27, 2004; accepted for publication (in revised form) March 24, 2006; published electronically December 21, 2006.
http://www.siam.org/journals/sicomp/36-5/44010.html

†The author is deceased. Former address: Department of Mathematics, University of California, San Diego, La Jolla, CA 92093-0112. This work was done while the author was a graduate student at MIT. This author was supported in part by NSF award CCR 0205390 and MIT NTT award 2001-04.

‡Department of Computer Science, Technion-Israel Institute of Technology, Technion City, Haifa 32000, Israel (eli@cs.technion.ac.il). This work was done while the author was a Postdoctoral Fellow at MIT and Harvard University. This author was supported by NSF grants CCR-0133096 and CCR-9877049, NSF award CCR 0205390, and NTT award MIT 2001-04.

[1]Ten formulas per $n = 2000, 2050, 2100, \ldots, 6000$ were selected.

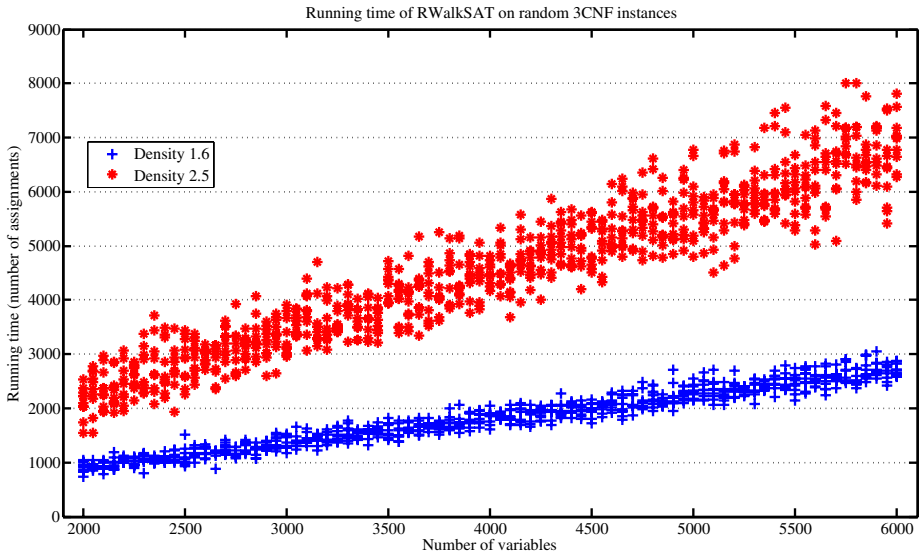


FIG. 1. *Running time of* `RWalkSAT` *on random* 3*-CNF instances with clause densities* 1.6 *and* 2.5.

instances. The upper batch (star sign) was similarly obtained by running `RWalkSAT` on 810 random 3-CNF instances with a higher clause density of 2.5.

Figure 1 raises the conjecture that for clause density 1.6 the running time is linear. Actually, it is even less than the number of variables (and clauses) and seems to have a slope of $\approx 1/2$. In this paper we offer an explanation for the seemingly linear running time of Figure 1. We prove that random 3-CNFs with clause density less than 1.63 take (with high probability) a *linear* number of `RWalkSAT` steps. (We leave the explanation of the running time displayed in the upper batch of Figure 1 as an interesting open problem.)

1.1. Techniques: Terminators. Our technique can be viewed as a generalization of the analysis of `RWalkSAT` on satisfiable 2-CNF formulas [35], so we briefly review this result. Papadimitriou showed that the Hamming distance of the assignment at time t from some fixed satisfying assignment α is a random variable that decreases at each step with probability at least $1/2$. Thus, in at most $O(n^2)$ steps this random variable will reach 0, implying we have found α. (The algorithm may succeed even earlier by finding some other satisfying assignment.)

We look at *weighted satisfying assignments*; i.e., we give nonnegative weights to the bits of α. Instead of Hamming distance, we measure the *weighted distance* between α and the current assignment α^t. We show that in some cases, one can find a satisfying assignment α and a set of weights w such that for *any* unsatisfied clause at time t, the expectation of the weighted distance (between α and α^t) decreases by at least 1. Moreover, the maximal weight given to any variable is constant. In this case the running time of `RWalkSAT` will be linear with high probability (even better than the quadratic upper bound of [35] for 2-CNFs). We call such weighted assignments *terminators*, as their existence assures us that `RWalkSAT` will terminate successfully in linear time.

Two parameters of a terminator bound the running time of `RWalkSAT`. The *total weight* (sum of weights of all variables) bounds the distance needed to be traversed by the random walk, because the weighted distance of α^0 from α can be as large as this

sum. The second parameter is the maximal weight of a variable, which bounds the variance of our random walk. Thus we define the termination weight of $\mathcal{C}$ (denoted $\mathrm{Term}(\mathcal{C})$) to be the minimal product of these two parameters, taken over all terminators for $\mathcal{C}$. As stated above, the running time of `RWalkSAT` is linear (at most) in the termination weight of $\mathcal{C}$. Not all satisfiable CNFs have these magical terminators, and if $\mathcal{C}$ has no terminator, we define its termination weight to be ∞.

1.2. Results. With the terminator concept in hand, we examine the running time of `RWalkSAT` on random 3-CNF formulas. If $\mathcal{C}$ is a random 3-CNF, then $\mathrm{Term}(\mathcal{C})$ is a random variable. Understanding this variable and bounding it from above bounds the running time of `RWalkSAT` . Our main result (Theorem 4.1) is that for clause density ≤ 1.63, a random 3-CNF has *linear* termination weight (hence `RWalkSAT` succeeds in linear time). This matches the behavior depicted in Figure 1 up to a multiplicative constant. We also present a determinisitic version of `RWalkSAT` and show it finds a satisfying assignment in linear time for the same clause density (section 3.1).

Our result relies on previous analysis done for bounding a different SAT heuristic, called the *pure literal heuristic* [10] (see also [31] for a different and shorter analysis). This heuristic is known to succeed up to a clause density threshold of 1.63 and fails above this density. We conjecture terminators should exist even beyond the pure literal threshold, as (unreported) experimental data seems to indicate. However, at clause density ≥ 2 only a negligible fraction of random CNFs has terminators (see section 5), meaning we need to develop new techniques for explaining the observed linear running time at (say) density 2.5 depicted in (the upper part of) Figure 1.

A terminator is a solution to a linear system of inequalities, and thus linear programming can be used to find it. Thus, the existence of a terminator for random $\mathcal{C}$ can be decided efficiently, and an upper bound on $\mathrm{Term}(\mathcal{C})$ can be computed efficiently. (However, obtaining the exact value of $\mathrm{Term}(\mathcal{C})$ is not known to be efficiently computable.) This may allow us to gain a better empirical understanding of the behavior of `RWalkSAT` and its connection to the termination weight parameter.

The success of the pure literal heuristic does not necessarily imply polynomial running time for `RWalkSAT`. Indeed, in section 6 we provide a counterexample that requires exponential time from `RWalkSAT`, although a solution can be found using the pure literal heuristic in linear time. Furthermore, for a random planted SAT instance with large enough clause density, `RWalkSAT` takes exponential time (section 7). This is in contrast to the efficient performance of spectral algorithms for planted SAT presented by Flaxman [18].

1.3. History and related results.

Local improvement algorithms. `RWalkSAT` was introduced by Papadimitriou, who showed it has quadratic running time on satisfiable 2-CNFs [35]. An elegant upper bound was given by Schoning, who showed that the expected running time of `RWalkSAT` on any k-CNF is at most $(1+1/k)^n$ (compared with the exhaustive search upper bound of 2^n) [38]. The (worst case) upper bound of [38] was improved in a sequence of results [15, 21, 8, 22, 37], and the best upper bound for 3-SAT is $(1.324)^n$, given by the recent paper [22].

`RWalkSAT` is one of a broad family of *local improvement* algorithms, (re)introduced in the 1990s with the work of [41]. Algorithms in this family start with an assignment to the input formula, and gradually change it one bit at a time, by trying to locally optimize a certain function. These algorithms (the most famous of which is WalkSAT) are close relatives of the simulated annealing method and were found to compete with DLL-type algorithms (also known as Davis–Putnam algorithms). Empirical

results on random 3CNFs with up to $100,000$ variables seem to indicate that `RWalkSAT` terminates successfully in linear time up to clause density ≤ 2.6 [36, 40]. More advanced algorithms such as WalkSAT (a Metropolis algorithm that is related to `RWalkSAT`) appear empirically to solve random 3CNF instances with clause density ≤ 4 in quadratic time, and there is data indicating polynomial running time up to density ≤ 4.2 (the empirical SAT threshold is ≈ 4.26) [39].

Random 3*-CNFs.* Random CNFs have received much interest in recent years, being a natural distribution on NP-complete instances that seems (empirically as well as theoretically) computationally hard for a wide range of parameters. This distribution is investigated in such diverse fields as physics [30, 32], combinatorics [24], proof complexity [13], algorithm analysis [3], and hardness of approximation [17], to mention just a few. One of the basic properties of random 3-CNFs is that for small density ($\Delta < 3.52\ldots$ (see [20, 29])) almost all formulas are satisfiable, whereas for large density ($\Delta > 4.506\ldots$ (see [16])) they are almost all unsatisfiable. Another interesting property is that the threshold between satisfiability and unsatisfiability is sharp [24]. It is conjectured that a *threshold constant* exists, and empirical experiments estimate it to be ≈ 4.26 [14]. The analysis of SAT solving algorithms on random CNFs has been extensively researched empirically, and random CNFs are commonly used as test cases for analysis and comparison of SAT solvers. From a theoretical point of view, several upper bounds were given on the running time of DPLL-type algorithms of increasing sophistication [1, 2, 3, 10, 31, 11, 12, 19, 20, 29]. The best rigorous upper bound for random 3-CNFs is given by the recent papers [20, 29]. An exponential lower bound on a wide class of DPLL algorithms for density ≈ 3.8 and above was given by [3]. Recently, Mézard et al. presented the survey propagation algorithm and showed that nonrigorous arguments based on replica symmetry and experimental results indicate it efficiently solves large random 3CNF instances very close to the empirical satisfiability [32, 33].

Upper bounds for algorithms imply lower bounds on the satisfiability threshold, and in fact, all lower bounds on the threshold (for $k = 3$) so far have come from analyzing specific SAT solving algorithms. Most of the algorithms for which average case analysis has been applied so far are DPLL algorithms (and typically, with the exception of the recent papers [20, 29], when proving upper bounds on these algorithms, myopic[2] versions are considered). Much less is known about non-DPLL algorithms, in particular local improvement ones. Our result is (to the best of our knowledge) the first rigorous theoretical analysis of a non-DPLL algorithm on random CNFs.

Paper outline. After giving the necessary formal definition in section 2, we discuss terminators in section 3. Using terminators we prove our upper bound in section 4. In section 5 we give some theoretical upper bounds on the terminator threshold. We then discuss the tightness of the terminator method (section 6). We conclude with exponential lower bounds on the running time of `RWalkSAT` on random CNFs from the "planted-SAT" distribution (section 7).

2. Preliminaries.

Random 3*-CNFs.* For x_i a Boolean variable, a *literal* ℓ_i over x_i is either x_i or $\bar{x_i}$ (the negation of x_i), where x_i is called a *positive literal* and $\bar{x_i}$ is a *negative* one. A *clause* is a disjunction of literals, and a CNF formula is a set of clauses. Throughout this paper we reserve calligraphic notation for CNF formulas. For $\mathcal{C}$ a CNF, let $Vars(\mathcal{C})$ denote the set of variable appearing in $\mathcal{C}$ (we will always assume $Vars(\mathcal{C}) =$

[2] See [3] for the definition and tightest analysis of myopic algorithms.

$\{x_1, \ldots, x_n\}$ for some n). An *assignment* to $\mathcal{C}$ is some Boolean vector $\alpha \in \{0,1\}^n$. A literal ℓ_i is satisfied by α iff $\ell_i(\alpha_i) = 1$. We study the following distribution.

DEFINITION 2.1. *Let $\mathbb{F}^n_\Delta$ be the probability distribution obtained by selecting Δn clauses uniformly at random from the set of all $8 \cdot \binom{n}{3}$ clauses of size 3 over n variables. $\mathcal{C} \sim \mathbb{F}^n_\Delta$ means that $\mathcal{C}$ is selected at random from this distribution. We call such a $\mathcal{C}$ a* random 3-CNF

The algorithm. `RWalkSAT` is described by the following pseudocode. $\mathcal{C}$ is the input CNF and T is the time bound; i.e., if no satisfying assignment is found in T steps, we give up. We use the notation $UNSAT(\mathcal{C}, \alpha)$ for the set of clauses of $\mathcal{C}$ that are unsatisfied by α.

```
RWalkSAT(C, T)
  Select α ∈ {0,1}^n (uniformly) at random
  Initialize t = 0
  While t < T  {
    If C(α) = 1 Return ("INPUT SATISFIED BY" α)
    Else {
      Select C ∈ UNSAT(C, α) at random
      Select literal ℓ ∈ C at random
      Flip assignment of α at ℓ
      t++ }
    }
  Return "FAILED TO FIND SATISFYING ASSIGNMENT IN T STEPS"
```

Martingales and Azuma's inequality. Below we state Azuma's inequality for martingales. We refer the reader to [34] for the definition of conditional expectation and for more information about martingales.

A *martingale* is a sequence $X_0, X_1, X_2, \ldots, X_m$ of random variables such that for $0 \leq i < m$ holds

$$\mathbf{E}[X_{i+1} | X_i] = X_i.$$

The following version of Azuma's inequality [7, 27] may be found in [6].

THEOREM 2.2 (Azuma's inequality). *Let $0 = X_0, \ldots, X_m$ be a martingale with $|X_{i+1} - X_i| \leq 1$ for all $0 \leq i < m$. Let $\lambda > 0$ be arbitrary. Then*

$$\Pr[X_m > \lambda\sqrt{m}] < e^{-\lambda^2/2}.$$

3. Terminators. In this section we develop the tools needed to bound the running time of `RWalkSAT` on various interesting instances.

Intuition. Suppose a k-CNF $\mathcal{C}$ over n variables has a satisfying assignment α such that each clause of $\mathcal{C}$ is satisfied by at least $k/2$ literals under α. In this case `RWalkSAT` will terminate in quadratic time (with high probability). The reason is that if a clause C is unsatisfied at time t by α^t, then α^t must disagree with α on at least half of the literals in C. So with probability $\geq 1/2$ we decrease the Hamming distance between our current assignment and α. If we let sim^t be the similarity of α^t and α, i.e., the number of bits that are identical in both assignments (notice $0 \leq \mathrm{sim}^t \leq n$), then sim^t is a submartingale, i.e., $E(\mathrm{sim}^t | \mathrm{sim}^1, \ldots, \mathrm{sim}^{t-1}) \geq \mathrm{sim}^{t-1}$. Standard techniques from the theory of martingales show that sim reaches n (so α^t reaches α) within $O(n^2)$ steps. One elegant example of this situation is when $\mathcal{C}$ is a satisfiable 2-CNF. Papadimitriou [35] proved quadratic upper bounds on the running time of `RWalkSAT` in this case, using the proof method outlined above.

For a general 3-CNF we do not expect a satisfying assignment to have two satisfying literals per clause. Yet all we need in order to prove good running time is to set up a measure of similarity between α^t and some fixed satisfying assignment α such that (i) if sim^t reaches its maximal possible value, then $\alpha^t = \alpha$; and (ii) the random variables $\mathrm{sim}^1, \mathrm{sim}^2, \ldots$ are a submartingale. We achieve both these properties by giving *nonnegative weights* $w_1, \ldots, w_n$ to the variables $x_1, \ldots, x_n$. Instead of similarity, we measure the *weighted similarity* between α and α^t, defined by $\mathrm{sim}_w(\alpha, \alpha^t) \stackrel{\mathrm{def}}{=} \sum_{\alpha_i^t = \alpha_i} w_i$. Now suppose there exists a satisfying assignment α such that for any clause C, the expected change in sim_w, conditional on C being unsatisfied, is nonnegative. Suppose, furthermore, that all w_i are bounded by a constant and every clause has a variable with nonzero weight bounded below by another constant. Then we may conclude as above that α^t will reach its maximal value $W = \sum_i w_i$ in time $O(W^2)$.

In some cases we can do even better. We set up a system of weights such that (for any clause C) the expected change in sim_w (conditional on C being unsatisfied) is *strictly positive*. In this case the running time is *linear* in $W = \sum w_i$ (instead of quadratic). As we shall later see, such a setting of weights is possible (with high probability) for random 3-CNFs. But first we formalize our intuition.

Notation. In what follows Boolean variables range over $\{-1, 1\}$. A CNF $\mathcal{C}$ with n variables and m clauses is represented by an $m \times n$ matrix $A^{\mathcal{C}}$ with $\{-1, 0, 1\}$-entries. The ith clause is represented by $A_i^{\mathcal{C}}$ (the ith row of $A^{\mathcal{C}}$) and has a -1-entry in the jth position if $\bar{x_j}$ is a literal of the ith clause of $\mathcal{C}$, a 1-entry if x_j is a literal of C_i, and is zero otherwise. Thus, if $\mathcal{C}$ is a k-CNF, then the support size of each row $A_i^{\mathcal{C}}$ is at most k. A Boolean assignment is $\alpha \in \{-1, 1\}^n$, and we say α satisfies $\mathcal{C}$ iff for all $i \in [m]$

$$\langle A_i^{\mathcal{C}}, \alpha \rangle > -\left\| A_i^{\mathcal{C}} \right\|_1, \tag{1}$$

where $\langle \alpha, \beta \rangle$ is the standard inner product over $\mathbb{R}^n$ (defined by $\sum_{i=1}^n \alpha_i \cdot \beta_i$) and $\|\cdot\|_1$ is the ℓ_1 norm (defined by $\|\beta\|_1 = \sum_{i=1}^n |\beta_i|$). It is easy to see that this definition of satisfiability coincides with the standard one.

Terminator: Definition. A terminator is a generalization of a satisfying assignment. On the one hand, we allow α to be any vector in $\mathbb{R}^n$, but we require a stronger satisfying condition than (1).

DEFINITION 3.1 (terminators). *Let $\mathcal{C}$ be a k-CNF with n variables and m clauses represented by the matrix $A^{\mathcal{C}}$. $\alpha \in \mathbb{R}^n$ is a* terminating *satisfying assignment* (*or* terminator) *if for all $i \in [m]$*

$$\langle A_i^{\mathcal{C}}, \alpha \rangle \geq 1. \tag{2}$$

The termination weight of $\mathcal{C}$ is

$$\mathrm{Term}(\mathcal{C}) \stackrel{\mathrm{def}}{=} \min\{\|\alpha\|_1 \cdot \|\alpha\|_\infty : \alpha \text{ terminator for } \mathcal{C}\}.$$

In case $\mathcal{C}$ has no terminator, we define $\mathrm{Term}(\mathcal{C})$ *to be ∞.*

One may think of $\mathrm{sign}(\alpha_i)$ as the Boolean assignment to variable x_i (where $\mathrm{sign}(\alpha_i)$ is 1 if $\alpha_i \geq 0$ and is -1 otherwise) and $|\alpha_i|$ as the weight given to x_i. Notice that if α is a terminator, then the $\{-1, 1\}$-vector $\mathrm{sign}(\alpha)$ satisfies $\mathcal{C}$. This is because by property (2) in each clause there is at least one literal that agrees in sign with α.

The decisive name given in the previous definition is justified by the following claim, which is the main theorem of this section.

THEOREM 3.2 (terminator theorem). *If a k-CNF $\mathcal{C}$ has a terminator α, then* `RWalkSAT` *succeeds on $\mathcal{C}$ in time $O(\|\alpha\|_1 \cdot \|\alpha\|_\infty)$ with probability $\geq 1 - \exp(-\Omega(\|\alpha\|_1 / \|\alpha\|_\infty))$.*

Notice that we do not claim that when `RWalkSAT` terminates, it finds the assignment $\text{sign}(\alpha)$, but rather the existence of any terminator of small weight implies short running time. We can say that `RWalkSAT` is "drawn to" α but only when using the weighted distance measure given by $|\alpha|$. If $|\alpha_i| = 1$, this means `RWalkSAT` indeed approaches α (as is the case when each clause is satisfied by two literals). But in general, being "close" according to the weighted measure $|\alpha|$ does not imply small Hamming distance.

Proof of Theorem 3.2. Let $\mathcal{C}$ be a k-CNF and α be a terminator of minimal weight for $\mathcal{C}$, i.e., $\text{Term}(\mathcal{C}) = \|\alpha\|_1 \cdot \|\alpha\|_\infty < \infty$. Let $\beta^t \in \{-1, 1\}^n$ be the sequence of assignments traversed by `RWalkSAT`$(\mathcal{C})$ starting from the random assignment β^1, where $t \leq T = c \cdot k \cdot \|\alpha\|_1 \cdot \|\alpha\|_\infty$ (c will be fixed later). For $t \geq 1$ let Y^t be the random variable $\langle \beta^t, \alpha \rangle$. If `RWalkSAT` fails to find a satisfying assignment in T steps, then the following event occurs:

$$Y^t < \|\alpha\|_1 \quad \text{for all } t < T. \tag{3}$$

This is because $\langle \beta^t, \alpha \rangle = \|\alpha\|_1$ implies $\beta^t = \text{sign}(\alpha)$ and $\text{sign}(\alpha)$ satisfies $\mathcal{C}$. Thus we need only to bound the probability of event (3). Suppose clause C_i is picked at time t (i.e., C_i is unsatisfied by β^{t-1}). We claim the expected change in Y^t (with respect to Y^{t-1}) is precisely

$$\frac{2}{k} \cdot \langle A_i^{\mathcal{C}}, \alpha \rangle. \tag{4}$$

With probability $1/k$ we flip the assignment to each literal x_j of C_i, which amounts to multiplying β_j^{t-1} by -1. Thus the expected change in Y^t is $\frac{-2}{k} \cdot \langle \beta^{t-1}|_i, \alpha \rangle$, where $\beta^{t-1}|_i$ is the restriction of β^{t-1} to support of $A_i^{\mathcal{C}}$. But C_i being unsatisfied by β^{t-1} implies $\beta^{t-1}|_i = -A_i^{\mathcal{C}}$, so (4) is proved. Thus by property (2) in Definition 3.1

$$E[Y^t | Y^1, \ldots, Y^{t-1}] = Y^{t-1} + \frac{2}{k} \langle A_i^{\mathcal{C}}, \alpha \rangle \geq Y^{t-1} + \frac{1}{k}.$$

We claim that the sequence of random variables

$$X_t \stackrel{\text{def}}{=} \sum_{\ell=1}^{t} \left(Y^\ell - \mathbf{E}[Y^\ell | Y^1, \ldots, Y^{\ell-1}] \right)$$

is a martingale satisfying $\mathbf{E}X_1 = 0$. Indeed,

$$\begin{aligned}
\mathbf{E}[X_t | X_1, \ldots, X_{t-1}] &= \mathbf{E}[X_t | Y^1, \ldots, Y^{t-1}] \\
&= \mathbf{E}\left[\sum_{\ell=1}^{t} \left(Y^\ell - \mathbf{E}[Y^\ell | Y^1, \ldots, Y^{\ell-1}] \right) | Y^1, \ldots, Y^{t-1} \right] \\
&= \mathbf{E}\left[Y^t | Y^1, \ldots, Y^{t-1} \right] - \mathbf{E}\left[\mathbf{E}[Y^t | Y^1, \ldots, Y^{t-1}] | Y^1, \ldots, Y^{t-1} \right]
\end{aligned}$$

$$+\mathbf{E}\left[\sum_{\ell=1}^{t-1}\left(Y^{\ell}-\mathbf{E}[Y^{\ell}|Y^{1},\ldots,Y^{\ell-1}]\right)|Y^{1},\ldots,Y^{t-1}\right]$$

$$=0+\mathbf{E}\left[\sum_{\ell=1}^{t-1}\left(Y^{\ell}-\mathbf{E}[Y^{\ell}|Y^{1},\ldots,Y^{\ell-1}]\right)|Y^{1},\ldots,Y^{t-1}\right]$$

$$=\sum_{\ell=1}^{t-1}\left(Y^{\ell}-\mathbf{E}[Y^{\ell}|Y^{1},\ldots,Y^{\ell-1}]\right)=X_{t-1}.$$

Also $E[X_1]=E[Y^1-E[Y^1]]=0$. For all t, $|X_{t+1}-X_t|=Y^{t+1}-\mathbf{E}[Y^{t+1}|Y^1,\ldots,Y^t]\le \|\alpha\|_\infty$. Note that

$$X_t=Y^t-\sum_{\ell=1}^{t}(\mathbf{E}[Y^{\ell}|Y^{1},\ldots,Y^{\ell-1}]-Y^{\ell})-\mathbf{E}Y^1\le Y^t-t/k+\|\alpha\|_1.$$

In order to bound the probability of event (3), it suffices to bound the probability of the event "$X_T<2\|\alpha\|_1-T/k$" (if this event does not occur, then $Y^T\ge -\|\alpha\|_1+X_T\ge\|\alpha\|_1$). Recalling $T=c\cdot k\cdot\|\alpha\|_1\cdot\|\alpha\|_\infty$ we will pick $c>\frac{4}{k}$ so that

$$2\|\alpha\|_1-\frac{T}{k}=2\|\alpha\|_1-c\cdot\|\alpha\|_1\cdot\|\alpha\|_\infty<-\frac{ck}{2}\|\alpha\|_1\cdot\|\alpha\|_\infty.$$

We now apply Azuma's inequality and get

$$\begin{aligned}(3)&\le\Pr\left[X_T<-\frac{ck}{2}\|\alpha\|_1\|\alpha\|_\infty\right]\\&=\Pr\left[\frac{X_T}{\|\alpha\|_\infty}<-\frac{ck}{2}\|\alpha\|_1\right]\\&\le\exp\left(-\frac{(\frac{ck}{2}\|\alpha\|_1)^2}{2T}\right)\\&\le\exp\left(-\frac{c^2k^2(\|\alpha\|_1)^2}{8ck\cdot\|\alpha\|_1\|\alpha\|_\infty}\right)\\&\le\exp\left(-\frac{ck\|\alpha\|_1}{8\|\alpha\|_\infty}\right)=\exp\left(-\Omega\left(\frac{\|\alpha\|_1}{\|\alpha\|_\infty}\right)\right).\end{aligned}$$

The theorem is proved. □

3.1. A deterministic variant of `RWalkSAT`. Consider the following deterministic variant of `RWalkSAT`, which we will call `DWalkSAT`. Fix an ordering on clauses in $\mathcal{C}$. Initialize α_0 to be (say) the all zero assignment. At each step t, select the smallest clause unsatisfied by α_t and flip the assignment to *all* literals in it. Repeat this process until all clauses are satisfied. Naturally, one can introduce a time bound T and declare failure if a satisfying assignment is not found within T steps. We immediately get the following result.

THEOREM 3.3. *If a CNF $\mathcal{C}$ has a terminator α, then* `DWalkSAT` *succeeds on $\mathcal{C}$ within $2\cdot\|\alpha\|_1$ steps.*

Proof. We closely follow the proof of the terminator theorem, Theorem 3.2. Let $\beta^1,\ldots$ be the (deterministic) sequence of assignments traversed by the algorithm.

Let $Y^t = \langle \beta^t, \alpha \rangle$ (noticing Y^t is no longer random). Clearly, $Y^1 \geq -\|\alpha\|_1$, and if $Y^t = \|\alpha\|_1$, then β^t (equals $\text{sign}(\alpha)$, hence) satisfies $\mathcal{C}$. So we have to only show for all t

$$Y^t \geq Y^{t-1} + 2. \tag{5}$$

This follows from the fact that the clause $\mathcal{C}_i$ flipped at time t was unsatisfied at time $t-1$. Flipping all variables in $\mathcal{C}_i$ amounts to adding to Y^{t-1} the amount $\langle A_i^{\mathcal{C}}, \alpha \rangle$, and this, by definition of terminator, is at least one. We have proved (5) and with it the theorem. □

4. Linear upper bounds on random CNFs. In this section we show that for clause densities for which the pure literal heuristic succeeds, there exist linear weight terminators. Our current analysis uses insights into the structure of such pure CNFs, but we see no reason to believe that the terminator threshold is linked to the pure literal threshold. The main theorem of this section is the following.

THEOREM 4.1. *For any $\Delta < 1.63$, there exists a constant c such that with high probability $\mathcal{C} \sim \mathbb{F}_\Delta^n$ has a terminator $\alpha \in \mathbb{R}^n$ with $\|\alpha\|_\infty \leq c$ and hence $\|\alpha\|_1 \leq c \cdot n$.*

COROLLARY 4.2. *For any $\Delta < 1.63, \epsilon > 0$, there exists a constant c such that with high probability for $\mathcal{C} \sim \mathbb{F}_\Delta^n$, `RWalkSAT` succeeds on $\mathcal{C}$ in time $c \cdot n$ with probability $\geq 1 - \epsilon$.*

COROLLARY 4.3. *For any $\Delta < 1.63, \epsilon > 0$, there exists a constant c such that with high probability for $\mathcal{C} \sim \mathbb{F}_\Delta^n$, `DWalkSAT` succeeds on $\mathcal{C}$ in time $c \cdot n$.*

To prove our main theorem, we construct small weight terminators for pure and expanding CNFs and then merge the two into one small weight terminator.

4.1. Terminators for pure CNFs. A literal ℓ in $\mathcal{C}$ is called *pure* if it appears only as a positive literal, or only as a negative literal, in $\mathcal{C}$. A clause in $\mathcal{C}$ is said to be *pure* if it contains a pure literal. When seeking a satisfying assignment, a natural strategy is to start by assigning all pure literals their satisfying assignment and thus remove all pure clauses. The removal of pure clauses may result in the emergence of new pure literals in the restricted CNF, and the process may be repeated. The *pure literal heuristic* is the heuristic that applies this removal process until no pure clauses remain. If the remaining CNF is empty, the pure literal heuristic has found a satisfying assignment, and otherwise it failed.

Let us introduce some notation. For $\mathcal{C}$ a CNF, define $\mathcal{C}_0 = \mathcal{C}$, L_0 to be the set of pure literals in $\mathcal{C}$, and P_0 to be the set of pure clauses in $\mathcal{C}$. Recursively define $\mathcal{C}_{i+1}$ to be $\mathcal{C}_i \setminus P_i$, and let L_{i+1}, P_{i+1} be, respectively, the set of pure literals and pure clauses in $\mathcal{C}_{i+1}$. Finally, let r be the minimal i such that $L_i = \emptyset$. Notice that the pure literal succeeds on $\mathcal{C}$ iff $\mathcal{C}_r = \emptyset$. If $\mathcal{C}_r = \emptyset$, we say $\mathcal{C}$ is *r-pure*.

THEOREM 4.4. *Every r-pure k-CNF over n variables has a terminator $\alpha \in \mathbb{R}^n$ with $\|\alpha\|_\infty \leq k^r$ and $\|\alpha\|_1 \leq n \cdot k^r$, so $\text{Term}(\mathcal{C}) \leq n \cdot k^{2r}$. Moreover, α is supported only on $\cup_{i=0}^{r-1} L_i$.*

Notice that invoking Theorem 3.2 we bound the running time of `RWalkSAT` on an r-pure k-CNF by $n \cdot k^{2r}$ (with high probability).

Proof. Let $L_0, \ldots, L_{r-1}$ be the pure literals in $\mathcal{C}_0, \ldots, \mathcal{C}_{r-1}$. Notice that $\cup_{j=0}^{r-1} L_j$ does not necessarily cover all variables in $\mathcal{C}$, but assigning each pure literal to 1 (i.e., if ℓ_i is pure, then set $\text{sign}(\alpha_i) = \text{sign}(\ell_i)$) and assigning the other variables arbitrarily gives a satisfying assignment α. We now deal with the weights (absolute values) of α. Fix the weight of each variable in L_j to k^{r-j}. For any variable $x_i \notin \cup_{j=0}^{r-1} L_j$ fix its weight to 0.

To see that α is a terminator (of weight nk^r), consider any clause $C_i \in P_j$. By definition of P_j there are no literals from $L_0, \ldots, L_{j-1}$ appearing in C. Thus all literals appearing in C have weight $\leq (k)^{r-j}$. There is at least one literal $\ell_s \in C$ that has weight k^{r-j} and agrees with α_s in sign, and any literal disagreeing with α must have weight $\leq (k)^{r-j-1}$. Hence

$$\langle A_i^{\mathcal{C}}, \alpha \rangle \geq k^{r-j} - (k-1) \cdot k^{r-j-1} \geq 1. \qquad \square$$

Broder, Frieze, and Upfal showed that with high probability the pure literal heuristic finds a satisfying assignment for a random 3-CNF with clause density < 1.63 [10] (for a simpler analysis of the same heuristic see [31]). In particular, the following theorem follows from the work of [10]. A proof of this theorem can be found in Appendix A.

THEOREM 4.5 (see [10]). *For every $\Delta < 1.63$, there exists a constant c such that with high probability $\mathcal{C} \sim \mathbb{F}_\Delta^n$ is $c \log n$-pure.*

By applying Theorems 3.2 and 4.4 to Theorem 4.5 we conclude that the running time of `RWalkSAT` on a random instance (with small enough clause density) is at most polynomial.

4.2. Terminators for expanding CNFs. Our next step in proving Theorem 4.1 starts with the following theorem, which is a combination of a result of Broder, Frieze, and Upfal [10] and (the now) standard analysis of random CNFs, originating in the work of Chvátal and Szemerédi [13]. Being standard and somewhat technical, we defer its proof to Appendix A.

DEFINITION 4.6. *For $\mathcal{C}$ a CNF, we say $\mathcal{C}$ is an* (r, c)-expander *if for all $\mathcal{C}' \subseteq \mathcal{C}$ $|\mathcal{C}'| \leq r$, $|Vars(\mathcal{C}')| \geq c \cdot |\mathcal{C}'|$.*

THEOREM 4.7. *For every $\Delta < 1.63$, there exists an integer d such that for $\mathcal{C} \sim \mathbb{F}_\Delta^n$, with high probability $\mathcal{C}_d$ is a $(|\mathcal{C}_d|, 7/4)$-expander, where $\mathcal{C}_d$ is the CNF remaining of $\mathcal{C}$ after removing the d outermost pure layers.*

This theorem assures us that after removing a *constant* number of the layers from a random $\mathcal{C}$ (with small clause density), we have in hand a residual CNF $\mathcal{C}_d$, such that *any* subset of it, including all of $\mathcal{C}_d$, has a very large set of neighbors. This in turn implies the existence of small weight terminators for $\mathcal{C}_d$.

THEOREM 4.8. *If $\mathcal{C}$ is an $(|\mathcal{C}|, 7/4)$-expanding 3-CNF over n variables, then $\mathcal{C}$ has a terminator $\alpha \in \mathbb{R}^n$ with $\|\alpha\|_\infty \leq 4$ (hence $\|\alpha\|_1 \leq 4n$).*

Proof. Form the following bipartite graph G. On the left-hand side, put one vertex for each clause in $\mathcal{C}$. On the right-hand side, put 4 distinct vertices for each variable appearing in $\mathcal{C}$. Connect the vertex labeled by the clause C to all 12 vertices labeled by variables appearing in C. We do not care if the appearance is as positive or negative literals.

Since $\mathcal{C}$ is an $(|\mathcal{C}|, \frac{7}{4})$-expander, G has expansion factor 7; i.e., for all subsets S on the left-hand side, $|N(S)| \geq 7 \cdot |S|$, where $N(S)$ is the set of neighbors of S. By Hall's matching theorem [26] we conclude that there is a 7-matching from the left-hand side to the right; i.e., each node C on the left-hand side can be associated with a set of seven of its neighbors on the right-hand side (denoted $N'(C)$), such that for all clauses $C \neq D$, $N'(C) \cap N'(D) = \emptyset$. We now use N' to define our terminator α. For any variable x, if there exists a clause C such that $N'(C)$ has at least three members labeled by x, then we say x is *associated* with C, and the weight of x is the number of copies of x in $N'(C)$ (notice this weight is either 3 or 4). For any variable x_i associated with a clause C, set $\text{sign}(\alpha_i)$ to the value that satisfies C and set $|\alpha_i|$ to the weight of x_i. Set all other variables to zero. α is well defined because a

variable can be associated with at most one clause. We are left with verifying that it is a terminator. This follows by a case analysis, using the fact that each clause has a dozen neighbors, and seven of them are in $N'(C_i)$. There are three cases to consider.

C_i has at least two associated variables: In this case, $\text{sign}(\alpha)$ agrees with C on at least two variables, and each variable has weight at least 3. The remaining variable has weight at most 4, so $\langle A_i^{\mathcal{C}}, \alpha \rangle \geq 6 - 4 \geq 2$.

C_i has one associated variable of weight three: The remaining four neighbors of $N'(C_i)$ must be evenly split between the two remaining variables of C (otherwise C_i would have two associated variables). So the remaining pair of variables of C_i have weight zero. Hence $\langle A_i^{\mathcal{C}}, \alpha \rangle = 3$.

C_i has one associated variable of weight four: The remaining three neighbors of $N'(C_i)$ are split between the remaining two variables. One variable has two such neighbors (and hence zero weight) and the other has one neighbor, so the weight of this literal is at most 3. Thus, $\langle A_i^{\mathcal{C}}, \alpha \rangle \geq 4 - 3 = 1$.

Theorem 4.8 follows. □

4.3. Small weight terminators for random CNFs.

Proof of Theorem 4.1. By Theorem 4.7, (with high probability) $\mathcal{C}$ can be partitioned into the d outermost pure layers $\mathcal{C}' \stackrel{\text{def}}{=} \cup_{i=0}^{d-1} P_i$ and the remaining residual inner core $\mathcal{C}'' = \mathcal{C}_d$. This inner core is a $(|\mathcal{C}''|, 7/4)$-expander. We know (by Theorems 4.4 and 4.8, respectively) how to construct terminators for each of these formulas, so all we need to do is merge them into a single terminator for $\mathcal{C}$.

Let α', α'' be the respective terminators of $\mathcal{C}', \mathcal{C}''$. By Theorem 4.4 α' has all its support on pure literals, which do not appear in $\mathcal{C}''$. Thus the supports of α' and α'' are disjoint. We merge the two assignments by defining α as the assignment that agrees with $9 \cdot \alpha'$ on the support of α' and agrees with α'' otherwise (the reason for multiplying α' by the scalar 9 will soon become clear). By our previous remark (that α' and α'' have disjoint supports) α is well defined, and we now prove it is a terminator.

Consider a clause $C_i \in \mathcal{C}$. If $C_i \in \mathcal{C}''$, then $\langle A_i^{\mathcal{C}}, \alpha \rangle = \langle A_i^{\mathcal{C}}, \alpha'' \rangle \geq 1$, because all literals appearing in $\mathcal{C}''$ are given zero weight by α'. Otherwise, $C_i \in \mathcal{C}'$ might have some of its (nonpure) literals in $Vars(\mathcal{C}'')$, but recall that the maximal weight of α'' is 4, so in the worst case C_i has two literals with weight 4 coming from α''. Thus $\langle A_i^{\mathcal{C}}, \alpha \rangle \geq 9 - 2 \cdot 4 = 1$. We have shown the existence of a terminator of linear total weight, and the proof of Theorem 4.1 is complete. □

5. Investigating the terminator threshold. When $\mathcal{C}$ is a random CNF, $\text{Term}(\mathcal{C})$ is a random variable. Since $\text{Term}(\mathcal{C})$ bounds the running time of `RWalkSAT`, investigating this random variable is an interesting question. The property of having a terminator α with $\|\alpha\|_\infty \leq w$ is monotone with respect to addition of new clauses. Thus one can define the *terminator threshold* θ_n^w as the density for which a terminator α, $\|\alpha\|_\infty \leq w$ exists with probability 1/2.

Claim 5.1. *A CNF $\mathcal{C}$ with m clauses and n variables has some terminator iff $0 \notin$ convex hull$(\{A_i^{\mathcal{C}} : i = 1, \ldots, m\})$.*

Proof. Think of a terminator α as the normal of a hyperplane in $\mathbb{R}^n$ passing through zero. This hyperplane partitions $\mathbb{R}^n$ into two parts. $\langle A_i^{\mathcal{C}}, \alpha \rangle > 0$ iff the point $A_i^{\mathcal{C}}$ lies in the positive half of $\mathbb{R}^n$. Thus $\langle A_i^{\mathcal{C}}, \alpha \rangle > 0, i = 1, \ldots, m$, iff zero is not in the convex hull of the points. □

Füredi proved the following general theorem (he gave a tighter bound than presented here, but the form we quote is sufficient for our purposes). A set of points

$P \subset \mathbb{R}^n$ is *symmetric* if $p \in P \Rightarrow (-p) \in P$.

THEOREM 5.2 (see [25]). *Let $\{P_n \subset \mathbb{R}^n\}_{n\in\mathbb{N}}$ be an infinite family of finite symmetric sets of points. Suppose $(2+\epsilon)n$ points are selected uniformly at random from P_n. Then*

$$\lim_{n\to\infty} \Pr[0 \notin \textit{ convex hull of points}] = 0.$$

In our case P_n is the symmetric set of $\{-1, 0, 1\}$-valued points with support size 3. Thus, by Füredi's theorem when the clause density is greater than 2, with high probability there is no terminator. Notice this upper bound on the terminator threshold holds for any k-CNF, even for nonconstant k (e.g., $k = n$). Combining Theorem 4.1 with Füredi's theorem gives for $k = 3$ the following bounds:

$$1.63 \le \theta_n^\infty \le 2.$$

We leave the resolution of the terminator threshold for $k = 3$ as an interesting open problem.

For the case of 2-CNFs we can bound the terminator threshold from above by 1, because this is the satisfiability threshold for random 2-CNFs (and a terminator implies satisfiability). It seems reasonable to conjecture that for $k = 2$ the satisfiability and terminator threshold coincide. This could be used to prove that for random 2-CNFs below the satisfiability threshold, `RWalkSAT` terminates in linear time (as opposed to the quadratic upper bound guaranteed for any satisfiable 2-CNF by [35]).

6. Tightness of terminator based bounds. In this section we show that the upper bound derived by the terminator method is tight, even for pure CNFs. We present pure CNFs such that the running time of `RWalkSAT` on them is exponential in the number of variables and also lower bounded by the terminator weight.

THEOREM 6.1. *For arbitrarily large n, there exist pure 3-CNFs over n variables, with total terminator weight $\ge 2^{n/2}$, and the running time of* `RWalkSAT` *on them is $2^{\epsilon n}$ for some $\epsilon > 0$.*

Proof. Use the following formula, which is a slight variation of the X-DAG contradiction used in [9].

DEFINITION 6.2. *Let $\mathcal{G}_n$ be the following CNF over variables $x_1, \ldots, x_n, y_1, \ldots, y_n, z$:*

$$\{\bar{x}_1\} \wedge \{\bar{y}_1\} \wedge \bigwedge_{i=1}^{n-1} \{x_i \vee y_i \vee \bar{x}_{i+1}\} \wedge \bigwedge_{i=1}^{n-1} \{x_i \vee y_i \vee \bar{y}_{i+1}\} \wedge \{x_n \vee y_n \vee \bar{z}\}.$$

$\mathcal{G}_n$ has a unique satisfying assignment, $\vec{0}$. Moreover, $\mathcal{G}_n$ is n-pure, because $\bar{z}$ appears only in one clause, and once z is satisfied and removed, $\bar{y}_n, \bar{x}_n$ each appear in one clause in the remaining formula. Thus, one can repeatedly remove x_{i-1}, y_{i-1} until all the formula is satisfied. This implies the existence of a terminator of weight 3^n, and it is not hard to see that any terminator must have weight 2^n at least. We claim that `RWalkSAT` requires exponential time to succeed on $\mathcal{G}_n$.

Let X_t be the number of ones assigned by α_t to the variables $x_2, \ldots, x_n, y_2, \ldots, y_n$. With high probability $X_0 > (1-\epsilon)n$, and if `RWalkSAT`$(\mathcal{G}_n, T)$ succeeds, we know $X_T = 0$. But for every step t, the probability of X_t decreasing is at most $1/3$. The theorem follows. ∎

7. Lower bounds for large density planted SAT. In this section, we state (without proofs) that RWalkSAT is not a good algorithm for random CNFs with large clause density. By definition, RWalkSAT gives the correct answer on any unsatisfiable formula. For large enough clause density ($\Delta > 4.6$), almost all formulas in $\mathbb{F}^n_\Delta$ are unsatisfiable [16]. Thus, one may argue that RWalkSAT operates very well for these densities. On second thought, on this distribution, even the constant time algorithm that fails on every input, without reading it, operates well. Thus, it makes sense to discuss the performance of RWalkSAT only on the uniform distribution over *satisfiable* formulas with Δn clauses (denoted $\mathbf{SAT}^\Delta_n$). The problem is that for small densities, $\mathbf{SAT}^\Delta_n$ is not well characterized, we do not know how to analyze it. Thus, we propose looking at the following pair of *planted SAT* distributions over satisfiable 3-CNFs.

DEFINITION 7.1 (planted SAT). *Let $\mathbb{S}^n_\Delta$ be the distribution obtained by selecting at random $\beta \in \{0,1\}^n$ and selecting at random Δn clauses out of all clauses of size 3 that are satisfied by β. Denote a random formula from this distribution by $\mathcal{C} \sim \mathbb{S}^n_\Delta$.*

Let $\mathbb{P}^n_\Delta$ be the distribution obtained by selecting at random $\beta \in \{0,1\}^n$, and for each clause C satisfied by β, select C to be in $\mathcal{C}$ with independent probability $p^\Delta_n = \frac{6\Delta}{7(n-1)(n-2)}$. Denote a random formula from this distribution by $\mathcal{C} \sim \mathbb{P}^n_\Delta$.

This distribution is highly interesting in its own right. It is the analogue of the *planted clique* and *planted bisection* distributions, studied, e.g., in [5, 23, 28]. There are efficient spectral algorithms for finding the satisfying assignment for the planted SAT distribution [18], and in this section we argue that RWalkSAT performs poorly (takes exponential running time) on this distribution. The proofs of this result are fairly straightforward, so we omit them from the paper.

THEOREM 7.2 (main lower bound). *There exists a constant $\Delta_0 > 0$, such that for all $\Delta \geq \Delta_0$ (Δ may be a function of n), with high probability for $\mathcal{C} \sim \mathbb{P}^n_\Delta$, or $\mathcal{C} \sim \mathbb{S}^n_\Delta$*

$$\mathbf{P}[\texttt{RWalkSAT}(\mathcal{C}, 2^{\epsilon n}) \text{ succeeds}] \leq 2^{-\epsilon n},$$

where $\epsilon > 0$ is some a constant, depending on Δ.

The rest of this section is devoted to a sketch of the proof of Theorem 7.2. We warm up by discussing the case of $\mathcal{C}$ being the maximal size CNF satisfying β and then apply our insights to the case of a random CNF. For the rest of this section we assume without loss of generality that β, the random planted assignment, is the all zero vector, denoted $\vec{0}$.

The full CNF of size n, denoted $\mathcal{F}_n$, has all clauses of size exactly 3 (without repetition of literals) that are satisfied by $\vec{0}$. Our starting point is the following.

LEMMA 7.3. $\mathbf{P}[\texttt{RWalkSAT}(\mathcal{F}_n, 2^{n/100}) \text{ succeeds}] \leq 2^{-n/100}$.

Intuitively, the lemma holds because for an assignment that is very close to $\vec{0}$, the fraction of falsified clauses that have two (or more) positive literals is significantly larger than the fraction of falsified clauses with only one positive literal. Thus, a random falsified clause is more likely to lead us away from $\vec{0}$ and hence the exponential running time.

To complete the proof of Theorem 7.2, notice $\mathcal{C} \sim \mathbb{P}^n_\Delta$ is a "random fraction" of $\mathcal{F}_n$. Additionally, for large Δ all satisfying assignments are close to $\vec{0}$. Thus, when the random walk algorithm reaches an assignment that is close to $\vec{0}$, the fraction of clauses with two or more positive literals is significantly larger than the fraction of falsified clauses with one positive literal. Thus, as in the case of $\mathcal{F}_n$, RWalkSAT is more likely to move away from $\vec{0}$ than to approach it, resulting in exponential running time. This completes the sketch of the proof of Theorem 7.2.

8. Open problems.

1. What is the largest Δ for which one can prove `RWalkSAT` to have polynomial running time on $\mathcal{C} \sim \mathbb{F}_{\Delta}^{n}$?
2. What are the statistics of the random variable $\mathrm{Term}(\mathcal{C})$ as a function of the clause density? Does $\mathrm{Term}(\mathcal{C}) < \infty$ have a sharp threshold? Is there a terminator threshold independent of n? How does $\mathrm{Term}(\mathcal{C})$ correspond to n (number of variables) above density 1.63 (below 1.63 it is linear)?

Appendix A. Proofs. In this section we prove Theorems 4.5 and 4.7. Our starting point is the following theorem and lemma proved implicitly in [10]. The lemma is a slight generalization of Lemma 4.4 in [10], so we provide its proof. (The original Lemma 4.4 of [10] needed only expansion factor of 3/2, whereas we need a constant fraction more than 3/2. The proof is essentially the same.)

THEOREM A.1 (see [10]). *For every $\Delta < 1.63$, there exists an integer d such that with high probability for $\mathcal{C} \sim \mathbb{F}_{\Delta}^{n}$, $|\mathcal{C}_d| \le \frac{n}{600\Delta^2}$.*

LEMMA A.2 (see [10]). *Let $\Delta_0 = 1.63$. For any constant $\Delta \le \Delta_0$ with high probability $\mathcal{C} \sim \mathbb{F}_{\Delta}^{n}$ is a $(\frac{n}{600\Delta_0^2}, 3/2 + 10^{-3})$-expander.*

Proof of Lemma A.2. Set $\epsilon = 10^{-3}$. Let A_k be the event that there exists a set of k clauses having less than $3/2 + \epsilon$ variables. Let us bound the probability of these bad events, using a union bound. Let $r = \frac{n}{600\Delta_0^2}$ and $c = 3/2 + \epsilon$. We make use of the following well-known inequality $\binom{n}{k} \le \left(\frac{en}{k}\right)^k$:

$$\begin{aligned}
\mathbf{P}[Bad] &\le \sum_{k=1}^{r} \mathbf{P}[A_k] \le \sum_{k=1}^{r} \binom{\Delta n}{k} \cdot \binom{n}{ck} \cdot \left(\frac{ck}{n}\right)^{3k} \\
&\le \sum_{k=1}^{r} \left(\frac{e\Delta_0 n}{k}\right)^{k} \cdot \left(\frac{en}{ck}\right)^{ck} \cdot \left(\frac{ck}{n}\right)^{3k} \\
&\le \sum_{k=1}^{r} \left[(e^{1+c} c^{3-c} \Delta_0) \cdot \left(\frac{k}{n}\right)^{2-c} \right]^{k} \\
&\le \sum_{k=1}^{r} \left[37 \cdot \left(\frac{k}{n}\right)^{\frac{1}{2}-\epsilon} \right]^{k} = o(1),
\end{aligned}$$

where the last inequality holds for $r \le \frac{n}{600\Delta_0^2}$. ∎

Notice that if $\mathcal{C}_d$ is a $(|\mathcal{C}_d|, \frac{3}{2} + \epsilon)$-expander, then every subset of $\mathcal{C}_d$ (including $\mathcal{C}_d$ itself) has at least ϵ *unique neighbors* (i.e., literals appearing in exactly one clause), and these unique neighbors are pure. Thus, $\mathcal{C}_d$ is $O(\log n)$-pure. Hence $\mathcal{C}$ is $O(\log n)$-pure (remember d is a constant), and this proves Theorem 4.5. In order to prove Theorem 4.7 we need the following lemma from [13].

LEMMA A.3 (see [13]). *For all constants $\Delta > 0, c < 2$, there exists some constant $\delta > 0$ such that with high probability $\mathcal{C} \sim \mathbb{F}_{\Delta}^{n}$ is a $(\delta\Delta n, c)$-expander.*

Let δ be the constant promised by Lemma A.3 for $\Delta = 1.63$ and $c = 7/4$. By Theorem 4.5, $|\mathcal{C}_d| \le n/(600\Delta_0^2)$ for some constant d. By Lemma A.2, $\mathcal{C}_d$ is a $(|\mathcal{C}_d|, \epsilon)$-boundary expander for some $\epsilon > 0$. Remove an additional d' layers from $\mathcal{C}$ (each containing at least an $\epsilon/3$ fraction of the remaining clauses) so that $|\mathcal{C}_{d+d'}| \le \delta n$, and by Lemma A.3 this remaining CNF is (with high probability) a $(|\mathcal{C}_{d+d'}|, 7/4)$-expander. This proves Theorem 4.7.

Acknowledgments. We thank Madhu Sudan for many useful discussions. We thank Bart Selman and Andrew Parkes for valuable information on the empirical results regarding `RWalkSAT` and Balint Virag for his help with the analysis of martingales. We thank Jon Feldman for providing code for running LP simulations for empirical investigation of the terminator threshold and Jeong Han Kim (via private communication) for allowing us to include the upper bound on the terminator threshold (section 5) in the paper. The second author thanks Rocco Servedio, Salil Vadhan, and Dimitris Achlioptas for helpful discussions. Finally, we thank the anonymous referees for helpful remarks.

REFERENCES

[1] D. Achlioptas, *Setting two variables at a time yields a new lower bound for random* 3*-SAT*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, 2000, pp. 28–37.

[2] D. Achlioptas, *Lower bounds for random* 3*-SAT via differential equations*, Theoret. Comput. Sci., 285 (2001), pp. 159–185.

[3] D. Achlioptas and G. B. Sorkin, *Optimal myopic algorithms for random* 3*-SAT*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, 2000, pp. 590–600.

[4] D. Achlioptas, P. Beame, and M. Molloy, *A sharp threshold in proof complexity*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, 2001, pp. 337–346.

[5] N. Alon, M. Krivelevich, and B. Sudakov, *Finding a large hidden clique in a random graph*, Random Structures Algorithms, 13 (1998), pp. 457–466.

[6] N. Alon and J. Spencer, *The Probabilistic Method*, 2nd ed., Wiley, New York, 2000.

[7] K. Azuma, *Weighted sums of certain dependent random variables*, Tôhoku Math. J., 19 (1967), pp. 357–367.

[8] S. Baumer and R. Schuler, *Improving a probabilistic* 3*-SAT algorithm by dynamic search and independent clause pairs*, in Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing, Lecture Notes in Comput. Sci. 2919, Springer, Berlin, 2004, pp. 150–161.

[9] E. Ben-Sasson, *Size space tradeoffs for resolution*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, 2002, pp. 457–464.

[10] A. Broder, A. Frieze, and E. Upfal, *On the satisfiability and maximum satisfiability of random* 3*-CNF formulas*, in Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, 1993, pp. 322–330.

[11] M. T. Chao and J. Franco, *Probabilistic analysis of two heuristics for the* 3*-satisfiability problem*, Inform. Sci., 51 (1990), pp. 289–314.

[12] V. Chvátal and B. Reed, *Mick gets some (the odds are on his side)*, in Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science, 1992, pp. 620–627.

[13] V. Chvátal and E. Szemerédi, *Many hard examples for resolution*, J. Assoc. Comput. Mach., 35 (1988), pp. 759–768.

[14] J. M. Crawford and L. D. Auton, *Experimental results on the crossover point in random* 3*-SAT*, Artificial Intelligence, 81 (1996), pp. 31–57.

[15] E. Danstin, A. Goerdt, E. A. Hirsch, J. Kleinberg, C. Papadimitriou, P. Raghavan, and U. Schoning, *A deterministic* $2 - \frac{2}{k+1}$ *algorithm for k-SAT based on local search*, Theoret. Comput. Sci., 223 (1999), pp. 1–72.

[16] O. Dubois, Y. Boufkhad, and J. Mandler, *Typical random* 3*-SAT formulae and the satisfiability threshold*, in Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, 2000, pp. 126–127.

[17] U. Feige, *Relations between average case complexity and approximation complexity*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, 2002, pp. 534–543.

[18] A. Flaxman, *A spectral technique for random satisfiable* 3*CNF formulas*, in Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2003, pp. 357–363.

[19] A. Frieze and S. Suen, *Analysis of two simple heuristics for random instances of k-SAT*, J. Algorithms, 20 (1996) pp. 312–355.

[20] M. T. Hajiaghayi and G. B. Sorkin, *The Satisfiability Threshold of Random* 3*-SAT Is at Least* 3.52, IBM Research Report RC22942, 2003, submitted.

[21] T. Hofmeister, U. Schoning, R. Schuler, and O. Watanabe, *Probabilistic* 3*-SAT algorithm*

further improved, in Proceedings of the 19th International Symposium on Theoretical Aspects of Computer Science, 2002, pp. 193–202.

[22] K. Iwama and S. Tamaki, *Improved bounds for* 3-*SAT*, in Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2004, pp. 321–322.

[23] U. Feige and R. Krauthgamer, *Finding and certifying a large hidden clique in a semi-random graph*, Random Structures Algorithms, 16 (2000), pp. 195–208.

[24] E. Friedgut, *Sharp thresholds of graph properties, and the k-sat problem*, J. Amer. Math. Soc., 12 (1999), pp. 1017–1054.

[25] Z. Füredi, *Random polytopes in the d-dimensional cube*, Discrete Comput. Geom., 1 (1986), pp. 315–319.

[26] P. Hall, *On representatives of subsets*, J. London Math. Soc., 10 (1935), pp. 26–30.

[27] W. Heoffding, *Probability inequalities for sums of bounded random variables*, J. Amer. Statist. Assoc., 58 (1963), pp. 13–30.

[28] M. Jerrum and G. B. Sorkin, *Simulated annealing for graph bisection*, in Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science, 1993, pp. 94–103.

[29] A. Kaporis, L. M. Kirousis, and E. G. Lalas, *The probabilistic analysis of a greedy satisfiability algorithm*, in Proceedings of the 10th Annual European Symposium on Algorithms, Rome, Italy, 2002.

[30] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, *Optimization by simulated annealing*, Science, 220 (1983), pp. 671–680.

[31] M. Luby, M. Mitzenmacher, and A. Shokrollahi, *Analysis of random processes via and-or tree evaluation*, in Proceeding of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, 1998, pp. 364–373.

[32] M. Mézard, G. Parisi, and R. Zecchina, *Analytic and algorithmic solution of random satisfiability problems*, Science, 297 (2002), pp. 812–815.

[33] M. Mézard and R. Zecchina, *Random k-satisfiability: From an analytic solution to an efficient algorithm*, Phys. Rev. E, 66 (2002), 056126.

[34] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.

[35] C. H. Papadimitriou, *On selecting a satisfying truth assignment*, in Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science, 1991, pp. 163–169.

[36] A. J. Parkes, *private communication*.

[37] D. Rolf, 3-*SAT in* $RTIME(O(1.32793^n))$*—Improving Randomized Local Search by Initializing Strings of* 3-*Clauses*, ECCC report TR03-054, 2003.

[38] U. Schoning, *A probabilistic algorithm for k-SAT and constraint satisfaction problems*, in Proceedings of the 40th Annual Symposium on Foundations of Computer Science, 1999, pp. 410–414.

[39] B. Selman, *private communication*.

[40] B. Selman and H. Kautz, *Local search strategies for satisfiability testing*, in Proceedings of the Second DIMACS Challenge on Cliques, Coloring, and Satisfiability, AMS, Providence, RI, 1993, pp. 521–532.

[41] B. Selman, H. Levesque, and D. Mitchell, *A new method for solving hard satisfiability problems*, in Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92), San Jose, CA, 1992, pp. 440–446.

SIAM J. COMPUT.
Vol. 36, No. 5, pp. 1264–1300

THE COMPLEXITY OF COMPUTING THE SIZE OF AN INTERVAL*

LANE A. HEMASPAANDRA†, CHRISTOPHER M. HOMAN‡, SVEN KOSUB§, AND KLAUS W. WAGNER¶

Abstract. Given a p-order A over a universe of strings (i.e., a transitive, reflexive, antisymmetric relation such that if $(x, y) \in A$, then $|x|$ is polynomially bounded by $|y|$), an interval size function of A returns, for each string x in the universe, the number of strings in the interval between strings $b(x)$ and $t(x)$ (with respect to A), where $b(x)$ and $t(x)$ are functions that are polynomial-time computable in the length of x. By choosing sets of interval size functions based on feasibility requirements for their underlying p-orders, we obtain new characterizations of complexity classes. We prove that the set of all interval size functions whose underlying p-orders are polynomial-time decidable is exactly #P. We show that the interval size functions for orders with polynomial-time adjacency checks are closely related to the class FPSPACE(poly). Indeed, FPSPACE(poly) is exactly the class of all nonnegative functions that are an interval size function minus a polynomial-time computable function. We study two important functions in relation to interval size functions. The function #DIV maps each natural number n to the number of nontrivial divisors of n. We show that #DIV is an interval size function of a polynomial-time decidable partial p-order with polynomial-time adjacency checks. The function #MONSAT maps each monotone boolean formula F to the number of satisfying assignments of F. We show that #MONSAT is an interval size function of a polynomial-time decidable total p-order with polynomial-time adjacency checks. Finally, we explore the related notion of cluster computation.

Key words. computational complexity, interval size functions, cluster computing, counting functions

AMS subject classifications. 03D15, 06A05, 06A06, 68Q05, 68Q10, 68Q15, 68Q17

DOI. 10.1137/S0097539705447013

1. Introduction.

The class NP, which is widely believed to contain computationally intractable problems, captures the complexity of determining for a given problem instance whether at least one suitable affirmative solution exists within an exponentially large set of (polynomial-sized) potential solutions. It is certainly not simpler, and seemingly much harder, to count all affirmative solutions in such solution sets. The corresponding *counting functions* constitute Valiant's widely studied counting class #P [Val79]. In the theory of counting functions, which is devoted to the study of counting versions of decision problems, most classes considered try to capture the pure phenomenon of counting, and in doing so they obscure other factors, e.g., orders on solution sets.

*Received by the editors February 13, 2005; accepted for publication (in revised form) April 25, 2006; published electronically December 21, 2006. A preliminary version of some parts of this paper was presented at the 28th International Colloquium on Automata, Languages and Programming held in Crete, Greece, in July 2001 [HKW01]. This work was supported in part by grants NSF-CCR-9322513, NSF-INT-9815095/DAAD-315-PPP-gü-ab, and NSF-CCF-0426761. This work was done in part while the second author was at the University of Rochester, and in part while the first two authors were visiting Julius-Maximilians-Universität Würzburg.

http://www.siam.org/journals/sicomp/36-5/44701.html

†Department of Computer Science, University of Rochester, Rochester, NY 14627 (www.cs.rochester.edu/u/lane).

‡Department of Computer Science, Rochester Institute of Technology, Rochester, NY 14623 (www.cs.rit.edu/~cmh).

§Institut für Informatik, Technische Universität, München, D-85748 Germany (www14.in.tum.de/personen/kosub).

¶Institut für Informatik, Julius-Maximilians-Universität Würzburg, D-97074 Würzburg, Germany (www4.informatik.uni-wuerzburg.de/personen/mitarbeiter/wagner).

Natural counting problems in #P, of course, sometimes exhibit strong relationships between solutions to the problems. As an example, consider the counting function #DIV, which counts for each natural number the number of its nontrivial divisors. Clearly, #DIV is in #P since division can be done in polynomial time. A suitable structure in the set of solutions is the partial order of divisibility, that is, the order defined by $n \leq_| m$ if and only if n divides m. Obviously, $\#\mathrm{DIV}(m) = \|\{k \mid 1 <_| k <_| m\}\|$, i.e., $\#\mathrm{DIV}(m)$ counts the number of elements in the open interval $(1, m)$ in the partial order "$\leq_|$" on natural numbers.

Is #DIV an exceptional case among #P functions in that it has such an interval size characterization? Interestingly, "no" is the answer. It turns out that a function f is in #P if and only if it is an interval size function of a P-decidable partial p-order. The latter means that there exist a *partial p-order* A (i.e., A is a partial order and in addition satisfies the requirement that for some polynomial p and all x and y, it holds that $x \leq_A y$ implies $|x| \leq p(|y|)$) that is P-*decidable* (i.e., $x \leq_A y$ is decidable in polynomial time) and polynomial-time computable functions b and t such that $f(x) = \|\{z \mid b(x) <_A z <_A t(x)\}\|$, where $a <_A b$ denotes $a \leq_A b \wedge a \neq b$.

However, knowing that a partial p-order is polynomial-time decidable does not give us as much information as sometimes is needed. For example, the polynomial-time decidability of a p-order seemingly does not ensure that it has *efficient adjacency checks*, i.e., that there is a polynomial-time algorithm checking whether two elements are adjacent in this partial p-order. Indeed, if every P-decidable partial p-order has efficient adjacency checks, then P = NP (and vice versa). Hence adding efficient adjacency checks to the properties listed above seems to be a restriction. Denote by $\mathrm{IF_p}$ the class of interval size functions of P-decidable partial p-orders with efficient adjacency checks. Denote by $\mathrm{IF_t}$ the class of interval size functions of P-decidable total p-orders with efficient adjacency checks. We have $\mathrm{IF_t} \subseteq \mathrm{IF_p} \subseteq \#\mathrm{P}$. Are these containments proper? On one hand, we prove that $\mathrm{IF_t} \text{ - } \mathrm{FP} = \mathrm{IF_p} \text{ - } \mathrm{FP} = \#\mathrm{P} \text{ - } \mathrm{FP}$, where $A \text{ - } B = \{a - b \mid a \in A \wedge b \in B\}$. Thus these three classes do not seem to be very different; indeed, they are identical given the smoothing power of subtracting polynomial-time computable adjustments. On the other hand, $\mathrm{IF_p} = \#\mathrm{P}$ is equivalent to P = NP, and $\mathrm{IF_t} = \mathrm{IF_p}$ only if UP = PH. Thus it is unlikely that any two of $\mathrm{IF_t}$, $\mathrm{IF_p}$, and #P coincide. Further, we study relationships between the classes $\mathrm{IF_t}$, FP, and UPSV_t.

We already mentioned that it is unlikely that every P-decidable partial p-order has efficient adjacency checks. What about the converse? This also is not likely; if every partial p-order with efficient adjacency checks is P-decidable, then P = PSPACE (and vice versa). Hence, in the presence of efficient adjacency checks, removing the P-decidability requirement seems to be a relaxation. Denote by $\mathrm{IF_p^*}$ the class of interval size functions of partial p-orders with efficient adjacency checks. Denote by $\mathrm{IF_t^*}$ the class of interval size functions of total p-orders with efficient adjacency checks. We have $\mathrm{IF_p} \subseteq \mathrm{IF_p^*}$ and $\mathrm{IF_t} \subseteq \mathrm{IF_t^*} \subseteq \mathrm{IF_p^*} \subseteq \mathrm{FPSPACE(poly)}$. We prove that $\mathrm{IF_t^*}$ (and $\mathrm{IF_p^*}$) are remarkably powerful: $\mathrm{IF_t^*} \text{ - } \mathrm{FP} = \mathrm{FPSPACE(poly)} \text{ - } \mathrm{FP}$. Thus $\mathrm{IF_t^*}$ (and $\mathrm{IF_p^*}$) are in a certain sense close to FPSPACE(poly), the class of polynomially length-bounded, polynomial-space computable functions. Nonetheless, we show that if these classes coincide, then UP = PSPACE. We clarify further relationships among such classes and also with respect to other function classes, in order to understand the power of interval computing.

We study two important natural functions in relation to interval size functions. We prove that the counting function #DIV is in $\mathrm{IF_p}$. Also, we show that the func-

tion #MONSAT, which counts for each monotone boolean formula the number of satisfying assignments that it has, belongs to IF_{t}.

Using order-theoretic notions to approach complexity issues has a rich tradition and appears in the literature in a variety of settings (e.g., [GHJY91, GS91, VW95, HVW96, Kos99]). The approaches in the examples just cited differ in intent from our approach in that they are based on a specific ordering, namely the lexicographical ordering. In contrast, for our purposes it is essential to consider more general feasible orderings (see [MP79, Ko83]).

Among earlier studies, perhaps the notion lying nearest to our approach is that of a cluster machine, which is a nondeterministic Turing machine that satisfies the promise that, on each input, all accepting computation paths are always neighbors with respect to the lexicographical ordering, i.e., the accepting paths must form a "cluster" [Kos99]. Based on this machine type, Kosub [Kos99] defined the counting class c#P (in a manner analogous to the way that #P is based on standard, nondeterministic polynomial-time Turing machines). Kosub obtained many interesting results about c#P, e.g., c#P seems to differ dramatically from #P in its closure properties (as regards, e.g., integer division, see [OH93, Kos99]), and he showed that c#P is closely related to a relatively simple unambiguous-nondeterminism-based function class, "$\mathrm{UPSV}_{\mathrm{t}}$."

Most of the known results about c#P are proven by techniques that are exceedingly dependent on the fact that c#P is defined using adjacency clusters *with respect to lexicographic order*. In particular, the fact that in lexicographic order the function $f(a,b) = \|\{c \mid a \leq_{\mathrm{lex}} c \leq_{\mathrm{lex}} b\}\|$ is easy to compute underpins the results.

In the present paper we define the class CL#P, which studies the complexity of cluster computing in a context of relatively general (though length-respecting and having efficient adjacency checks) orders, rather than merely in the extremely special case of lexicographic order. We study CL#P and show, for example, that it does not equal c#P unless UP = PP (and thus the polynomial hierarchy collapses). On the other hand, we also prove that c#P and CL#P coincide on polynomially bounded functions, and that CL#P shows some behaviors quite reminiscent of c#P, e.g., though #P is closed under increment, we show that CL#P is closed under increment only if unexpected complexity collapses occur. More generally, we explore the relationship between CL#P and such classes as IF_{t}, IF_{p}, and #P. Though CL#P is in general flavor like an interval function (over a total order satisfying appropriate conditions but freed from the polynomial-time computability constraints of the functions defining the top and bottom of the interval), our results usually show that CL#P differs from the these classes unless unexpected complexity class collapses occur.

2. Preliminaries. Fix our finite alphabet to be $\Sigma = \{0,1\}$, and let Σ^* denote the set of all finite strings over Σ. Let ε denote the empty string. The length of a string $x \in \Sigma^*$ is denoted by $|x|$. The set of all strings of length n is denoted by Σ^n. The complement of a set $L \subseteq \Sigma^*$ is denoted by $\overline{L}$, i.e., $\overline{L} = \Sigma^* \setminus L$. For any class $\mathcal{K}$ of subsets of Σ^*, let co$\mathcal{K}$ be the class $\{L \subseteq \Sigma^* \mid \overline{L} \in \mathcal{K}\}$. The cardinality of a finite set S is denoted by $\|S\|$. The characteristic function of a set $L \subseteq \Sigma^*$ is denoted by χ_L, i.e., for all $x \in \Sigma^*$, $\chi_L(x) = 1 \Leftrightarrow x \in L$ and $\chi_L(x) = 0 \Leftrightarrow x \notin L$. Let $\mathbb{N}$ denote the set $\{0,1,2,\dots\}$. Let $\mathbb{N}^+$ denote the set $\{1,2,3,\dots\}$.

For the basic notions of complexity theory such as P, NP, PSPACE, and so on see, e.g., the handbook [HO02].

The computation model we use is the standard nondeterministic Turing machine.

We review the definitions of some complexity classes of functions, already existing

in the literature, that we will use in this paper.

- FP is the class of all (deterministic) polynomial-time computable, total functions from Σ^* to $\mathbb{N}$. We will at times use FP to mean the class of all polynomial-time computable, total functions from Σ^* to Σ^*. Via the natural, efficient bijection between $\mathbb{N}$ and Σ^*, these two notions are essentially the same.
- [Lad89] FPSPACE(poly) is the class of all polynomial-space computable, total functions from Σ^* to $\mathbb{N}$ having polynomially length-bounded outputs. We will at times use FPSPACE(poly) to mean the class of all polynomial-space computable, total functions from Σ^* to Σ^* having polynomially length-bounded outputs. Via the natural, efficient bijection between $\mathbb{N}$ and Σ^*, these two notions are essentially the same.
- [Val79] #P is the class of all total functions f for which there exists a nondeterministic polynomial-time Turing machine M such that, for each x, $f(x)$ is the number of accepting computations of $M(x)$. Equivalently, #P is the class of all total functions f for which there exist a set $B \in \mathrm{P}$ and a polynomial p such that, for all $x \in \Sigma^*$, $f(x) = \|\{z \mid |z| = p(|x|) \wedge (x, z) \in B\}\|$.
- [GS88, Kos99] $\mathrm{UPSV_t}$ is the class of all total functions f for which there exists a nondeterministic polynomial-time Turing machine M that, on each input $x \in \Sigma^*$, has exactly one accepting path, and the output of this unique accepting path is $f(x)$.

For function classes $\mathcal{F}$ and $\mathcal{G}$ where each $f \in \mathcal{F} \cup \mathcal{G}$ maps from Σ^* to $\mathbb{N}$, let $\mathcal{F} - \mathcal{G}$ denote the class of all functions $\{f - g \mid f \in \mathcal{F} \text{ and } g \in \mathcal{G}\}$. Note that the codomain of $\mathcal{F} - \mathcal{G}$ functions is $\{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$. For each class $\mathcal{K}$ of sets, let $\mathrm{FP}^{\mathcal{K}}$ (respectively, $\mathrm{P}^{\mathcal{K}}$) be the class of functions (respectively, sets) that can be computed in polynomial time with an oracle from $\mathcal{K}$.

Next, we review the definitions of some complexity classes (of sets), already existing in the literature, that we will use in this paper.

- [Val76] UP is the class of all sets L such that $\chi_L \in \#\mathrm{P}$.
- [Coo71, Lev75] NP is the class of all sets L for which there exists a function $f \in \#\mathrm{P}$ such that, for all $x \in \Sigma^*$, $x \in L \Leftrightarrow f(x) > 0$.
- [Sim75, Gil77] PP is the class of all sets L for which there exist functions $f \in \#\mathrm{P}$ and $g \in \mathrm{FP}$ such that, for all $x \in \Sigma^*$, $x \in L \Leftrightarrow f(x) \geq g(x)$.
- [OH93, FFK94] SPP is the class of all sets L such that $\chi_L \in \#\mathrm{P} - \mathrm{FP}$.
- [CH90] Few is the class of all sets L for which there exist a function $f \in \#\mathrm{P}$, a set $B \in \mathrm{P}$, and a polynomial p such that, for all $x \in \Sigma^*$, $f(x) \leq p(|x|)$ and $x \in L \Leftrightarrow (x, 1^{f(x)}) \in B$. In this definition, changing from "$f(x) \leq p(|x|)$" to "$0 < f(x) \leq p(|x|)$" can easily be seen to also yield Few.
- [MS72, Sto77] $\mathrm{PH} = \mathrm{P} \cup \mathrm{NP} \cup \mathrm{NP}^{\mathrm{NP}} \cup \mathrm{NP}^{\mathrm{NP}^{\mathrm{NP}}} \cup \ldots$.

The following results are well-known or easy to see.

PROPOSITION 2.1.

1. $\mathrm{FP} \subseteq \mathrm{UPSV_t} = \mathrm{FP}^{\mathrm{UP} \cap \mathrm{coUP}} \subseteq \#\mathrm{P} \subseteq \mathrm{FPSPACE(poly)}$.
2. $\mathrm{P} \subseteq \mathrm{UP} \subseteq \mathrm{Few} \cap \mathrm{NP} \subseteq \mathrm{Few} \cup \mathrm{NP} \subseteq \mathrm{P}^{\mathrm{NP}} \subseteq \mathrm{PH} \subseteq \mathrm{PSPACE}$.
3. $\mathrm{NP} \cup \mathrm{SPP} \subseteq \mathrm{PP}$.
4. [KSTT92] $\mathrm{Few} \subseteq \mathrm{SPP}$.

In this paper, we will sometimes for conciseness refer to the jth part of Theorem i as Theorem $i.j$, e.g., we may refer to the third part of the above proposition as Proposition 2.1.3.

We will use the complexity-theoretic function-to-set operator $\exists$ of Hempel and

Wechsung [HW00], which maps function classes to set classes. For a function class $\mathcal{F}$, $\exists \cdot \mathcal{F}$ is the class of all sets L for which there exists a function $f \in \mathcal{F}$ such that, for all $x \in \Sigma^*$, $x \in L \Leftrightarrow f(x) > 0$.

The following statements are easy to see.

PROPOSITION 2.2.

1. $\exists \cdot \mathrm{FP} = \exists \cdot (\mathrm{FP} - \mathrm{FP}) = \mathrm{P}$.
2. $\exists \cdot \mathrm{UPSV_t} = \exists \cdot (\mathrm{UPSV_t} - \mathrm{FP}) = \exists \cdot (\mathrm{UPSV_t} - \mathrm{UPSV_t}) = \mathrm{UP} \cap \mathrm{coUP}$.
3. $\exists \cdot \#\mathrm{P} = \mathrm{NP}$.
4. $\exists \cdot (\#\mathrm{P} - \mathrm{FP}) = \mathrm{PP}$.
5. $\exists \cdot \mathrm{FPSPACE(poly)} = \mathrm{PSPACE}$.

3. Orders with feasibility constraints. In this section, we define the notions of ordering that we use for the remainder of this paper (see also [Ko83]).

A binary relation $A \subseteq \Sigma^* \times \Sigma^*$ is a *partial order* if it is reflexive, antisymmetric (i.e., $(\forall x, y \in \Sigma^*)[x \neq y \implies ((x,y) \notin A \vee (y,x) \notin A)]$), and transitive. A partial order A is a *total order* if, for all $x, y \in \Sigma^*$, $(x,y) \in A$ or $(y,x) \in A$. A partial order A is a *partial p-order* if there exists a polynomial q such that for all $(x,y) \in A$ it holds that $|x| \leq q(|y|)$.

For any partial p-order A, we employ the following standard notational conventions. We write $x \leq_A y$ if $(x,y) \in A$. We write $x <_A y$ if $x \leq_A y$ and $x \neq y$. We write $x \prec_A y$ if $x <_A y$ and there is no z such that $x <_A z <_A y$. If $x \prec_A y$, we say that x *precedes* y or, equivalently, y *succeeds* x. We let $A_\prec =_{\mathrm{def}} \{(x,y) \mid x \prec_A y\}$. The lexicographical order is denoted by $\leq_{\mathrm{lex}}$, and lexicographical adjacency is denoted by $\prec_{\mathrm{lex}}$.

Note that, for every partial p-order A and every string y, there exist at most exponentially (in the length of y) many strings that are less than y with respect to A. Thus, the output of an interval size function on a partial p-order is always at most exponential in the input length. Note that such exponential value bounds are typically the case with function classes, such as FP and #P, that are based on Turing machines having polynomial-time running bounds.

Feasibility constraints on orders are essential to our study. A partial p-order A is P-*decidable* if $A \in \mathrm{P}$. A partial p-order A is said to have *efficient adjacency checks* if $A_\prec \in \mathrm{P}$.

There are complexity-theoretic connections between these two feasibility requirements.

PROPOSITION 3.1. *Let A be a partial p-order.*

1. *If $A \in \mathrm{P}$, then $A_\prec \in \mathrm{coNP}$.*
2. *If $A_\prec \in \mathrm{P}$, then $A \in \mathrm{PSPACE}$.*

Proof. The proof of (1) is immediate.

For (2), let A be a partial p-order that has efficient adjacency checks. Let M be an NPSPACE machine that accepts A by, on input (x,y), accepting immediately if $x = y$ and otherwise guessing a sequence $z_1, \ldots, z_k$ such that $x \prec_A z_1 \prec_A z_2 \prec_A \cdots \prec_A z_k \prec_A y$. Since A is a partial p-order, for each $i \in \{1, \ldots, k\}$, $|z_i|$ is polynomially bounded with respect to $|y|$, so we need only guess such z_i's whose lengths are polynomially bounded in $|y|$. So $A \in \mathrm{NPSPACE}$. However, as is well-known, $\mathrm{NPSPACE} = \mathrm{PSPACE}$. □

COROLLARY 3.2.

1. *If $\mathrm{P} = \mathrm{NP}$, then all P-decidable partial p-orders have efficient adjacency checks.*

2. *If* P = PSPACE*, then all partial p-orders with efficient adjacency checks are* P*-decidable.*

In what follows we will see that the converse of each of the claims of Corollary 3.2 also holds.

4. Orders without efficient adjacency checks. We say that a function $f : \Sigma^* \to \mathbb{N}$ is an *interval size function* if there exist *boundary functions* b and t mapping from Σ^* to Σ^* and a partial order $A \subseteq \Sigma^* \times \Sigma^*$ such that, for all $x \in \Sigma^*$, $f(x) = \|\{z \mid b(x) <_A z <_A t(x)\}\|$. In this section, we characterize #P in terms of interval size functions with polynomial-time decidable p-orders and polynomial-time computable boundary functions. We also note that if we omit all feasibility restrictions on p-orders, then all polynomially length-bounded functions can be characterized in a manner analogous to the way that interval size functions of resource-bounded orders characterize #P.

THEOREM 4.1.

1. *For any function f, the following statements are equivalent.*
   (a) $f \in$ #P*.*
   (b) *There exist a partial p-order $A \in$ P and functions $b, t \in$ FP such that, for all $x \in \Sigma^*$, $f(x) = \|\{z \mid b(x) <_A z <_A t(x)\}\|$.*
   (c) *There exist a total p-order $A \in$ P and functions $b, t \in$ FP such that, for all $x \in \Sigma^*$, $b(x) \leq_A t(x)$ and $f(x) = \|\{z \mid b(x) <_A z <_A t(x)\}\|$.*
2. *For any function f the following statements are equivalent.*
   (a) *f is polynomially length-bounded.*
   (b) *There exist a partial p-order A and functions $b, t \in$ FP such that, for all $x \in \Sigma^*$, $f(x) = \|\{z \mid b(x) <_A z <_A t(x)\}\|$.*
   (c) *There exist a total p-order A and functions $b, t \in$ FP such that, for all $x \in \Sigma^*$, $b(x) \leq_A t(x)$ and $f(x) = \|\{z \mid b(x) <_A z <_A t(x)\}\|$.*

Proof. The implications (1c) ⇒ (1b), (1b)⇒ (1a), (2c) ⇒ (2b), and (2b) ⇒ (2a) are obvious. We prove that (1a) ⇒ (1c) and (2a) ⇒ (2c).

It is easy to see that, for every polynomially length-bounded function $f : \Sigma^* \to \mathbb{N}$, there exist a set $B \subseteq \Sigma^* \times \Sigma^*$ and a strictly increasing polynomial p such that $f(x) = \|\{z \mid |z| = p(|x|) \wedge (x, z) \in B\}\|$. Note that we may choose B so that, for all $x \in \Sigma^*$, $(x, 0^{p(|x|)}) \notin B$ and $(x, 1^{p(|x|)}) \notin B$. If, in addition, $f \in$ #P, then B can be chosen from P.

We construct a total p-order A on Σ^* as follows. Generally, A will coincide with the lexicographical order on Σ^* except that, for every $x \in \Sigma^*$, the interval between $x0^{p(|x|)}$ and $x1^{p(|x|)}$ (inclusively) is ordered differently in the following way.

- First comes $x1^{p(|x|)}$.
- Next come the elements of $\{xz \mid |z| = p(|x|) \wedge (x, z) \in B\}$ in lexicographical order.
- Finally come the elements of $\{xz \mid |z| = p(|x|) \wedge (x, z) \notin B \wedge z \neq 1^{p(|x|)}\}$ in lexicographical order.

Note that $f(x) = \|\{w \mid x1^{p(|x|)} <_A w <_A x0^{p(|x|)}\}\|$. If, in addition, $B \in$ P, then $A \in$ P. □

We pass on a referee's comment that if one feels that putting $x0^{p(|x|)}$ before $x1^{p(|x|)}$ results in a more natural order, one could slightly tweak the order used and still have the proof go through.

5. Polynomial-time orders with efficient adjacency checks. We know from Theorem 4.1 that counting the size of intervals with respect to P-decidable partial p-orders that have polynomial-time computable boundaries computes some

function in #P. The situation changes if in addition we require each P-decidable partial p-order to have efficient adjacency checks.

DEFINITION 5.1. $\mathrm{IF_p}$ *(respectively,* $\mathrm{IF_t}$*) is the class of all functions* $f : \Sigma^* \to \mathbb{N}$ *for which there exist a partial (respectively, total) p-order* $A \in \mathrm{P}$ *having efficient adjacency checks and functions* $b, t \in \mathrm{FP}$*, such that, for every* $x \in \Sigma^*$*,* $f(x) = \|\{z \mid b(x) <_A z <_A t(x)\}\|$.

The following theorem places the classes $\mathrm{IF_t}$ and $\mathrm{IF_p}$ between two well-known complexity classes.

THEOREM 5.2. $\mathrm{FP} \subseteq \mathrm{IF_t} \subseteq \mathrm{IF_p} \subseteq \#\mathrm{P}$.

Proof. The second inclusion follows from the definitions of $\mathrm{IF_t}$ and $\mathrm{IF_p}$, and the third inclusion follows from Theorem 4.1. Thus, it remains to prove that $\mathrm{FP} \subseteq \mathrm{IF_t}$. For each $f \in \mathrm{FP}$, there exists a strictly increasing polynomial p such that $f(x) < 2^{p(|x|)} - 1$. For $x \in \Sigma^*$ and $i < 2^{p(|x|)}$, let $\mathrm{bin}(x, i)$ be the binary description of i having exactly $p(|x|)$ bits.

We construct a total p-order A on Σ^* as follows. Generally, A coincides with the lexicographical order on Σ^* except that, for every $x \in \Sigma^*$, the interval between $x0^{p(|x|)}$ and $x1^{p(|x|)}$ (inclusively) is ordered in the following way.

- First come the elements of $\{x\mathrm{bin}(x, i) \mid 0 \le i \le f(x)\}$ in lexicographical order.
- Next comes $x1^{p(|x|)}$.
- Finally come the elements of $\{x\mathrm{bin}(x, i) \mid f(x) < i < 2^{p(|x|)} - 1\}$ in lexicographical order.

Note that A is P-decidable, has efficient adjacency checks, and satisfies $f(x) = \|\{w \mid x0^{p(|x|)} <_A w <_A x1^{p(|x|)}\}\|$. □

What else can we say about the relationships between FP, $\mathrm{IF_t}$, $\mathrm{IF_p}$, and #P? We start by providing a characterization of $\mathrm{IF_p}$ based on an important subset of #P. Let $\mathrm{supp}(f)$ denote the support of f, i.e., $\mathrm{supp}(f) = \{x \mid f(x) \neq 0\}$.

THEOREM 5.3. $\mathrm{IF_p} = \{f \in \#\mathrm{P} \mid \mathrm{supp}(f) \in \mathrm{P}\}$.

Proof. Suppose that $f \in \mathrm{IF_p}$, via p-order $A \in \mathrm{P}$ having polynomial-time adjacency checks and boundary functions $b, t \in \mathrm{FP}$. Note that $\overline{\mathrm{supp}(f)} = \{x \mid b(x) \prec_A t(x) \vee b(x) \not\preceq_A t(x)\}$. Thus, since $A \in \mathrm{P}$ and $A_\prec \in \mathrm{P}$, it follows that $\overline{\mathrm{supp}(f)} \in \mathrm{P}$ and thus that $\mathrm{supp}(f) \in \mathrm{P}$. By Theorem 5.2, $f \in \#\mathrm{P}$. Therefore $\mathrm{IF_p} \subseteq \{f \in \#\mathrm{P} \mid \mathrm{supp}(f) \in \mathrm{P}\}$.

We now show that $\{f \in \#\mathrm{P} \mid \mathrm{supp}(f) \in \mathrm{P}\} \subseteq \mathrm{IF_p}$. Suppose $f \in \#\mathrm{P}$ and $\mathrm{supp}(f) \in \mathrm{P}$. Since $f \in \#\mathrm{P}$, there exist a set $B \subseteq \Sigma^* \times \Sigma^*$ from P and a strictly increasing polynomial p such that $f(x) = \|\{z \mid |z| = p(|x|) \wedge (x, z) \in B\}\|$.

We construct a partial p-order A on Σ^* as follows. Generally, A coincides with the lexicographical order on Σ^* except that, for every $x \in \Sigma^*$, the interval between $x0^{p(|x|)}00$ and $x1^{p(|x|)}11$ (inclusively) is ordered according to the following rules.

1. $x0^{p(|x|)}00 <_A x0^{p(|x|)}01 <_A x0^{p(|x|)}11$.
2. The elements from $\{xz10 \mid |z| = p(|x|) \wedge (x, z) \in B\}$ are pairwise incomparable, and all are between $x0^{p(|x|)}01$ and $x0^{p(|x|)}11$.
3. The elements from $\{xz10 \mid |z| = p(|x|) \wedge (x, z) \notin B\} \cup \{xz\sigma \mid |z| = p(|x|) \wedge z \neq 0^{p(|x|)} \wedge \sigma \in \{00, 01, 11\}\}$ are pairwise incomparable, and all are between $x0^{p(|x|)}00$ and $x0^{p(|x|)}01$.

Note that A is P-decidable and satisfies $f(x) = \|\{w \mid x0^{p(|x|)}01 <_A w <_A x0^{p(|x|)}11\}\|$. Define $b(x) =_{\mathrm{def}} x0^{p(|x|)}01$ and $t(x) =_{\mathrm{def}} x0^{p(|x|)}11$. For each x, we have by the construction of A that $b(x) \prec_A t(x)$ if and only if $f(x) = 0$. Since by assumption $\{x \mid f(x) > 0\} \in \mathrm{P}$ the set $\{x \mid b(x) \prec_A t(x)\}$ belongs to P. By our

construction, all other adjacency questions are very easily answered by the obvious, efficient test. So $A_{\prec} \in \mathrm{P}$. □

From this it follows that $\mathrm{IF_p}$ and #P coincide on Nonzero, defined as the set $\{f \mid (\forall x \in \Sigma^*)[f(x) > 0]\}$.

COROLLARY 5.4. $\mathrm{IF_p} \cap \mathrm{Nonzero} = \#\mathrm{P} \cap \mathrm{Nonzero}$.

In what follows, we will sometimes write 1 for the function class consisting of precisely the constant function $\lambda x.1$, and we will sometimes write $\mathcal{O}(1)$ for the function class consisting of precisely the functions $\lambda x.0$, $\lambda x.1$, $\lambda x.2$, ...

COROLLARY 5.5.

1. $\#\mathrm{P} \subseteq \mathrm{IF_p} - 1$.
2. $\#\mathrm{P} - \mathcal{O}(1) = \mathrm{IF_p} - \mathcal{O}(1)$.

From Theorem 5.2 and Corollary 5.5 we can conclude that $\mathrm{IF_p} \subseteq \mathrm{IF_p} - 1$, which is equivalent to saying that $\mathrm{IF_p}$ is closed under increment, i.e., for every $f \in \mathrm{IF_p}$, the function f' is also in $\mathrm{IF_p}$, where, for all $x \in \Sigma^*$, $f'(x) =_{\mathrm{def}} f(x) + 1$.

COROLLARY 5.6. *The class* $\mathrm{IF_p}$ *is closed under increment.*

Regarding $\mathrm{IF_t}$, we have the following theorem. Note that this theorem's second part says that the three function classes $\mathrm{IF_t}$, $\mathrm{IF_p}$, and #P are so closely related that in the presence of easy-to-compute subtractive postcomputation adjustments they become the same. Though it is not concerned with interval functions, we commend to the attention of the interested reader a beautiful paper by Ogihara et al. [OTTW96] that studies whether for #P postcomputation adjustments can annihilate even the effects of various operators.

THEOREM 5.7.

1. $\#\mathrm{P} \subseteq \mathrm{IF_t} - \mathrm{FP}$.
2. $\mathrm{IF_t} - \mathrm{FP} = \mathrm{IF_p} - \mathrm{FP} = \#\mathrm{P} - \mathrm{FP}$.

Proof. (1) For $f : \Sigma^* \to \mathbb{N}$ in #P, there exist a set $B \subseteq \Sigma^* \times \Sigma^*$ from P and a strictly increasing polynomial p such that $f(x) = \|\{z \mid |z| = p(|x|) \wedge (x, z) \in B\}\|$.

We construct a total p-order A on Σ^* as follows. Generally, A coincides with the lexicographical order on Σ^* except that, for every x, the interval between $x0^{p(|x|)+2}$ and $x1^{p(|x|)+2}$ (inclusively) is ordered differently in the following way.

- First come the elements of $\{xz00 \mid |z| = p(|x|)\}$ in lexicographical order.
- Next come the elements of $\{xz11 \mid |z| = p(|x|) \wedge (x, z) \in B\} \cup \{xz01 \mid |z| = p(|x|)\}$ in lexicographical order.
- Finally come the elements of $\{xz11 \mid |z| = p(|x|) \wedge (x, z) \notin B\} \cup \{xz10 \mid |z| = p(|x|)\}$ in lexicographical order.

Note that A is in P, has efficient adjacency checks, and satisfies $\|\{w \mid x1^{p(|x|)}00 <_A w <_A x0^{p(|x|)}10\}\| = f(x) + 2^{p(|x|)}$.

(2) This follows from Theorem 5.2 and part 1 of the present theorem. □

COROLLARY 5.8. $\mathrm{FP}^{\mathrm{IF_t}} = \mathrm{FP}^{\mathrm{IF_p}} = \mathrm{FP}^{\#\mathrm{P}}$.

The previous results indicate that the computational power of $\mathrm{IF_p}$ and $\mathrm{IF_t}$ are not far from the computational power of #P. Nonetheless, Theorem 5.10 shows that these classes cannot coincide unless P = NP. In the proof of Theorem 5.10 we will draw on the following lemma regarding the application of the $\exists$ operator to $\mathrm{IF_p}$ and $\mathrm{IF_t}$. Comparing Lemma 5.9 with Corollary 5.4 and taking into account that $\exists \cdot \#\mathrm{P} = \mathrm{NP}$, it turns out that it is precisely the possibility that $f(x) = 0$ that makes the classes #P and $\mathrm{IF_p}$ potentially differ.

LEMMA 5.9. $\exists \cdot \mathrm{IF_p} = \exists \cdot \mathrm{IF_t} = \mathrm{P}$.

Proof. For $L \in \exists \cdot \mathrm{IF_p}$ there exist a p-order $A \in \mathrm{P}$ having efficient adjacency checks and $b, t \in \mathrm{FP}$ such that, for all x, it holds that $x \in L \Leftrightarrow \|\{z \mid b(x) <_A z <_A t(x)\}\| > 0$.

Thus, for all $x \in \Sigma^*$, $x \in L \Leftrightarrow [b(x) \leq_A t(x)$ and $b(x) \not\leq_A t(x)]$, so $x \in L$ can be checked in polynomial time.

Choose $L \in \mathrm{P}$. Thus $\chi_L \in \mathrm{FP}$. By Theorem 5.2, $\chi_L \in \mathrm{IF_t}$, thus $L \in \exists \cdot \mathrm{IF_t}$. □

THEOREM 5.10. *The following statements are equivalent.*

1. $\mathrm{P} = \mathrm{NP}$.
2. $\mathrm{IF_p} = \#\mathrm{P}$.
3. $\mathrm{IF_t} = \#\mathrm{P}$.
4. *Every* P*-decidable partial p-order has efficient adjacency checks.*
5. *Every* P*-decidable total p-order has efficient adjacency checks.*

Proof. (1) ⇒ (4) follows from Corollary 3.2.1. (4) ⇒ (5) is immediate from the definitions. (5) ⇒ (3) follows from Theorem 4.1.1. (3) ⇒ (2) follows from Theorem 5.2. To see that (2) ⇒ (1), if $\mathrm{IF_p} = \#\mathrm{P}$ then $\exists \cdot \mathrm{IF_p} = \exists \cdot \#\mathrm{P}$. By Lemma 5.9 and Proposition 2.2.3 we have $\mathrm{P} = \mathrm{NP}$. □

We know from Theorem 5.2 that $\mathrm{FP} \subseteq \mathrm{IF_t}$. However, if $\mathrm{IF_t} \subseteq \mathrm{FP}$ or even $\mathrm{IF_t} \subseteq \mathrm{UPSV_t}$, then severe consequences follow.

THEOREM 5.11.

1. $\mathrm{FP} = \mathrm{IF_t}$ *if and only if* $\mathrm{P} = \mathrm{PP}$.
2. $\mathrm{IF_t} \subseteq \mathrm{UPSV_t}$ *if and only if* $\mathrm{UP} = \mathrm{PP}$.
3. $\mathrm{UPSV_t} \subseteq \mathrm{IF_p}$ *if and only if* $\mathrm{P} = \mathrm{UP} \cap \mathrm{coUP}$.

Proof. For items (1) and (2) we consider the left-to-right direction first. From Theorem 5.7 and Proposition 2.2, we can conclude under the assumption $\mathrm{FP} = \mathrm{IF_t}$ that $\mathrm{PP} = \exists \cdot (\#\mathrm{P} - \mathrm{FP}) = \exists \cdot (\mathrm{IF_t} - \mathrm{FP}) = \exists \cdot (\mathrm{FP} - \mathrm{FP}) = \mathrm{P}$ and we can conclude under the assumption $\mathrm{IF_p} \subseteq \mathrm{UPSV_t}$ that $\mathrm{PP} = \exists \cdot (\#\mathrm{P} - \mathrm{FP}) = \exists \cdot (\mathrm{IF_t} - \mathrm{FP}) \subseteq \exists \cdot (\mathrm{UPSV}_t - \mathrm{FP}) = \mathrm{UP} \cap \mathrm{coUP}$. For the right-to-left directions, if $\mathrm{P} = \mathrm{PP}$, then $\mathrm{IF_t} \subseteq \#\mathrm{P} \subseteq \mathrm{FP}^{\#\mathrm{P}} = \mathrm{FP}^{\mathrm{PP}} = \mathrm{FP}$. Thus, $\mathrm{IF_t} = \mathrm{FP}$. If $\mathrm{UP} = \mathrm{PP}$, then $\mathrm{IF_t} \subseteq \#\mathrm{P} \subseteq \mathrm{FP}^{\#\mathrm{P}} = \mathrm{FP}^{\mathrm{PP}} = \mathrm{FP}^{\mathrm{UP} \cap \mathrm{coUP}} = \mathrm{UPSV_t}$.

For item (3), from $\mathrm{UPSV}_t \subseteq \mathrm{IF_p}$, Proposition 2.2, and Lemma 5.9 it follows that $\mathrm{UP} \cap \mathrm{coUP} = \exists \cdot \mathrm{UPSV}_t \subseteq \exists \cdot \mathrm{IF_p} = \mathrm{P}$. For the right-to-left direction, by Proposition 2.1.1, $\mathrm{P} = \mathrm{UP} \cap \mathrm{coUP}$ implies $\mathrm{UPSV_t} = \mathrm{FP}$. So, by Theorem 5.2, $\mathrm{P} = \mathrm{UP} \cap \mathrm{coUP}$ implies $\mathrm{UPSV_t} \subseteq \mathrm{IF_p}$ (and even $\mathrm{UPSV_t} \subseteq \mathrm{IF_t}$). □

In contrast to Theorem 5.11.3, when restricted to strictly positive functions the class $\mathrm{UPSV_t}$ is even included in $\mathrm{IF_t}$.

THEOREM 5.12. $\mathrm{UPSV_t} \cap \mathrm{Nonzero} \subseteq \mathrm{IF_t} \cap \mathrm{Nonzero}$.

A proof of Theorem 5.12 is in the technical report version [HHKW05] of this paper. Since $\mathrm{UPSV_t}$ is closed under increment, Theorem 5.12 yields the following corollary.

COROLLARY 5.13. $\mathrm{UPSV_t} \subseteq \mathrm{IF_t} - 1$.

Corollary 5.6 showed that the class $\mathrm{IF_p}$ is closed under increment. This is also true for the class $\mathrm{IF_t}$.

THEOREM 5.14. *The class* $\mathrm{IF_t}$ *is closed under increment.*

Proof. For $f \in \mathrm{IF_t}$ there exist a P-decidable p-order A on Σ^* with efficient adjacency checks and functions $b, t \in \mathrm{FP}$ such that, for all $x \in \Sigma^*$, $f(x) = \|\{w \mid b(x) <_A w <_A t(x)\}\|$. Without loss of generality we may require that $b(x) \leq_A t(x)$, since on inputs not satisfying that we may modify $t(x)$ to output $b(x)$. Let p be a strictly increasing polynomial such that, for all $y \in \Sigma^*$ satisfying $y \leq_A t(x)$, $|y| < p(|x|)$.

We construct a total p-order A' on Σ^* as follows. Generally, A' coincides with the lexicographical order on Σ^* except that, for every $x \in \Sigma^*$, the interval between $x0^{p(|x|)+2}$ and $x1^{p(|x|)+2}$ (inclusively) is ordered in the following way.

- First comes $x0^{p(|x|)+2}$.

- Next come the elements of $D_x =_{\text{def}} \{x0^{p(|x|)-|z|}1z0 \mid b(x) \leq_A z \leq_A t(x)\}$, for which we set $x0^{p(|x|)-|y|}1y0 \leq_{A'} x0^{p(|x|)-|z|}1z0$ if and only if $y \leq_A z$.
- Finally come the elements of $\{xu \mid |u| = p(|x|) + 2\} - (D_x \cup \{0^{p(|x|)+2}\})$ in lexicographical order.

Note that A' is P-decidable, that it has efficient adjacency checks, and that $f(x)+1 = \|\{w \mid x0^{p(|x|)+2} <_{A'} w <_{A'} x0^{p(|x|)-|t(x)|}1t(x)0\}\|$. □

COROLLARY 5.15. $\text{IF}_\text{t} \subseteq \text{IF}_\text{t}$ - 1.

Although the statement "$\text{UPSV}_\text{t} = \text{IF}_\text{t}$" is not likely to be true (see Theorem 5.11), for the case of strictly positive, polynomially bounded functions the analogous statement holds. We define PolyBounded $=_{\text{def}} \{f \mid (\exists \text{ polynomial } p)(\forall x)[f(x) \leq p(|x|)]\}$.

THEOREM 5.16.

1. $\text{IF}_\text{t} \cap \text{PolyBounded} \subseteq \text{UPSV}_\text{t} \cap \text{PolyBounded}$.
2. $\text{IF}_\text{t} \cap \text{PolyBounded} \cap \text{Nonzero} = \text{UPSV}_\text{t} \cap \text{PolyBounded} \cap \text{Nonzero}$.
3. $\text{UPSV}_\text{t} \cap \text{PolyBounded} \subseteq \text{IF}_\text{p} \cap \text{PolyBounded}$ *if and only if* $\text{P} = \text{UP} \cap \text{coUP}$.

A proof of Theorem 5.16 is in the technical report version [HHKW05] of this paper.

From Theorem 4.1 we know that total p-orders that are efficiently decidable and partial p-orders that are efficiently decidable describe the same class of functions in our setting (namely #P). If we consider p-orders that additionally have efficient adjacency checks, then the analogous confluence of total and partial does not hold unless an unexpected complexity class collapse occurs.

THEOREM 5.17. *If* $\text{IF}_\text{t} = \text{IF}_\text{p}$, *then* $\text{UP} = \text{PH}$.

Proof. Assume that $\text{IF}_\text{t} = \text{IF}_\text{p}$. We show that coNP $\subseteq$ UP (which is equivalent to the statement UP = PH). Let $L \in \text{coNP}$, i.e., there is a function $f \in \#\text{P}$ such that, for all $x \in \Sigma^*$, $x \in L \Leftrightarrow f(x) = 0$. Consider the function f', where $f'(x) =_{\text{def}} f(x) + 1$. Thus $x \in L \Leftrightarrow f'(x) = 1$ and, since #P is closed under increment, we conclude that $f' \in \#\text{P} \cap \text{Nonzero} = \text{IF}_\text{p} \cap \text{Nonzero} = \text{IF}_\text{t} \cap \text{Nonzero}$. Thus, there exist a total p-order $A \in \text{P}$ with efficient adjacency checks and functions $b, t \in \text{FP}$ such that $f'(x) = \|\{z \mid b(x) <_A z <_A t(x)\}\|$. Let q be a polynomial such that $(x, y) \in A$ implies $|x| \leq q(|y|)$. Define M to be a machine that, on input $x \in \Sigma^*$, nondeterministically guesses z such that $|z| \leq q(|t(x)|)$ and checks whether $b(x) \prec_A z \prec_A t(x)$. Clearly, M runs in polynomial time (since A has efficient adjacency checks) and always has at most one accepting path (since A is a total p-ordering and we are doing two adjacency checks in our test). Moreover, $x \in L$ if and only if M on x has an accepting computation path. Thus, $L \in \text{UP}$. □

6. Arbitrary orders with efficient adjacency checks. In the previous section, we studied polynomial-time-decidable p-orders having efficient adjacency checks. We showed that the classes defined by interval size functions over such orders, IF_p and IF_t, are very close to #P. In the present section, we consider what happens when we do not insist on polynomial-time decidability for the order but still require efficient adjacency checks. Section 6.1 presents our results on this. Due to its complexity and length, the proof of one key claim of that section, Lemma 6.5, is presented separately as section 6.2.

6.1. Results on arbitrary orders with efficient adjacency checks. In this section, we study p-orders that have efficient adjacency checks but that are not required to be polynomial-time decidable. We define two classes to capture this behavior.

DEFINITION 6.1. *The class* $\mathrm{IF}_{\mathrm{p}}^{*}$ *(respectively,* $\mathrm{IF}_{\mathrm{t}}^{*}$*) is the set of all functions* $f : \Sigma^* \to \mathbb{N}$ *for which there exist a partial (respectively, total) p-order* A *having efficient adjacency checks and functions* $b, t \in \mathrm{FP}$ *such that, for every* $x \in \Sigma^*$, $f(x) = \|\{z \mid b(x) <_A z <_A t(x)\}\|$.

We have the following inclusions between classes of interval size functions and other complexity classes of functions.

PROPOSITION 6.2. $\mathrm{IF}_{\mathrm{t}} \subseteq \mathrm{IF}_{\mathrm{t}}^{*} \subseteq \mathrm{IF}_{\mathrm{p}}^{*} \subseteq \mathrm{FPSPACE(poly)}$ *and* $\mathrm{IF}_{\mathrm{t}} \subseteq \mathrm{IF}_{\mathrm{p}} \subseteq \mathrm{IF}_{\mathrm{p}}^{*} \cap \#\mathrm{P} \subseteq \#\mathrm{P} \subseteq \mathrm{FPSPACE(poly)}$.

Proof. The only inclusion that is nontrivial is $\mathrm{IF}_{\mathrm{p}}^{*} \subseteq \mathrm{FPSPACE(poly)}$. Let f be in $\mathrm{IF}_{\mathrm{p}}^{*}$ via a partial p-order A having efficient adjacency checks and functions $b, t \in \mathrm{FP}$. Let p be a polynomial such that, for all $x, y \in \Sigma^*$, $(x, y) \in A$ implies $|x| \leq p(|y|)$. From Proposition 3.1 we know that A is in PSPACE. Thus, there is a polynomial-space Turing machine M that, for any input $x \in \Sigma^*$, counts by brute force how many strings z of length at most $p(|t(x)|)$ satisfy $b(x) <_A z <_A t(x)$. We may thus conclude that f is in FPSPACE(poly). □

The main results of this section show that the computational powers of $\mathrm{IF}_{\mathrm{p}}^{*}$ and $\mathrm{IF}_{\mathrm{t}}^{*}$ are close to the computational power of FPSPACE(poly). In fact, within the flexibility of the simple postcomputation adjustment of subtracting polynomial-time computable functions, these three classes become the same.

THEOREM 6.3. $\mathrm{IF}_{\mathrm{t}}^{*} \text{ - } \mathrm{FP} = \mathrm{IF}_{\mathrm{p}}^{*} \text{ - } \mathrm{FP} = \mathrm{FPSPACE(poly)} \text{ - } \mathrm{FP}$.

THEOREM 6.4. $\exists \cdot \mathrm{IF}_{\mathrm{t}}^{*} = \exists \cdot \mathrm{IF}_{\mathrm{p}}^{*} = \mathrm{PSPACE}$.

Theorem 6.4 can be interpreted to say something a bit surprising about the complexity of reachability, namely, that in a certain sense reachability checking in succinctly specified chains is PSPACE-complete. To see this, we reason as follows. There exist PSPACE-complete problems. So from that and Theorem 6.4, in light of Definition 6.1, we have that there are a total p-order A and functions $b, t \in \mathrm{FP}$ such that the set $\{x \mid f(x) > 0\}$ is PSPACE-complete, where $f(x) = \|\{z \mid b(x) <_A z <_A t(x)\}\|$. To interpret this all in terms of reachability in a succinctly specified chain, we can view A as specifying an infinite chain such that a has an edge to b precisely if b is right-adjacent to a. Testing whether $f(x) > 0$ is asking "Is it the case that both (a) $t(x)$ is not right-adjacent to $b(x)$, and (b) there is a path from $b(x)$ to $t(x)$ within the chain?" Note that the "(a)" part of this test is a polynomial-time test, so the complexity is coming from the "(b)" part. Thus, in the sense just mentioned, Theorem 6.4 shows the unexpected result that even for succinctly, simply specified chains, reachability testing where the "to" and "from" elements are indirectly polynomial-time specified by the input is PSPACE-complete. However, the problem just discussed, where the "to" and "from" elements are determined by polynomial-time computable functions of the input, can easily be seen—being careful of course about condition "(a)" from above—to polynomial-time many-one reduce to the more flexible case in which the "to" and "from" elements are the input. And so we have that there exists a set $A \subseteq \mathbb{N}^2$, $A \in \mathrm{P}$, such that the directed graph $(\mathbb{N}, A)$ is a chain and its reachability problem,

$$\{(x, y) \mid x, y \in \mathbb{N} \text{ and } y \text{ is reachable from } x \text{ in } (\mathbb{N}, A)\},$$

is PSPACE-complete. Even in light of the existing results showing that many problems about succinctly specified graphs are hard, and often PSPACE-complete (see [Wag84, Wag86, PY86, GW83], but for contrast see also [HHW05, NT05]), this result, which speaks to succinctly specified *chains*, seems surprising. (These comments all regard the "total" part of Theorem 6.4. The "partial" part of Theorem 6.4 implies the

analogous but far less surprising claim for succinctly specified graphs (as opposed to succinctly specified chains), and in fact would give one an alternate way of seeing the known (see [Wag84, Wag86, PY86, GW83]) result that the graph reachability problem in succinctly specified graphs is PSPACE-complete. By the way, PSPACE-completeness is known from those earlier works to still hold even when the graphs are restricted to outdegree one (see [Wag84, Wag86, PY86, GW83], and the discussion in [Tan01]), which brings one somewhat closer to the chains cases mentioned above.)

Theorems 6.3 and 6.4 follow immediately from Proposition 6.2 and the following lemma, whose proof is deferred to section 6.2.

LEMMA 6.5. *For each* $f \in \mathrm{FPSPACE(poly)}$*, there exist a total p-order* A *having efficient adjacency checks and polynomial-time computable functions* $s : \mathbb{N} \to \mathbb{N}$*,* $b : \Sigma^* \to \Sigma^*$*,* $b' : \Sigma^* \to \Sigma^*$*, and* $t : \Sigma^* \to \Sigma^*$ *such that, for all* $x \in \Sigma^*$*,*

1. s *is polynomially bounded,*
2. $\|\{z \mid b(x) <_A z <_A t(x)\}\| = 2^{2s(|x|)+1} + f(x) - 2$*, and*
3. $\|\{z \mid b'(x) <_A z <_A t(x)\}\| > 0$ *if and only if* $f(x) = 1$.

As a consequence of Theorems 6.3 and 6.4, we obtain characterizations for the class FPSPACE(poly) in terms of $\mathrm{IF}_{\mathrm{t}}^*$. For classes $\mathcal{F}$ and $\mathcal{G}$ of functions from Σ^* to $\mathbb{N}$, let $\mathcal{F} \ominus \mathcal{G}$ denote the class of all total, nonnegative functions in $\mathcal{F}$ - $\mathcal{G}$, i.e., the class of all total functions h for which there exist total functions $f \in \mathcal{F}$ and $g \in \mathcal{G}$ such that, for all $x \in \Sigma^*$, $f(x) \geq g(x)$ and $h(x) = f(x) - g(x)$.

COROLLARY 6.6.

1. $\mathrm{FPSPACE(poly)} = \mathrm{IF}_{\mathrm{t}}^* \ominus \mathrm{FP} = \mathrm{FP}^{\mathrm{IF}_{\mathrm{t}}^*} = \mathrm{FP}^{\exists \cdot \mathrm{IF}_{\mathrm{t}}^*}$.
2. $\mathrm{FPSPACE(poly)} = \mathrm{IF}_{\mathrm{p}}^* \ominus \mathrm{FP} = \mathrm{FP}^{\mathrm{IF}_{\mathrm{p}}^*} = \mathrm{FP}^{\exists \cdot \mathrm{IF}_{\mathrm{p}}^*}$.

Proof. Regarding part 1, by Theorem 6.3, Proposition 6.2, and Theorem 6.4 we have $\mathrm{FPSPACE(poly)} \subseteq \mathrm{IF}_{\mathrm{t}}^* \ominus \mathrm{FP} \subseteq \mathrm{FP}^{\mathrm{IF}_{\mathrm{t}}^*} \subseteq \mathrm{FP}^{\mathrm{FPSPACE(poly)}} \subseteq \mathrm{FP}^{\mathrm{PSPACE}} \subseteq \mathrm{FP}^{\exists \cdot \mathrm{IF}_{\mathrm{t}}^*} \subseteq \mathrm{FP}^{\mathrm{PSPACE}} \subseteq \mathrm{FPSPACE(poly)}$. Part 2 holds by the same inclusion chain applied to $\mathrm{IF}_{\mathrm{p}}^*$. □

Though Theorem 6.3 shows that $\mathrm{IF}_{\mathrm{t}}^*$ is almost as powerful as FPSPACE(poly), the following theorem shows that it is unlikely that $\mathrm{IF}_{\mathrm{t}}^*$ actually coincides with FPSPACE(poly).

THEOREM 6.7. *If* $\mathrm{FPSPACE(poly)} \subseteq \mathrm{IF}_{\mathrm{p}}^*$*, then* $\mathrm{UP} = \mathrm{PSPACE}$.

Proof. Suppose that $\mathrm{FPSPACE(poly)} \subseteq \mathrm{IF}_{\mathrm{p}}^*$. Let $L \in \mathrm{PSPACE}$. Then its characteristic function χ_L is in FPSPACE(poly), and by hypothesis $\chi_L \in \mathrm{IF}_{\mathrm{p}}^*$ via some partial p-order A having efficient adjacency checks, some polynomial p such that $(x, y) \in A$ implies $|x| \leq p(|y|)$, and functions $b, t \in \mathrm{FP}$ such that $\chi_L(x) = \|\{z \mid b(x) <_A z <_A t(x)\}\|$. Note that $L = \{x \mid (\exists z)[|z| \leq p(|t(x)|) \wedge b(x) \prec_A z \prec_A t(x)]\}$. Thus, keeping in mind that $(\forall x)[\chi_L(x) \leq 1]$, we have $L \in \mathrm{UP}$. □

From Theorems 6.3 and 6.4, if $\mathrm{IF}_{\mathrm{t}}^* = \mathrm{IF}_{\mathrm{t}}$ or $\mathrm{IF}_{\mathrm{t}}^* \subseteq \#\mathrm{P}$ - FP, then strong consequences follow, as the following two corollaries show.

COROLLARY 6.8. *The following statements are equivalent.*

1. $\mathrm{P} = \mathrm{PSPACE}$.
2. $\mathrm{IF}_{\mathrm{p}} = \mathrm{IF}_{\mathrm{p}}^*$.
3. $\mathrm{IF}_{\mathrm{t}} = \mathrm{IF}_{\mathrm{t}}^*$.
4. *Every partial p-order with efficient adjacency checks is* P*-decidable.*
5. *Every total p-order with efficient adjacency checks is* P*-decidable.*

Proof. (1) ⇒ (4) is just Corollary 3.2.2. (4) ⇒ (5) is trivial. (4) ⇒ (2) and (5) ⇒ (3) follow from the definitions of IF_{p}, $\mathrm{IF}_{\mathrm{p}}^*$, IF_{t}, and $\mathrm{IF}_{\mathrm{t}}^*$. By Theorem 6.4 and Lemma 5.9, (2) implies $\mathrm{PSPACE} = \exists \cdot \mathrm{IF}_{\mathrm{p}}^* = \exists \cdot \mathrm{IF}_{\mathrm{p}} = \mathrm{P}$ and so implies (1). Similarly, (3) implies $\mathrm{PSPACE} = \exists \cdot \mathrm{IF}_{\mathrm{t}}^* = \exists \cdot \mathrm{IF}_{\mathrm{t}} = \mathrm{P}$ and so implies (1). □

COROLLARY 6.9.

1. *If* $\mathrm{IF}_{\mathrm{t}}^* \subseteq \#\mathrm{P} - \mathrm{FP}$, *then* $\mathrm{SPP} = \mathrm{PSPACE}$.
2. *If* $\mathrm{IF}_{\mathrm{t}}^* \subseteq \#\mathrm{P}$, *then* $\mathrm{NP} = \mathrm{SPP} = \mathrm{PSPACE}$.

Proof. (1): For $L \in \mathrm{PSPACE}$, $\chi_L \in \mathrm{FPSPACE(poly)}$. By Proposition 6.2, Theorem 6.3, and our assumption, $\chi_L \in \#\mathrm{P} - \mathrm{FP}$. Thus, $L \in \mathrm{SPP}$.

(2): From Theorem 6.4 and our hypothesis, we obtain $\mathrm{PSPACE} \subseteq \exists \cdot \#\mathrm{P} = \mathrm{NP}$. Combining this with the first part of this theorem we have $\mathrm{SPP} = \mathrm{NP} = \mathrm{PSPACE}$. □

The next result is analogous to results regarding the potential equality of IF_{t} and IF_{p}.

THEOREM 6.10. *If* $\mathrm{IF}_{\mathrm{t}}^* = \mathrm{IF}_{\mathrm{p}}^*$, *then* $\mathrm{UP} = \mathrm{PH}$.

Proof. The proof follows the proof of Theorem 5.17, except that, for the function there called f', we now conclude that $f' \in \#\mathrm{P} \cap \mathrm{Nonzero} = \mathrm{IF}_{\mathrm{p}} \cap \mathrm{Nonzero} \subseteq \mathrm{IF}_{\mathrm{p}}^* \cap \mathrm{Nonzero} = \mathrm{IF}_{\mathrm{t}}^* \cap \mathrm{Nonzero}$. This approach works because the hypothesis $f' \in \mathrm{IF}_{\mathrm{t}}^*$ can be exploited in the same way as the hypothesis $f' \in \mathrm{IF}_{\mathrm{t}}$ was exploited in the proof of Theorem 5.17. This is because in the proof of Theorem 5.17 the P-decidability of the total p-order underlying $f' \in \mathrm{IF}_{\mathrm{t}}$ was not even used. □

Figure 1 summarizes the results we have obtained regarding the inclusion structure of our classes. Although we have not proven consequences of collapses other than those drawn in the figure, we conjecture that the inclusions in the figure are all one can prove without assuming unexpected collapses of complexity classes.

6.2. Proof of Lemma 6.5. The goal of this section is to prove Lemma 6.5. For convenience, we repeat its statement here.

Lemma 6.5. *For each* $f \in \mathrm{FPSPACE(poly)}$, *there exist a total p-order* A *having efficient adjacency checks and polynomial-time computable functions* $s : \mathbb{N} \to \mathbb{N}$, $b : \Sigma^* \to \Sigma^*$, $b' : \Sigma^* \to \Sigma^*$, *and* $t : \Sigma^* \to \Sigma^*$ *such that, for all* $x \in \Sigma^*$,

1. s *is polynomially bounded,*
2. $\|\{z \mid b(x) <_A z <_A t(x)\}\| = 2^{2s(|x|)+1} + f(x) - 2$, *and*
3. $\|\{z \mid b'(x) <_A z <_A t(x)\}\| > 0$ *if and only if* $f(x) = 1$.

Constructing the p-order A mentioned in Lemma 6.5 is, compared to the other p-orders described in this paper, more technically involved. Before we prove Lemma 6.5, we will show, for any $f \in \mathrm{FPSPACE(poly)}$, how to construct A based on the behavior of a Turing machine that computes f. We will then prove Lemma 6.5 by showing that A has all the properties claimed by the lemma.

Our approach is based on the fact that, for any deterministic Turing machine M and any halting configuration c of M, the set of all configurations of M that lead to c form a tree having c as its root and as its edges every pair of configurations where in one time step M moves from one configuration to the other. We base the order of A on a traversal of these trees. Each argument to A encodes either nonsense or an input to M, a configuration of M whose length is bounded in the space bound on $M(x)$, a direction relative to the traversal of the tree to which the given configuration belongs, a guess as to what the eventual halting configuration of M will be when run from the given configuration, and some additional information we describe later. These arguments effectively yield multiple copies of each tree. The order A organizes the arguments in such a way that, in the case of Lemma 6.5.2 (Lemma 6.5.3 uses basically the same approach) $b(x)$ and $t(x)$ delimit an interval that has two elements for every configuration within the space bound imposed by $M(x)$, and $f(x)$ additional elements.

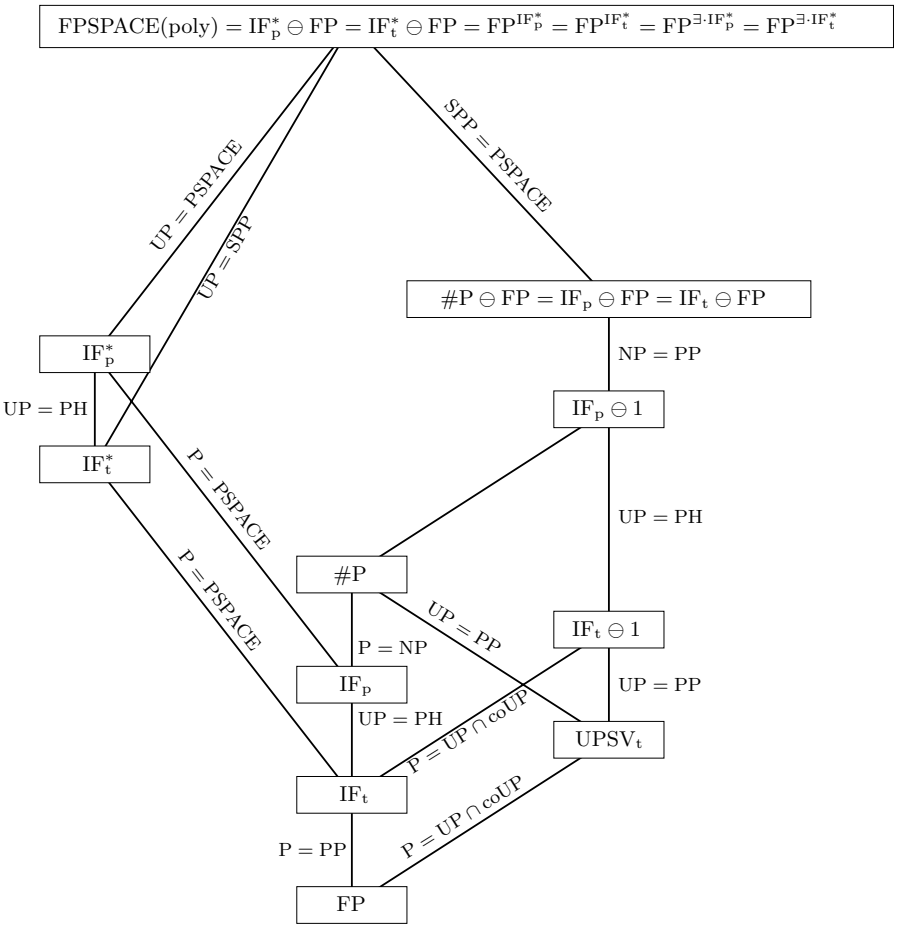


FIG. 1. *The landscape of interval size function classes and related function classes. An equation E on the edge between the function classes $\mathcal{F}_1$ and $\mathcal{F}_2$ means that $\mathcal{F}_1 = \mathcal{F}_2$ implies E. The edge equations that are not immediate consequences of the results of this paper are well-known or easy to see. Since* FP*, which forms the base of this containment tower, is of type $\Sigma^* \to \mathbb{N}$, the fact that in the above figure we use "$\ominus$" rather than "$-$" is of no consequence.*

The key "trick" in this construction is the sequence of "guess bits" each argument to A has, which allow us to send along each tree "messages" that link each configuration to its actual halting configuration. We "pad" A, near arguments containing initial configurations, with as many additional arguments as the guess bits predict. We of course do not know at this point (i.e., "near" an initial configuration) if the guess is correct. However, by the end of the computation we do know. So we simply place together in A all trees corresponding to correct guesses and set, in the case of Lemma 6.5.2, $b(x)$ and $t(x)$ to the boundaries of this interval (which "squeezes out" from the desired interval all incorrect guesses). This yields the claimed results.

We will construct A in five phases, described as follows.

1. **Fixing the computational model**. We will base A on a Turing machine M that computes f in a natural but somewhat nonstandard way. The benefit of using M rather than an arbitrary FPSPACE(poly) Turing machine for f is that it will be easier to work with binary encodings of the configurations

of M and the actions of M than with those of an arbitrary FPSPACE(poly) Turing machine for f.

2. **Fixing the encoding**. We will base A on binary encodings of the configurations of M, which we call *enhanced instantaneous descriptions*. Our encodings are like standard instantaneous descriptions (IDs) [HMU01] but differ in three crucial ways. First, our encodings are actual binary strings rather than sequences of abstract symbols. Second, we use different syntax (which we describe below). Finally, our descriptions contain more information than is actually needed to describe a configuration of M at an instant in time. This additional information is never accessed by M, so its presence in the encodings does not affect the performance of M. At the same time, its presence will greatly aid us in constructing A.
3. **Building trees**. For some appropriate polynomial s, we will, for each $x \in \Sigma^*$, define a tree whose nodes are enhanced instantaneous descriptions of M and whose edges are based on the next move function of M. This tree will have a subtree T_x having exactly $2^{2s(|x|)}$ nodes.
4. **Traversing the trees**. We will associate multiple strings with each node in the tree described above (by padding the labels of the nodes) in such a way that $f(x) + 2$ strings are associated with one of the nodes in T_x and two strings are associated with each of the remaining $2^{2s(|x|)} - 1$ nodes in T_x. We will then define a total, one-to-one, polynomial-time computable function D_M over these strings in such a way that D_M, applied repeatedly to some appropriate starting point, represents a traversal of the tree such that the traversal visits each of these strings once, i.e., from a particular one of the strings z associated with the root of the tree, for each string y associated with some node of the tree there is an integer $i \in \mathbb{N}$ such that $D_M^{(i)}(z) = y$, where $D_M^{(0)}(z) = z$, and, for each $i \in \{1, 2, 3, \dots\}$, $D_M^{(i)}(z) = D_M(D_M^{(i-1)}(z))$. Moreover, for strings w and y, $D_M(w) = y$ only if the nodes associated with w and y are related (i.e., parent/child, sibling, or identical nodes).
5. **Constructing A**. We will base A on D_M. For example, A crucially will have the property that if w and z are two of the strings described in Phase 4, then $w \prec_A z$ if and only if $z = D_M(w)$. Note there will also be many strings on which D_M is not defined that will nonetheless have to be accounted for. Through careful encoding at each phase in the construction, it will be easy to account for these strings in such a way that A has all the properties we desire.

After we handle these five phases, we will prove Lemma 6.5. We now proceed with the construction. Please note that, due to the length of this construction, we overload certain variables. For instance, the variable t denotes both a function over strings and over natural numbers, and has distinct semantics in each case. Over strings it is the function that determines the "bottom" of an interval (i.e., it is used as it typically is throughout this paper), and over the natural numbers it bounds the amount of space needed for part of the encodings we use.

Phase 1: Fixing the computational model. Let $M = (Q, \Sigma, \Gamma, \delta, B, q_0, F)$ be a Turing machine that computes f, where

- Q is the set of *state symbols*,
- $\Sigma = \{0, 1\}$ is the set of *input symbols*,
- B is the *blank symbol*,
- $\Gamma \supseteq \{0, 1, B\}$ is the set of allowable *tape symbols*,

- δ is the *next move function*, i.e., a mapping from $Q \times \Gamma$ to $Q \times \Gamma \times \{-1, 1\}$,
- q_0 is the *start state*, and
- $F \subseteq Q$ is the set of *final states*.

We assume that M has the following properties.

- For some $m \in \mathbb{N}$, $\|Q\| = \|\Gamma\| = 2^m$ (any Turing machine not having this property can be turned into one having this property by adding extra "dummy" states and symbols to its current sets of state and tape symbols, respectively). Since $\Gamma \supseteq \{0, 1, B\}$, $m \geq 2$.
- F contains a single element, q_f, and $q_0 \neq q_f$.
- M has a single, one-way infinite tape (a standard PSPACE(poly) Turing machine would have distinct input, output, and work tapes). On no input x does a true run of M move off the left end of the tape. (One way to ensure that M has this latter property is to include the symbols, 0^e, 1^e, and B^e in Γ. These symbols will be used, exactly on the leftmost cell of the tape, as replacements for 0, 1, and B. We can then construct M so that it is in its start state just once, namely at the beginning of the run, and that, from its start state, it always replaces the then-current symbol (which, in a true run, will always be located in the leftmost tape cell and will be either 0, 1, or B) not with whatever symbol it would normally write during that step but rather with the appropriate analogue among 0^e, 1^e, and B^e. Similarly, our machines can be forced to be such that they attempt to ensure that at all future times this left-marking is preserved, i.e., a $0^e/1^e/B^e$-marker square may be changed during the run but just among 0^e, 1^e, and B^e, as appropriate. A Turing machine constructed in this way can, on any true run, determine when it is about to (were it to mindlessly perform the simulation of the underlying machine) move off the left end, and can indeed handle—without itself running off the left end and in a fashion that is consistent in effect with whatever standard behavior (typically either rejection or "bouncing off" the left end) we in our notion of Turing machines associate with attempting to go off the left end—the left-end move-off that was about to happen.)
- δ on input $(q, r) \in Q \times \Gamma$ is defined if and only if $(q, r) \notin \{q_f\} \times \Gamma$.
- For all $r \in \Gamma$ and all $i \in \{-1, 1\}$, (q_0, r, i) is not in the image of δ. (That is, nothing moves *to* the start state.)
- For all $x \in \Sigma^*$, M on input x halts with $y \in \Sigma^*$ written on its $|y|$ leftmost tape cells, where y is the shortest binary representation of $f(x)$ (i.e., no leading zeros, unless $f(x) = 0$), and with every other tape cell containing the blank symbol.
- There is a strictly increasing polynomial p such that, on each input $x \in \Sigma^*$, M uses, at most, $p(|x|)$ tape cells and $p(|x|) > 0$.

Phase 2: Fixing the encoding. We now describe the binary encoding we use to describe the configurations of M. Figure 2 provides an overview of this phase of the construction. Let $\varphi : Q \to \{0, 1\}^m$ be a total bijection (recall that $\|Q\| = 2^m$ and $m \geq 2$) such that $\varphi(q_0) = 0^m$ and $\varphi(q_f) = 1^m$. The function φ^{-1} denotes the unique total bijection from $\{0, 1\}^m$ to Q that inverts φ. Let $\theta : \Gamma \to \{0, 1\}^m$ be a total bijection (recall that $\|\Gamma\| = 2^m$ and $m \geq 2$) such that $\theta(B) = 0^m$, $\theta(0) = 1^{m-1}0$, and $\theta(1) = 1^m$. Define $\hat{\theta} : \Gamma^* \to (\{0, 1\}^m)^*$ recursively as $\hat{\theta}(\epsilon) = \epsilon$, and, for all $y \in \Gamma$ and $w \in \Gamma^*$, $\hat{\theta}(wy) = \hat{\theta}(w)\theta(y)$. Since $\hat{\theta}$ is also a bijection, we use $\hat{\theta}^{-1}$ to denote the unique total bijection from $(\{0, 1\}^m)^*$ to Γ^* that inverts $\hat{\theta}$.

We define the "partially encoded" next move function $\delta' : \{0, 1\}^m \times \{0, 1\}^m \to$

FIG. 2. *A brief comparison between standard instantaneous descriptions (IDs) and the enhanced IDs we use. Before the computation step illustrated above, the tape head is at cell $b+1$ and the machine is in state q. Afterwards, the head is at cell b and the machine is in state q'. The symbol μ represents the next move function. In standard IDs, the state q appears immediately before the tape cell that the head is currently visiting (e.g., in the case illustrated above, cell $b+1$ before the move and b afterwards). Our enhanced IDs contain additional strings: x, c, and w. The string x encodes the input to the Turing machine, c encodes the number of computation steps the Turing machine has performed so far, and w is the position of the tape head. The state string remains in the same place throughout the computation, and instead w is updated with the position of the tape head. Thus, w encodes the number $b+1$ (i.e., the position of the tape head before the computation step), and w' encodes b (i.e., the position of the tape head after the computation step). The strings c and c' also represent numbers, where the number encoded by c' is one greater than the number encoded by c. For more details on* eIDs *and encodings, see the text.*

$\{0,1\}^m \times \{0,1\}^m \times \{-1,1\}$ on input $(q,r) \in \{0,1\}^m \times \{0,1\}^m$ as $\delta'(q,r) = (\varphi(q'), \theta(r'), i)$, where q', r', and i are specified by $\delta(\varphi^{-1}(q), \theta^{-1}(r)) = (q', r', i)$.

Recall that $\Sigma = \{0,1\}$. Define $\nu : \Gamma^* \to \mathbb{N}$ recursively as $\nu(\epsilon) = 0$ and, for each $y \in \Gamma$ and $w \in \Gamma^*$,

$$\nu(wy) = \begin{cases} 1 + 2\nu(w) & \text{if } y = 1 \wedge w \in \Sigma^* \\ 2\nu(w) & \text{if } y = 0 \wedge w \in \Sigma^* \\ \nu(w) & \text{if } y = B \\ 0 & \text{otherwise.} \end{cases}$$

This has the property that if $z \in \Sigma^*B^*$, then $\nu(z)$ is the natural number that z represents in binary. And if $z \in \Gamma^* - \Sigma^*B^*$, then $\nu(z) = 0$.

We also need the following notation. For any domain S, any (possibly partial)

function $h : S \to S$, any $i \in \mathbb{N}$, and any $s \in S$, we define $h^{(i)}(s)$ as

$$h^{(i)}(s) =_{\text{def}} \begin{cases} s & \text{if } i = 0 \\ h(h^{(i-1)}(s)) & \text{if } i > 0 \wedge (h^{(i-1)}(s) \text{ is defined}) \wedge \\ & \qquad (h^{(i-1)}(s) \in \text{domain}(h)) \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Note that if $h(a)$ is undefined, then so, for example, will be $h^{(1)}(a)$ and $h^{(2)}(a)$.

All logarithms in this paper are base two, i.e., $\log m$ means $\log_2 m$. Define functions r, s, and t on input $n \in \mathbb{N}$ as $r(n) =_{\text{def}} \lceil \log p(n) \rceil$ (recall that, by assumption, on any input of length n, M uses at most $p(n)$ tape cells and $p(n) > 0$), $t(n) =_{\text{def}} m2^{r(n)}$, and $s(n) =_{\text{def}} m + r(n) + t(n)$.

Let $\mathtt{eID} =_{\text{def}} \bigcup_{n=0}^{\infty} \{0,1\}^{n+2s(n)}$ be the set of *enhanced instantaneous descriptions* of M. Informally speaking, for each $n \in \mathbb{N}$ and $x \in \Sigma^n$, $q \in \{0,1\}^m$, $c \in \Sigma^{s(n)}$, $w \in \Sigma^{r(n)}$, and $X_0, X_1, \ldots, X_{2^{r(n)}-1} \in \Sigma^m$, the string $xqcwX_0X_1 \cdots X_{2^{r(n)}-1} \in \mathtt{eID}$ is interpreted as follows.

- The string x represents the input to f.
- The string q represents the instantaneous state of M.
- The string c will be used as an external clock ("external" because it is not maintained by M itself but rather by an "outside observer") to count the number of computational steps M has made so far. The presence of the external clock will allow us to adapt the next move function of M to the enhanced instantaneous descriptions of M in such a way that cycles never occur, even if M from a particular configuration may cycle. Note that, since the number of tape cells M uses is polynomially bounded in the length of its input, we need only a polynomial amount of bits for the clock. Intuitively speaking, if the clock "runs out of time" by running out of bits, then (assuming we chose a large enough polynomial to control the number of clock bits) we know that a cycle has occurred.
- The string w encodes the instantaneous position of the tape head, i.e., a position of 0 or 1 or ... or $2^{r(|x|)} - 1$ is encoded (respectively) by the string $0^{r(x)}$ or $0^{r(x)-1}1$ or ... or $1^{r(x)}$.
- The strings $X_0, X_1, \ldots, X_{2^{r(n)}-1}$ represent the instantaneous contents of the leftmost $2^{r(n)}$ tape cells of M.

Note that the second, fourth, and fifth sections of the string described above (i.e., q, w, and $X_0, X_1, \ldots, X_{2^{r(n)}-1}$) are already sufficient to describe M at any instant. Note also that, because s, r, and t are all polynomial-time computable and nondecreasing, we can, in polynomial time, for each $n \in \mathbb{N}$ and each $z \in \Sigma^{n+2s(n)}$, compute from z the value n and the locations of the five above-described sections of z, and these locations are well-defined.

For each $x \in \Sigma^*$, we call $x0^m0^{s(|x|)}0^{r(|x|)}\varphi(x)0^{t(|x|)-|\varphi(x)|} = x0^{2s(|x|)-t(|x|)}\varphi(x)0^{t(|x|)-|\varphi(x)|} \in \mathtt{eID}$ the *initial configuration* of M on x, denoted $i_{M,x}$. The string $i_{M,x}$ represents a configuration on which M would be started under "normal usage." Note that eID contains strings that represent configurations of M that are never reached under "normal usage." From these "unreachable" configurations, M may run forever or attempt to move off the left end of the tape. (Note that the true run of M on input x certainly does not run forever, since M is computing an FPSPACE(poly) function and FPSPACE(poly) is a class of total functions, and our model of function computing requires M to halt in order for it to compute a value. Recall that we assume that on no true run of M on input x will M attempt to move off the left end

of the tape. We did not explicitly discuss the semantics of attempting to move off the left end of the tape, but the point of the comment above is that even if our model of computing FPSPACE(poly) functions is such that moving off the left end of the tape is considered like running forever and makes a function be undefined on the input, and so never happens on a true run of a machine computing an FPSPACE(poly) function, it nonetheless may be the case that such a machine when started at some "unreachable" configuration might attempt to run off the left end of the tape.)

We define a *move* over $\texttt{eID}$ via a function $\mu : \Sigma^* \to \Sigma^*$ that we will define now. An important consideration in the design of μ is to exploit the additional information present in the enhanced IDs to guarantee that μ never loops and that it always "ends" (i.e., returns the value undefined) "gracefully" (in a sense that will soon become clear, including, for example, that it does not blindly try to move off the left end of the tape).

For each $x \in \Sigma^*$, $c \in \{0,1\}^{s(|x|)}$, $w \in \{0,1\}^{r(|x|)}$, $X_0, X_1, \ldots, X_{2^{r(|x|)}-1} \in \{0,1\}^m$, and $q \in \{0,1\}^m - \{1^m\}$,

$$\mu(xqcwX_0X_1\cdots X_{2^{r(|x|)}-1}) =_{\text{def}} xq'c'w'X_0X_1\cdots X_{\nu(w)-1}YX_{\nu(w)+1}X_{\nu(w)+2}\cdots X_{2^{r(|x|)}-1} \tag{1}$$

if

$$\delta'(q, X_{\nu(w)}) \text{ is defined } \wedge\, c \neq 1^{s(|x|)} \wedge 0 \leq \nu(w) + i < 2^{r(|x|)}, \tag{2}$$

where

$$\delta'(q, X_{\nu(w)}) = (q', Y, i),\ c' \in \{0,1\}^{s(|x|)},\ w' \in \{0,1\}^{r(|x|)},$$
$$\nu(c') = \nu(c) + 1, \text{ and } \nu(w') = \nu(w) + i,$$

and

$$\mu(xqcwX_0X_1\cdots X_{2^{r(|x|)}-1}) =_{\text{def}} x1^mcwX_0X_1\cdots X_{2^{r(|x|)}-1} \tag{3}$$

otherwise. If $q = 1^m$, $\mu(xqcwX_0X_1\cdots X_{2^{r(|x|)}-1})$ is undefined. For all $y \notin \texttt{eID}$, $\mu(y)$ is undefined. It is easy to see that the behavior of μ described by (1) is roughly analogous to the behavior of δ. Indeed, for all $x \in \Sigma^*$, there exists a number $j \in \mathbb{N}$ such that $\mu^{(j)}(i_{M,x}) = x1^mcwz$, where $c \in \{0,1\}^{s(|x|)}$, $w \in \{0,1\}^{r(|x|)}$, $z \in \{0,1\}^{t(|x|)}$, $\nu(c) = j$, and $\nu(\hat{\theta}^{-1}(z)) = f(x)$. Equation (3) enforces "gracefulness" by detecting when the configuration encoded by the input string is about to move off the left end of the tape or is about to use too much tape or has a "c" value that has already reached $2^{s(|x|)}$ (note that no actual run can ever run more than $2^{s(n)}$ steps without running forever, but running forever can never happen on actual runs since all functions in FPSPACE(poly) are total). In such cases, μ simply changes the state bits to represent the final state (i.e., 1^m).

Proposition 6.11 collects several easy-to-see properties of μ.

PROPOSITION 6.11.

1. *The function μ is polynomial-time computable.*
2. *The function μ is length-preserving, i.e., for all $w \in \Sigma^*$, if $\mu(w)$ is defined, then $|w| = |\mu(w)|$.*
3. *For all $x \in \Sigma^*$, all $w \in \{0,1\}^{2s(|x|)-m}$, and all $q \in \{0,1\}^m$, $\mu(xqw)$ is defined if and only if $q \neq 1^m$.*
4. *For all $w \in \Sigma^*$, there exists a number j such that $\mu^{(j)}(w)$ is undefined.*

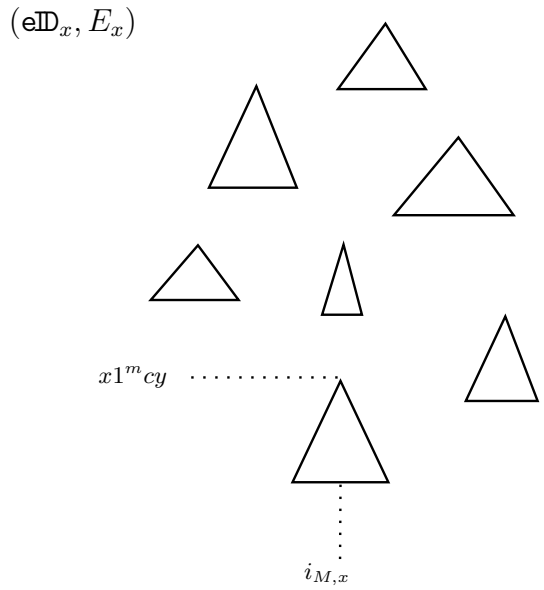


FIG. 3. *The directed forest* $(\mathbb{eID}_x, E_x)$. *Note that precisely one tree in the digraph* $(\mathbb{eID}_x, E_x)$ *has* $i_{M,x}$ *as a node, and note that in that tree* $i_{M,x}$ *will be a leaf node. For some* c *and* y *satisfying* $c \in \{0,1\}^{2s(|x|)-t(|x|)-m}$, $y \in \{0,1\}^{t(|x|)}$, *and* $\nu(\hat{\theta}^{-1}(y)) = f(x)$, *that tree will have as its root node* $x1^m cy$.

5. *In polynomial time we can, for each* $z \in \Sigma^*$, *enumerate all* y *such that* $\mu(y) = z$.
6. *For each* $w \in \mathbb{eID}$ *and each* $j \in \mathbb{N}^+$, *if* $\mu^{(j)}(w)$ *is defined, then* $\mu^{(j)}(w) \neq w$.

Proof. All items are easy to see. However, item 5 deserves some additional explanation. To perform this enumeration, if $z \notin \mathbb{eID}$, then there is no y such that $\mu(y) = z$. If $z \in \mathbb{eID}$, then examine the next move function of M to determine the configurations from which M in one step will move into the configuration encoded by z. There are only a constant number of such configurations. Output the strings of length $|z|$ that encode these configurations. This takes care of all preimages of z that satisfy equation (2). If, for some $x \in \Sigma^*$, $c \in \{0,1\}^{s(|x|)}$, $w \in \{0,1\}^{r(|x|)}$, and $X_0, X_1, \ldots, X_{2^{r(|x|)}-1} \in \{0,1\}^m$ it holds that $z = x1^m cwX_0X_1 \cdots X_{2^{r(|x|)}-1}$ (i.e., if z satisfies the conditions of equation (3)) then, for each $q \in \{0,1\}^m - \{1^m\}$ such that $xqcwX_0X_1 \cdots X_{2^{r(|x|)}-1}$ does not satisfy equation (2), output $xqcwX_0X_1 \cdots X_{2^{r(|x|)}-1}$. This takes care of all preimages of z that do not satisfy equation (2). □

Phase 3: Building trees. For each $x \in \Sigma^*$, let

$$\mathbb{eID}_x = \{xw \mid w \in \{0,1\}^{2s(|x|)}\}$$

and

$$E_x = \{(xw, xz) \mid xw, xz \in \mathbb{eID}_x \wedge \mu(xw) = xz\}.$$

A directed forest is an acyclic digraph in which all nodes have outdegree at most one. Note that the digraph $(\mathbb{eID}_x, E_x)$ has outdegree at most one. By Proposition 6.11.6, $(\mathbb{eID}_x, E_x)$ is acyclic. Thus, $(\mathbb{eID}_x, E_x)$ is a directed forest (see Figure 3).

For each $x \in \Sigma^*$, let (keep in mind that given the string $xw \in \mathbb{eID}$, it is easy to identify x and w)

$$\mathbb{eID}'_x = \{xwy \mid xw \in \mathbb{eID}_x \wedge y \in \{0,1\}^{t(|x|)}\}$$

and

$$E'_x = \{(xwy, xzy) \mid w \in \{0,1\}^{2s(|x|)} \wedge y \in \{0,1\}^{t(|x|)} \wedge xwy \in \mathbb{eID}'_x \wedge \mu(xw) = xz\}.$$

Note that the digraph $(\mathbb{eID}'_x, E'_x)$ is a directed forest, and that, for each tree in $(\mathbb{eID}_x, E_x)$, there are exactly $2^{t(|x|)}$ corresponding trees in $(\mathbb{eID}'_x, E'_x)$ (see Figure 4 for a pictorial preview of this part of the construction).

Let $\mathrm{R}_x =_{\mathrm{def}} \{xwy \in \mathbb{eID}'_x \mid w \in \{0,1\}^{2s(|x|)} \wedge y \in \{0,1\}^{t(|x|)} \wedge (\mu(xw) \text{ is undefined})\}$. Note that, by Proposition 6.11.3, $\mathrm{R}_x = \{xwy \in \mathbb{eID}'_x \mid w \in \{0,1\}^{2s(|x|)} \wedge y \in \{0,1\}^{t(|x|)} \wedge (xw \text{ is the root of a tree in } (\mathbb{eID}_x, E_x))\} = \{x1^m wy \in \mathbb{eID}'_x \mid w \in \{0,1\}^{2s(|x|)-m} \wedge y \in \{0,1\}^{t(|x|)}\}$. Let $\leq_{\mathrm{R}_x}$ denote the order (with $<_{\mathrm{R}_x}$ and $\prec_{\mathrm{R}_x}$ denoting the corresponding "less than" and "predecessor" relations, respectively) defined over R_x that is determined by the following sequence. (The reader is cautioned that in what follows "w" is used as a variable to catch substrings of various lengths other than the $2s(|x|)$-length strings it has been primarily used for so far.)

- First come the elements of $\{xwyy \in \mathrm{R}_x \mid w \in \{0,1\}^{2s(|x|)-t(|x|)} \wedge y \in \{0,1\}^{t(|x|)}\}$ in lexicographic order. Note that the last element in this sequence is $x1^{2s(|x|)+t(|x|)}$.
- Next come the elements of $\{xwdy \in \mathrm{R}_x \mid w \in \{0,1\}^{2s(|x|)-t(|x|)} \wedge d, y \in \{0,1\}^{t(|x|)} \wedge d \neq y\}$ in lexicographic order. Note that the last element in this sequence is $x1^{2s(|x|)+t(|x|)-1}0$.

For each $x \in \Sigma^*$, $w \in \{0,1\}^{2s(|x|)}$, and $y \in \{0,1\}^{t(|x|)}$, we define $\mu_1 : \Sigma^* \to \Sigma^*$, on input xwy, as

$$\mu_1(xwy) = \begin{cases} \mu(xw)y & \text{if } xwy \notin \mathrm{R}_x \\ xz & \text{if } xwy \in \mathrm{R}_x \wedge xwy \neq x1^{2s(|x|)+t(|x|)-1}0, \text{ where } xwy \prec_{R_x} xz. \end{cases}$$

In all other cases, μ_1 is undefined. Informally speaking, μ_1 is an "augmented next move" function based on μ but with the difference that μ_1 in effect strings together all the trees in $(\mathbb{eID}'_x, E'_x)$ into one giant tree $T_{M,x}$ (see Figure 4 again).

PROPOSITION 6.12. *For each $x \in \Sigma^*$, let $E''_x =_{\mathrm{def}} \{(w,z) \mid w \in \mathbb{eID}'_x \wedge \mu_1(w) = z\}$, and define $T_{M,x}$ to be the digraph $(\mathbb{eID}'_x, E''_x)$.*

1. *The function μ_1 is polynomial-time computable.*
2. *The function μ_1 is length-preserving (i.e., on inputs a for which it is not undefined, $|\mu_1(a)| = |a|$).*
3. *In polynomial time we can, for any $z \in \Sigma^*$, enumerate all $y \in \Sigma^*$ such that $\mu_1(y) = z$.*
4. *For every $x \in \Sigma^*$ and every $w \in \{0,1\}^{2s(|x|)+t(|x|)}$, there exists a number $j \in \mathbb{N}$ such that $\mu_1^{(j)}(xw) = x1^{2s(|x|)+t(|x|)-1}0$. (See also Figure 4.)*
5. *For every $x \in \Sigma^*$ and every $w \in \{0,1\}^{2s(|x|)+t(|x|)}$, $\mu_1(xw)$ is undefined if and only if $w = 1^{2s(|x|)+t(|x|)-1}0$.*
6. *For each $x \in \Sigma^*$ and each $w \in \{0,1\}^{2s(|x|)}$, there is a unique $y \in \{0,1\}^{t(|x|)}$ such that, for some $k \in \mathbb{N}$, $\mu_1^{(k)}(xwy) = x1^{2s(|x|)+t(|x|)}$. (Again, viewing Figure 4—paying particular attention to the black trees—will help make this clear).*
7. *For each $x \in \Sigma^*$, $\|\{w \mid (\exists j \in \mathbb{N})[\mu_1^{(j)}(w) = x1^{2s(|x|)+t(|x|)}]\}\| = 2^{2s(|x|)}$.*

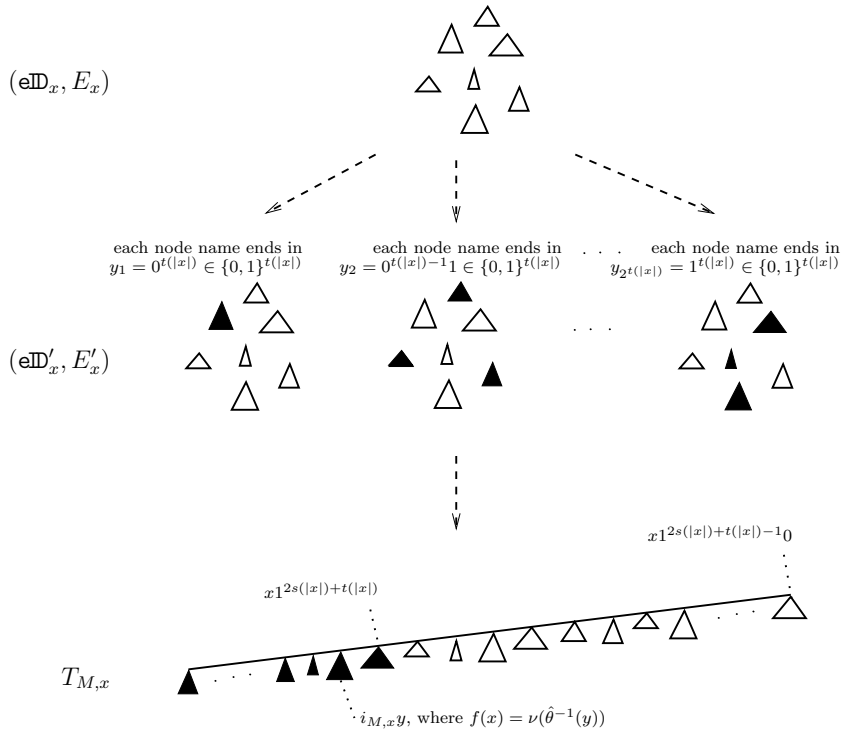


FIG. 4. *Transforming the directed forest* $(\mathbb{eID}_x, E_x)$ *into* $T_{M,x}$. *First,* $2^{t(|x|)}$ *copies of each tree in* $(\mathbb{eID}_x, E_x)$ *are made by appending* $t(|x|)$ *"guess" bits to each node in each original tree, creating the directed forest* $(\mathbb{eID}'_x, E'_x)$. *Next, the trees in* $(\mathbb{eID}'_x, E'_x)$ *are strung together into a single tree* $T_{M,x}$ *in such a way that a subtree of* $T_{M,x}$ *is formed by the trees in* $(\mathbb{eID}'_x, E'_x)$ *having (note:* R_x *will be defined in the main text) roots in* $\{xwyy \in \mathrm{R}_x \mid w \in \{0,1\}^{2(|s|)-t(|x|)} \wedge y \in \{0,1\}^{t(|x|)}\}$ *(represented in the figure by the black trees), i.e., the trees whose "guess" bits equal the contents of the machine tape at the end of the computation. This subtree has exactly one node for each string in* $\mathbb{eID}_x$, *including* $i_{M,x}$, *and the node associated with* $i_{M,x}$ *has as its "guess" bits the true output of* M *on input* x. *We will later exploit this information when we define a traversal of this tree.*

8. *For each* $x \in \Sigma^*$, *the unique (by item 6)* $y \in \{0,1\}^{t(|x|)}$, *and each* $k \in \mathbb{N}$ *such that* $\mu_1^{(k)}(i_{M,x}y) = x1^{2s(|x|)+t(|x|)}$, *it holds that* $f(x) = \nu(\hat{\theta}^{-1}(y))$.
9. *For each* $x \in \Sigma^*$, *the digraph* $T_{M,x}$ *is a tree.*
10. *The subtree of* $T_{M,x}$ *rooted at* $x1^{2s(|x|)+t(|x|)}$ *has exactly* $2^{2s(|x|)}$ *nodes.*

Proof. Items 1–5 follow from the definition of μ_1.

For item 6, choose an arbitrary $x \in \Sigma^*$, $w \in \{0,1\}^{2s(|x|)}$, and $y \in \{0,1\}^{t(|x|)}$, and let $j \in \mathbb{N}$, $v \in \{0,1\}^{2s(|x|)-t(|x|)}$, and $d \in \{0,1\}^{t(|x|)}$ be such that $\mu^{(j)}(xw) = xvd$ and $\mu(xvd)$ is undefined (such j, v, and d exist by Propositions 6.11.4 and 6.11.2). By the definition of R_x, $xvdy \in \mathrm{R}_x$. By the definition of $\leq_{\mathrm{R}_x}$, $\mu^{(j)}(xw)d \leq_{\mathrm{R}_x} x1^{2s(|x|)+t(|x|)}$ and so, by the definition of μ_1, there exists a number $k \geq j$ such that $\mu_1^{(k)}(xwd) = x1^{2s(|x|)+t(|x|)}$. On the other hand, for all $y \in \{0,1\}^{t(|x|)}$ such that $y \neq d$, by the definition of $\leq_{\mathrm{R}_x}$, $x1^{2s(|x|)+t(|x|)} <_{\mathrm{R}_x} \mu^{(j)}(xw)y$, and so, by items 4 and 5 (which guarantee that μ_1 does not cycle), there is no k such that $\mu_1^{(k)}(xwy) = x1^{2s(|x|)+t(|x|)}$.

Item 7 follows from item 6.

For item 8, choose an arbitrary $x \in \Sigma^*$, and by item 6 let y be the unique member of $\{0,1\}^{t(|x|)}$ such that, for some $k \in \mathbb{N}$, $\mu_1^{(k)}(i_{M,x}y) = x1^{2s(|x|)+t(|x|)}$. Choose

$j \in \mathbb{N}$ such that $\mu^{(j)}(i_{M,x})y \in \mathrm{R}_x$. By the definition of μ_1, there exists a number $v \in \{0,1\}^{2s(|x|)-t(|x|)}$ such that $\mu^{(j)}(i_{M,x}) = xvy$ and, by the definition of μ, M on input x halts with y on its tape. Thus, $f(x) = \nu(\hat{\theta}^{-1}(y))$.

Item 9 follows from items 4 and 5.

Item 10 follows from item 7 and the observation that, for any $x \in \Sigma^*$ and any $w, y \in \mathtt{eID}'$, w is in the subtree of $T_{M,x}$ rooted at y if and only if y is a node of $T_{M,x}$ and there exists a number $k \in \mathbb{N}$ such that $\mu_1^{(k)}(w) = y$. □

Phase 4: Defining a traversal. We define $\mathtt{dwn} : \Sigma^* \to \Sigma^* \cup \{\perp\}$, on input w, as

$$\mathtt{dwn}(w) = \begin{cases} \max_{\mathrm{lex}} \mu_1^{-1}(w) & \text{if } \mu_1^{-1}(w) \neq \emptyset \\ \perp & \text{otherwise,} \end{cases}$$

where $\max_{\mathrm{lex}}$ returns the maximal element (with respect to the lexicographical order) of a set of strings and we define $\mathtt{acr} : \Sigma^* \to \Sigma^* \cup \{\perp\}$ on input w as

$$\mathtt{acr}(w) = \begin{cases} \max_{\mathrm{lex}}\{w' \mid w' \in \mu_1^{-1}(\mu_1(w)) \wedge w' <_{\mathrm{lex}} w\} & \text{if } \mu_1(w) \text{ is defined} \\ & \quad \wedge\, w \neq \min_{\mathrm{lex}} \mu_1^{-1}(\mu_1(w)) \\ \perp & \text{otherwise,} \end{cases}$$

where $\min_{\mathrm{lex}}$ returns the minimal element (with respect to the lexicographical order) of a set of strings. Clearly, both $\mathtt{dwn}$ and $\mathtt{acr}$ are polynomial-time computable. The function $\mathtt{dwn}$ is named "$\mathtt{dwn}$" because it describes a descent down the tree $T_{M,x}$, and $\mathtt{acr}$ is named "$\mathtt{acr}$" because it describes movement across the tree (i.e., from one sibling node to another). Note that, for all $x \in \Sigma^*$ and all $w \in \{0,1\}^{2s(|x|)+t(|x|)}$ satisfying $xw \in \mathrm{R}_x - \{x1^m0^{2s(|x|)+t(|x|)-m}\}$, it holds that $\mathtt{dwn}(xw) \in \mathrm{R}_x$.

Now, for each $x \in \Sigma^*$, $w \in \{0,1\}^{2s(|x|)}$, $a \in \{0,1\}$, and $y, z \in \{0,1\}^{t(|x|)}$, we define $D_M : \Sigma^* \to \Sigma^*$, a "depth-first"-like traversal of $T_{M,x}$, on input $xwyza$, as

$$D_M(xwyza) = \begin{cases} \mathtt{dwn}(xwy)z0 & \text{if } a = 0 \wedge \mathtt{dwn}(xwy) \neq \perp \wedge \nu(z) = 0 \\ xwyz1 & \text{if } a = 0 \wedge \mathtt{dwn}(xwy) = \perp \wedge xw \neq i_{M,x} \wedge \nu(z) = 0 \\ xwyz'0 & \text{if } a = 0 \wedge \mathtt{dwn}(xwy) = \perp \wedge xw = i_{M,x} \wedge \\ & \qquad \nu(z) < \nu(\hat{\theta}^{-1}(y)), \text{ where } z \prec_{\mathrm{lex}} z' \\ xwy0^{t(|x|)}1 & \text{if } a = 0 \wedge \mathtt{dwn}(xwy) = \perp \wedge xw = i_{M,x} \wedge \\ & \qquad \nu(z) = \nu(\hat{\theta}^{-1}(y)) \\ \mathtt{acr}(xwy)z0 & \text{if } a = 1 \wedge \mathtt{acr}(xwy) \neq \perp \wedge \nu(z) = 0 \\ \mu_1(xwy)z1 & \text{if } a = 1 \wedge \mathtt{acr}(xwy) = \perp \wedge \nu(z) = 0. \end{cases}$$

On all other inputs, D_M is undefined. Figure 5 illustrates the action of D_M and Proposition 6.13 formally establishes the most important aspects of D_M's action, most of which we will draw on soon.

PROPOSITION 6.13.

1. *The function D_M is polynomial-time computable.*
2. *The function D_M is length-preserving (i.e., for each v, either $D_M(v)$ is undefined or $|D_M(v)| = |v|$).*
3. *For each $x \in \Sigma^*$, each subtree (and here we really mean each subtree, i.e., not just those corresponding to the trees in digraph $(\mathtt{eID}'_x, E'_x)$—the purpose of this item is to provide insight into how D_M describes a traversal of $T_{M,x}$) T of $T_{M,x}$, each $w \in \{0,1\}^{2s(|x|)}$, and each $y \in \{0,1\}^{t(|x|)}$, xwy is a node of T if and only if there exist i, j, and k such that $|v| = |xwy|$ (where v is the*

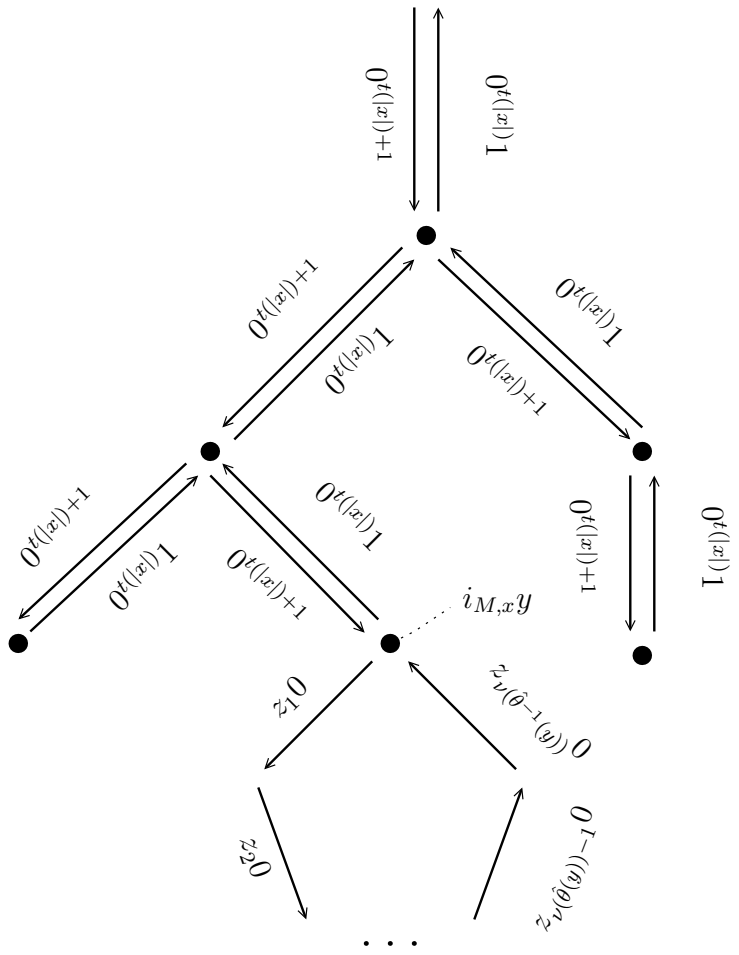


FIG. 5. *The traversal described by D_M. Pictured is a portion of $T_{M,x}$ that contains a node in the initial configuration. The arrows represent the strings associated with the node below them (in the case of the initial configuration node, the arrows below are also associated with it) by padding. The string that is the actual padding appears next to each arrow. D_M is defined over these padded strings. The last bit of each padding string can be seen as controlling the "direction" in which D_M "moves." Note that $y \in \{0,1\}^{t(|x|)}$ and $z_1 = 0^{t(|x|)-1}1, z_2 = 0^{t(|x|)-2}10, \ldots$.*

root of T), $D_M^{(i)}(v0^{t(|x|)+1}) = xwy0^{t(|x|)+1}$, $D_M^{(j)}(xwy0^{t(|x|)+1}) = xwy0^{t(|x|)}1$, and $D_M^{(k)}(xwy0^{t(|x|)}1) = v0^{t(|x|)}1$.

4. *For every $x \in \Sigma^*$, $w \in \{0,1\}^{2s(|x|)}$, $a \in \{0,1\}$, and $y, z \in \{0,1\}^{t(|x|)}$, $D_M(xwyza)$ is defined if and only if $xwyza = x1^{2s(|x|)+t(|x|)-1}0^{t(|x|)+2} \vee (wy \in \{0,1\}^{2s(|x|)+t(|x|)} - \{1^{2s(|x|)+t(|x|)-1}0\} \wedge z = 0^{t(|x|)}) \vee (xw = i_{M,x} \wedge \nu(z) \leq \nu(\hat{\theta}^{-1}(y)) \wedge a = 0)$.*
5. *For every $x \in \Sigma^*$ and every $w \in \{0,1\}^{2(s(|x|)+t(|x|))+1}$, if $D_M(xw)$ is defined, then there exists an $i \in \mathbb{N}$ such that $D_M^{(i)}(x1^{2s(|x|)+t(|x|)-1}0^{t(|x|)+2}) = xw$.*
6. *For all $x \in \Sigma^*$, all $w \in \{0,1\}^{2s(|x|)+t(|x|)}$, all $z \in \{0,1\}^{t(|x|)+1}$, and all $i \in \mathbb{N}$, if $D_M^{(i)}(xwz) = x1^{2s(|x|)+t(|x|)}0^{t(|x|)+1}$, then $xw \in \mathrm{R}_x$.*
7. *The function $\lambda y.\min_{\mathrm{lex}}\{w \mid y <_{\mathrm{lex}} w \wedge (D_M(w)$ is undefined$)\}$ is polynomial-time computable.*

Proof. Items 1 and 2 follow from the definition of D_M.

For item 3, choose an arbitrary $x \in \Sigma^*$. We prove item 3 by induction over the depth of the subtrees of $T_{M,x}$.

For the base case, choose an arbitrary subtree T of $T_{M,x}$ having depth 1. Let v be the (only) node of T. Thus, $\mathtt{dwn}(v) = \perp$. If, for all $y \in \{0,1\}^{t(|x|)}$, $v \neq i_{M,x}y$, then, by the definition of D_M, $D_M^{(0)}(v0^{t(|x|)+1}) = v0^{t(|x|)+1}$, $D_M(v0^{t(|x|)+1}) = v0^{t(|x|)}1$, and $D_M^{(0)}(v0^{t(|x|)}1) = v0^{t(|x|)}1$. Otherwise, let $y \in \{0,1\}^{t(|x|)}$ be such that $v = i_{M,x}y$. Then $D_M^{(0)}(v0^{t(|x|)+1}) = v0^{t(|x|)+1}$, $D_M^{(\nu(\hat{\theta}^{-1}(y))+1)}(v0^{t(|x|)+1}) = v0^{t(|x|)}1$, and $D_M^{(0)}(v0^{t(|x|)}1) = v0^{t(|x|)}1$.

For the induction case, suppose, for some n that is less than the depth of $T_{M,x}$ and all subtrees T of $T_{M,x}$ having depth at most n, that the induction hypothesis holds. Let S be a subtree of $T_{M,x}$ of depth $n+1$, and let v be the root of S. Let $\{a_1, \ldots, a_b\} = \mu_1^{-1}(v)$, where $a_b <_{\text{lex}} \cdots <_{\text{lex}} a_1$. It follows that each $a_1, \ldots, a_b$ is the root of a subtree of S of depth at most n. By the definition of D_M, $D_M(v0^{t(|x|)+1}) = a_1 0^{t(|x|)+1}$, $D_M(a_1 0^{t(|x|)}1) = a_2 0^{t(|x|)+1}, \ldots, D_M(a_{b-1}0^{t(|x|)}1) = a_b 0^{t(|x|)+1}$, and $D_M(a_b 0^{t(|x|)}1) = v0^{t(|x|)}1$. By applying the induction hypothesis to the subtrees of S rooted at $a_1, \ldots, a_b$, we conclude that z is a node of S if and only if there exist i, j, k such that $D_M^{(i)}(v0^{t(|x|)+1}) = z0^{t(|x|)}1$, $D_M^{(j)}(z0^{t(|x|)+1}) = z0^{t(|x|)}1$, and $D_M^{(k)}(z0^{t(|x|)}1) = v0^{t(|x|)}1$.

Item 4 follows from the definition of D_M (to see the case where $xwyza = x1^{2s(|x|)+t(|x|)-1}0^{t(|x|)+2}$, it helps to note that μ_1 is undefined on $x1^{2s(|x|)+t(|x|)-1}0$ and thus $D_M(x1^{2s(|x|)+t(|x|)-1}0^{t(|x|)+2})$ is defined but $D_M(x1^{2s(|x|)+t(|x|)-1}0^{t(|x|)+1}1)$ is not).

For item 5, choose arbitrary $x \in \Sigma^*$, $w \in \{0,1\}^{2s(|x|)}$, $a \in \{0,1\}$, and $y, z \in \{0,1\}^{t(|x|)}$. If $xwyza = x1^{2s(|x|)+t(|x|)-1}0^{t(|x|)+2} \vee (wy \in \{0,1\}^{2s(|x|)+t(|x|)} - \{1^{2s(|x|)+t(|x|)-1}0\} \wedge z = 0^{t(|x|)})$, then, by item 3, there exists an $i \in \mathbb{N}$ such that $D_M^{(i)}(x1^{2s(|x|)+t(|x|)-1}0^{t(|x|)+2}) = xwyza$. If $xw = i_{M,x} \wedge \nu(z) \leq \nu(\hat{\theta}^{-1}(y)) \wedge a = 0$, then, by the definition of D_M, $D_M^{(\nu(z))}(xwy0^{t(|x|)+1}) = xwyza$. Since, by item 3, there exists an $i \in \mathbb{N}$ such that $D_M^{(i)}(x1^{2s(|x|)+t(|x|)-1}0^{t(|x|)+2}) = xwy0^{t(|x|)+1}$, it holds that $D_M^{(i+\nu(z))}(x1^{2s(|x|)+t(|x|)-1}0^{t(|x|)+2}) = xwyza$.

For item 6, choose an arbitrary $x \in \Sigma^*$. Recall that, for all $xw \in \mathrm{R}_x - \{x1^m 0^{2s(|x|)+t(|x|)-m}\}$, $\mathtt{dwn}(xw) \in \mathrm{R}_x$. Thus, since $x1^{2s(|x|)+t(|x|)} \in \mathrm{R}_x$ and $x1^{2s(|x|)+t(|x|)-1}0 \in \mathrm{R}_x$, it follows from the definitions of $\mathtt{dwn}$ and $\leq_{\mathrm{R}_x}$ that, for some $i \in \mathbb{N}$, $\mathtt{dwn}^{(i)}(x1^{2s(|x|)+t(|x|)-1}0) = x1^{2s(|x|)+t(|x|)}$, and for all $j \in \mathbb{N}$ such that $0 \leq j \leq i$, it holds that $\mathtt{dwn}^{(j)}(x1^{2s(|x|)+t(|x|)-1}0) \in \mathrm{R}_x$. Thus, by the definition of D_M, $D_M^{(i)}(x1^{2s(|x|)+t(|x|)-1}0^{t(|x|)+2}) = x1^{2s(|x|)+t(|x|)}0^{t(|x|)+1}$, and for all $j \in \mathbb{N}$ such that $0 \leq j \leq i$, it holds that $D_M^{(j)}(x1^{2s(|x|)+t(|x|)-1}0^{t(|x|)+2}) = w0^{t(|x|)+1}$, where $w \in \mathrm{R}_x$.

For item 7, note that, by item 4, for all $w, y, z \in \Sigma^*$ such that $w \prec_{\text{lex}} y \prec_{\text{lex}} z$, either $D_M(y)$ is undefined or $D_M(z)$ is undefined. □

Phase 5: Creating *A*. We are now ready to define A. A is the same as the lexicographical ordering except that the strings between $x0^{2(s(|x|)+t(|x|))+1}$ and $x1^{2(s(|x|)+t(|x|))+1}$ are ordered as follows (let $z = x1^{2s(|x|)+t(|x|)-1}0^{t(|x|)+1}$).

- First come the strings $D_M^{(0)}(z0) = z0$, $D_M^{(1)}(z0) = D_M(z0)$, $D_M^{(2)}(z0), \ldots, z1$, in the order just stated.
- Next come the strings $\{xw \mid w \in \{0,1\}^{2(s(|x|)+t(|x|))+1} \wedge D_M(xw)$ is undefined

$\wedge xw \neq z1\}$, in lexicographical order.
By Proposition 6.13.2, A is a p-order. By Proposition 6.13.4, A is total. By Propositions 6.13.1 and 6.13.7, A has efficient adjacency checks.

End of Construction

We are now ready to prove Lemma 6.5.

Proof of Lemma 6.5. For each $f \in$ FPSPACE(poly), we define A as above. We define $b : \Sigma^* \to \Sigma^*$, $t : \Sigma^* \to \Sigma^*$, and $b' : \Sigma^* \to \Sigma^*$ on input $x \in \Sigma^*$ as, respectively, $b(x) =_{\text{def}} x1^{2s(|x|)+t(|x|)}0^{t(|x|)+1}$, $t(x) =_{\text{def}} x1^{2s(|x|)+t(|x|)}0^{t(|x|)}1$, and $b'(x) =_{\text{def}} i_{M,x}y0^{t(|x|)+1}$, where $y = \theta(1)0^{t(|x|)-|\theta(1)|}$ (thus $\nu(\hat{\theta}^{-1}(y)) = 1$). Note that each of these functions is in FP.

For item 1, note that s is polynomially bounded.

For item 2, we prove that, for all $x \in \Sigma^*$, $\|\{z \mid b(x) <_A z <_A t(x)\}\| = 2^{2s(|x|)+1} + f(x) - 2$. Choose an arbitrary $x \in \Sigma^*$. By Proposition 6.13.4, both $D_M(x1^{2s(|x|)+t(|x|)}0^{t(|x|)+1})$ and $D_M(x1^{2s(|x|)+t(|x|)}0^{t(|x|)}1)$ are defined. Thus, by the definition of A, $\{z \mid b(x) <_A z <_A t(x)\} = \{z \mid (\exists i, k \in \mathbb{N} : i > 0 \wedge k > 0)[D_M^{(i)}(b(x)) = z \wedge D_M^{(k)}(z) = t(x)]\}$. By Proposition 6.12.10, there are exactly $2^{2s(|x|)}$ strings in the subtree of $T_{M,x}$ rooted at $x1^{2s(|x|)+t(|x|)}$. Let $S = \{xwy0^{t(|x|)}a \mid w \in \{0,1\}^{2s(|x|)} \wedge a \in \{0,1\} \wedge y \in \{0,1\}^{t(|x|)} \wedge b(x) <_A xwy0^{t(|x|)}a <_A t(x)\}$. By Proposition 6.13.3, $\|S\| = 2^{2s(|x|)+1} - 2$. By Propositions 6.12.6 and 6.13.3, there is a unique $y' \in \{0,1\}^{t(|x|)}$ such that $i_{M,x}y'0^{t(|x|)+1} \in \{z \mid b(x) <_A z <_A t(x)\}$. Moreover, by Proposition 6.12.8, $\nu(\hat{\theta}^{-1}(y')) = f(x)$. By the definition of D_M, $D_M^{(\nu(\hat{\theta}^{-1}(y'))+1)}(i_{M,x}y'0^{t(|x|)+1}) \in S$, and for each $i \in \mathbb{N}$ such that $0 < i \leq \nu(\hat{\theta}^{-1}(y'))$, it holds that $D_M^{(i)}(i_{M,x}y'0^{t(|x|)+1}) \notin S$. For each of the remaining $2^{2s(|x|)+1} - 3$ strings w in S, $D_M(w) \in S \cup \{t(x)\}$. Thus $\|\{z \mid b(x) <_A z <_A t(x)\}\| = 2^{2s(|x|)+1} + f(x) - 2$.

For item 3, we prove that $\|\{z \mid b'(x) <_A z <_A t(x)\}\| > 0$ if and only if $f(x) = 1$. Choose $x \in \Sigma^*$ and let $y = \theta(1)0^{t(|x|)-|\theta(1)|}$. Suppose that $f(x) = 1$. Then, by Proposition 6.12.8, $xi_{M,x}y$ is in the subtree of $T_{M,x}$ rooted at $x1^{2s(|x|)+t(|x|)}$. Thus, by Proposition 6.13.3, there exists a k such that $D_M^{(k)}(b'(x)) = t(x)$. By the definitions of D_M, b', and t, $D_M(b'(x)) \neq t(x)$, thus $k > 1$. By the definition of A, $\|\{z \mid b'(x) <_A z <_A t(x)\}\| > 0$. Now, suppose $f(x) \neq 1$. Since $f(x) \neq \nu(\hat{\theta}^{-1}(y))$, it follows from Proposition 6.12.8 that $i_{M,x}y$ is not in the subtree of $T_{M,x}$ rooted at $x1^{2s(|x|)+t(|x|)}$. Thus, by Proposition 6.13.3, for all $k \in \mathbb{N}$, $D_M^{(k)}(b'(x)) \neq t(x)$. Thus $b'(x) \not<_A t(x)$, and so $\|\{z \mid b'(x) <_A z <_A t(x)\}\| = 0$. □

7. The complexity of counting divisors. Consider the function $\#\text{DIV} : \mathbb{N} \to \mathbb{N}$, defined on input $m \in \mathbb{N}$ as

$$\#\text{DIV}(m) =_{\text{def}} \begin{cases} \|\{n \in \mathbb{N} \mid n \neq 1, n \neq m, \text{ and } n \text{ divides } m\}\| & \text{if } m \geq 1 \\ 0 & \text{otherwise.} \end{cases}$$

What can we say about its complexity? We claim that #DIV belongs to the interval size function class IF_{p}.

THEOREM 7.1. #DIV *is in* IF_{p}.

Proof. Let PRIMES be the set of all prime numbers. Observe that #DIV $\in$ #P and PRIMES $= \{x \mid \#\text{DIV}(x) = 0\}$. PRIMES $\in$ P [AKS04]. Thus Theorem 7.1 follows from Theorem 5.3. □

8. The complexity of counting satisfying assignments of monotone formulas. In this section, we show that the #MONSAT function fits into our collection of function classes. A *monotone boolean function* is any boolean function such that

changing an input from 0 to 1 (while keeping all other inputs fixed) never changes the value of the function from 1 to 0. A *positive boolean formula* is a boolean formula that computes a monotone boolean formula. A *monotone boolean formula* is a formula that is built from propositional variables and the connectives $\wedge$ and $\vee$. Note that the class of functions computed by monotone boolean formulas is exactly the monotone boolean functions. Monotone computing models have long been studied (see, e.g., Grigni and Sipser [GS92] and the references therein).

Define

$$\#\mathrm{MONSAT}(F) =_{\mathrm{def}} \begin{cases} \|\{(a_1,\ldots,a_n) \mid \\ \quad (\forall i : 1 \le i \le n)[a_i \in \{0,1\}] \wedge F(a_1,\ldots,a_n) = 1\}\| \\ \qquad\qquad \text{if } F \text{ is a monotone boolean formula} \\ 0 \quad \text{otherwise,} \end{cases}$$

i.e., $\#\mathrm{MONSAT}(F)$ counts the number of satisfying assignments of monotone boolean formulas. For the remainder of this section, we identify each assignment $(a_1,\ldots,a_n)$ to the n variables of F with the n-bit string $a_1\ldots a_n \in \{0,1\}^n$. Theorem 8.5 states that #MONSAT belongs to the class IF_{t}. To prove this theorem, we will use the following proposition.

PROPOSITION 8.1. *Let φ be the function that is defined for every boolean formula $F(x_1,\ldots,x_n)$, $a \in \{0,1\}^n$, and $r \in \{0,1\}$ as $\varphi(F,a,r) =_{\mathrm{def}} \min\{\, b \mid b \in \{0,1\}^n \wedge a \le_{\mathrm{lex}} b \wedge F(b) = r \,\}$ if $\{\, b \mid b \in \{0,1\}^n \wedge a \le_{\mathrm{lex}} b \wedge F(b) = r \,\}$ is nonempty and F is a monotone boolean formula, and $\varphi(F,a,r) =_{\mathrm{def}} \perp$ otherwise, where the* min *in the above definition is taken with respect to the lexicographical order. The function φ is polynomial-time computable.*

Proof. To prove this proposition we use two natural properties of monotone boolean formulas. First, note that, for each monotone boolean formula F of arity n and for each $a = a_1\ldots a_n \in \{0,1\}^n$ and $b = b_1\ldots b_n \in \{0,1\}^n$, it holds that $F(a) \le F(b)$ whenever $(\forall i \le n)[a_i \le b_i]$. Second, there is an assignment making F true (respectively, false) if and only if $F(1^n) = 1$ (respectively, $F(0^n) = 0$). Consider the algorithm of Figure 6 running on an n-ary monotone boolean formula F, $a \in \{0,1\}^n$, and $r \in \{0,1\}$.

The algorithm works as follows. If none of the boundary conditions in lines [1] through [6] are met, then assume that the assignments to the variables of F are just the labels of the leaves of a complete binary tree having 2^n leaves, i.e., the leftmost leaf is 0^n, and the rightmost leaf 1^n. The algorithm starts in the leaf numbered a, and searches the next node u on the path from a to the root such that the path comes into u from the left, and the right subtree below u contains an assignment b with $F(b) = r$ (lines [6] to [9]). The least b of the subtree having this property is determined via binary search (lines [10] to [18]). Thus, the algorithm is correct and runs in polynomial time with respect to the input length. □

We state as Proposition 8.2 some subcases of Proposition 8.1. (A "part 2 of Proposition 8.2" parallel to the first sentence of part 1 of Proposition 8.2 is not included since that trivially holds (test the all-0 assignment).) Though we could not find Proposition 8.2 in the literature, it is sufficiently fundamental that we believe it may well be known or a folk theorem.

PROPOSITION 8.2.

1. *The problem of finding the least satisfying assignment for monotone boolean formulas has a polynomial-time algorithm. Indeed, the problem of finding the least satisfying assignment lexicographically greater than or equal to a given assignment has, for monotone boolean formulas, a polynomial-time algorithm.*

```
[1]   b ← a
[2]   if (b = 1^n and F(b) ≠ r) or F(r^n) ≠ r
[3]      then
[4]         return ⊥
[5]      else
[6]         while b ≠ ε and F(br^{n−|b|}) ≠ r do
[7]            b ← the string which succeeds b in lexicographical order
[8]            b ← longest prefix of b which ends with 1
[9]         endwhile
[10]        m ← |b| + 1
[11]        for j ← m to n do
[12]           if F(b0r^{n−|b|−1}) = r
[13]              then
[14]                 b ← b0
[15]              else
[16]                 b ← b1
[17]           endif
[18]        endfor
[19]        return b
[20]  endif
```

FIG. 6. *An algorithm used in the proof of Proposition* 8.1.

2. *The problem of finding the least unsatisfying assignment lexicographically greater than or equal to a given assignment has, for monotone boolean formulas, a polynomial-time algorithm.*

This section has, so far, spoken of monotone boolean formulas. However, note that if we view the algorithm from Figure 6 as accessing a black-box boolean *function*, the algorithm in fact shows that the *query complexity* of the task is polynomial—indeed linear—if the black-box function is a monotone boolean function. Thus we have the following results.

PROPOSITION 8.3. *Let φ be the function that is defined for every $n \geq 1$, every boolean formula $f(x_1, \ldots, x_n)$, every $a \in \{0,1\}^n$, and every $r \in \{0,1\}$ as*

$$\varphi^f(a,r) =_{\text{def}} \begin{cases} \min\{b \mid b \in \{0,1\}^n \wedge a \leq_{\text{lex}} b \wedge f(b) = r\} \\ \quad \textit{if } \{b \mid b \in \{0,1\}^n \wedge a \leq_{\text{lex}} b \wedge f(b) = r\} \neq \emptyset \\ \bot \quad \textit{otherwise,} \end{cases}$$

where the min *in the above definition is taken with respect to the lexicographical order. When restricted to monotone boolean functions, the function φ is of linear (in the number of variables) query complexity (and polynomial, in the number of variables, time complexity). That is, there exist a Turing machine M and a linear function q and a polynomial s such that for each $n \geq 1$, each monotone boolean n-variable function f, each $a \in \{0,1\}^n$, and each $r \in \{0,1\}$ it holds that*

1. *$M^f(a,r)$ makes at most $q(n)$ queries to f, and*
2. *$M^f(a,r)$ halts within $s(n)$ steps with $\varphi^f(a,r)$ on its output tape.*

Similarly to Proposition 8.2, we have the following (where the time and query complexities are relative to the number of variables or, equivalently, relative to the size of the "input," i.e., $|a| + |r|$).

PROPOSITION 8.4.

1. *The problem of finding the least satisfying assignment when restricted to monotone boolean functions has a linear-query-complexity algorithm (that in addition is of polynomial-time complexity). Indeed, the problem of finding the least satisfying assignment lexicographically greater than or equal to a given assignment has, when restricted to monotone boolean functions, a linear-query-complexity algorithm (that in addition is of polynomial-time complexity).*
2. *The problem of finding the least unsatisfying assignment lexicographically greater than or equal to a given assignment has, when restricted to monotone boolean functions, a polynomial-time algorithm.*

Note that in neither Proposition 8.3 nor Proposition 8.4 do we make any claims about what the procedure will compute if the black-box function is not a monotone boolean formula.

We now relate #MONSAT to interval functions.

THEOREM 8.5. $\#\mathrm{MONSAT} \in \mathrm{IF_t}$.

Proof. We assume that F is given as a string over the alphabet Σ. We construct a total p-order $A \in \mathrm{P}$ having efficient adjacency checks as follows. Generally, A coincides with the lexicographical order on Σ^* except that, for each monotone boolean formula F of arity n, the interval between $1^{|F|}0F0000^n$ and $1^{|F|}0F1001^n$ is ordered in the following way.

- First comes $\{1^{|F|}0F000y \mid |y| = n\}$ in lexicographical order (we always use $n = n_F$ to denote the arity of F).
- Next comes the set $\{1^{|F|}0F001a \mid a \text{ is a satisfying assignment of } F\}$ in lexicographical order.
- Next comes $\{1^{|F|}0F010y \mid |y| = n\}$ in lexicographical order.
- Next comes the set $\{1^{|F|}0F011a \mid a \text{ is not a satisfying assignment of } F\}$ in lexicographical order.
- Finally comes the set $\{1^{|F|}0F100y \mid |y| = n\}$ in lexicographical order.

Clearly, A is a total p-order that is decidable in polynomial time. In light of the function φ from Proposition 8.1 it is not hard to see that A has efficient adjacency checks. Also, for any monotone boolean formula $F(x_1, \ldots, x_n)$, let $b(F) =_{\mathrm{def}} 1^{|F|}0F0001^n$ and $t(F) =_{\mathrm{def}} 1^{|F|}0F0100^n$. Obviously, $b, t \in \mathrm{FP}$, and we obtain $\#\mathrm{MONSAT}(F) = \|\{z \mid b(F) <_A z <_A t(F)\}\|$. Thus, $\#\mathrm{MONSAT} \in \mathrm{IF_t}$. □

Valiant [Val79] showed that counting the number of satisfying assignments of 2CNF monotone formulas is Turing complete for #P. Since #2CNFMONSAT metrically reduces to #MONSAT, we immediately obtain from this theorem that #MONSAT is complete for $\mathrm{IF_t}$ under Turing reductions, and we get an alternate proof for Corollary 5.8.

9. Cluster computations. Finally, we discuss the complexity of computing the size of intervals for which the boundaries are not required to be polynomial-time computable. This leads to the notion of cluster computation, as introduced in [Kos99] for the case of the lexicographical order. We first review the formal definitions related to cluster computation, but here we present a more general version of the definitions than what previously appeared in [Kos99].

Let M be any nondeterministic Turing machine that is "balanced" in the sense that, on every input, the graph of the nondeterministic choices M makes is a complete, balanced, binary tree. Let y and z encode computation paths of M on x. By the above assumption that M is "balanced," $|y| = |z|$. Fix a total order A on Σ^*. We say

that $y \sim_{A,M,x} z$ if and only if (a) $y \prec_A z$ or $z \prec_A y$, and (b) M on x accepts on path y if and only if M on x accepts on path z. Let $\equiv_{A,M,x}$ be the equivalence closure (i.e., the reflexive-symmetric-transitive closure) of $\sim_{A,M,x}$. Then the relation $\equiv_{A,M,x}$ is an equivalence relation and thus induces a partitioning of the computation tree of M on x. An A-cluster is an equivalence class whose representatives are accepting paths. Additionally, we consider $\emptyset$ to be a valid A-cluster.

For a nondeterministic Turing machine M, let $acc_M(x) \subseteq \Sigma^*$ denote the set of all accepting paths of M on input x. Let $\#acc_M : \Sigma^* \to \mathbb{N}$ be the function defined as $\#acc_M(x) =_{\text{def}} \|acc_M(x)\|$. Let $out_M(x) \subseteq \Sigma^*$ denote the set of all distinct outputs of accepting paths of M on input x. A nondeterministic Turing machine M is a *lexicographical cluster machine* if and only if M is balanced in the sense defined earlier and, for every x, there is a computation path y of M on x such that

$$acc_M(x) = \{z \mid z \equiv_{\text{lex},M,x} y \text{ and } y \in acc_M(x)\}.$$

The intuition here is simple: Such machines on each input in the set have a single, nonempty, contiguous stretch of accepting paths.

DEFINITION 9.1 (Kosub [Kos99]).

c#P $=_{\text{def}} \{\#acc_M \mid M$ *is a polynomial-time lexicographical cluster machine*$\}$.

We mention some basic properties of the class c#P.

DEFINITION 9.2. *A nondeterministic Turing machine* computes a function f almost-uniquely *if and only if, for each* x,

1. $f(x) > 0$ *implies* $out_M(x) = \{f(x)\}$ *and* $\#acc_M(x) = 1$, *and*
2. $f(x) = 0$ *implies* $out_M(x) = \emptyset$.

Recalling from section 6.1 the definition of $\ominus$, we have the following.

PROPOSITION 9.3 (Kosub [Kos99]).

1. *A function* f *lies in* c#P *if and only if there exists a nondeterministic polynomial-time Turing machine that computes* f *almost-uniquely.*
2. $\text{UPSV}_\text{t} \subseteq \text{c\#P} = \text{c\#P} \ominus \text{FP} \subseteq \#\text{P}$.
3. $\text{UPSV}_\text{t} \cap \text{Nonzero} = \text{c\#P} \cap \text{Nonzero}$.
4. c#P $= \#$P *if and only if* UP $=$ PP.

PROPOSITION 9.4.

1. $\exists \cdot \text{c\#P} = \exists \cdot (\text{c\#P} - \text{FP}) = \text{UP}$.
2. *If* $\text{IF}_\text{t} \subseteq \text{c\#P}$, *then* UP $=$ PP.
3. *If* $\text{c\#P} \subseteq \text{IF}_\text{t}$, *then* P $=$ UP.

Proof. (1): It is easy to see that UP $\subseteq \exists \cdot$ c#P, since any balanced machine for a given UP language already implicitly shows that that language is in $\exists \cdot$ c#P due to the unique paths being each a size-one equivalence class. It follows from the definitions that $\exists \cdot (\text{c\#P} \ominus \text{FP}) \subseteq \exists \cdot (\text{c\#P} - \text{FP})$ and from Proposition 9.3.2 we have $\exists \cdot \text{c\#P} = \exists \cdot (\text{c\#P} \ominus \text{FP})$. However, in light of Proposition 9.3.1, we can see that each set in $\exists \cdot (\text{c\#P} - \text{FP})$ is in fact in UP.

(2): By Theorem 5.7, $\text{IF}_\text{t} \subseteq$ c#P implies $\#\text{P} - \text{FP} \subseteq \text{c\#P} - \text{FP}$. From this, Proposition 2.2.4, and the first part of the present result we have $\text{PP} = \exists \cdot (\#\text{P} - \text{FP}) \subseteq \exists \cdot (\text{c\#P} - \text{FP}) = \text{UP}$.

(3): Apply the operator $\exists$ to both sides of the inclusion, and apply Lemma 5.9 and the first part of the present result. □

Proposition 9.3, which in essence says that c#P functions are relatively simple, is *extremely* dependent on the fact that c#P is built based on lexicographical order. In particular, the results reflect the fact that it is easy, given two strings, a and b, to compute $\|\{c \mid a \leq_{\text{lex}} c \leq_{\text{lex}} b\}\|$. Proposition 9.3.1 for example is driven in large part

by the fact that one can, for inputs where the function is not zero, guess (and check the guess of) the rightmost and leftmost accepting paths, and then, since one knows that the complete set of accepting paths is simply the contiguous block between and including these, one can easily compute the number of accepting paths.

It is natural to wish to remove the focus here on lexicographic order, and to instead study machines whose set of accepting paths is always a contiguous block—with respect to some total order that has efficient adjacency checks like lexicographic order, but that perhaps does not satisfy the extremely restrictive "interval sizes are always trivial to compute" property of lexicographic order. We introduce the class CL#P, which captures exactly this more flexible, natural notion of cluster computing. (The work of this paper led to further study of CL#P in [HHK06], which studies the closure properties of, alternate definitions of, and classes related to CL#P.)

An order A on Σ^* is said to be *length-respecting* if and only if, for all x, y, $|x| < |y|$ implies $x <_A y$. Note that a length-respecting order is always a p-order.

DEFINITION 9.5. *A function f belongs to the class* CL#P *if and only if there exist a nondeterministic polynomial-time Turing machine M, a polynomial p, and a length-respecting total order A with efficient adjacency checks such that, for all x, the following conditions hold.*

1. *All computation paths of M on x have length exactly $p(|x|)$.*[1]
2. *The set of all accepting paths of M on x is an A-cluster.*
3. *$f(x) = \#acc_M(x)$.*

As might be expected, the class $\mathrm{IF_t}$ is included in CL#P. Indeed, the following inclusions hold.

THEOREM 9.6. $\mathrm{c\#P} \cup \mathrm{IF_t} \subseteq \mathrm{CL\#P} \subseteq \#\mathrm{P}$.

Proof. The inclusions $\mathrm{c\#P} \subseteq \mathrm{CL\#P}$ and $\mathrm{CL\#P} \subseteq \#\mathrm{P}$ are trivial. It remains to prove the inclusion $\mathrm{IF_t} \subseteq \mathrm{CL\#P}$. Choose $f \in \mathrm{IF_t}$ via a total p-order $A \in \mathrm{P}$ having polynomial-time adjacency checks, functions $b, t \in \mathrm{FP}$, and a polynomial p that witnesses that A is a p-order. We may without loss of generality assume that p is monotonic. For each $x \in \Sigma^*$, let $S_x = \{x0^{p(|x|)-|y|}1y0 \mid y \leq_A x\}$. Define A' as follows. Generally, A' corresponds to the lexicographical order on Σ^*, except that, for every $x \in \Sigma^*$, the interval between $x0^{p(|x|)+2}$ and $x1^{p(|x|)+2}$ is defined as follows.

- First come all strings in S_x, such that, for any strings $x0^{p(|x|)-|y_1|}1y_10$, $x0^{p(|x|)-|y_2|}1y_20 \in S_x$, let $x0^{p(|x|)-|y_1|}1y_10 \leq_{A'} x0^{p(|x|)-|y_2|}1y_20$ if and only if $y_1 \leq_A y_2$.
- Next come all the strings not in S_x, in lexicographical order.

We claim that A' is a total, polynomial-time computable p-order having efficient adjacency checks. Clearly, A' is total. Also, it is clear that, for any $s \in \Sigma^*$, it is possible to determine in polynomial time whether there is an $x \in \Sigma^*$ such that $s \in S_x$. It follows by this and by the definition of A that A' is polynomial-time computable. We claim that A' has efficient adjacency checks. For any $x \in \Sigma^*$, the lexicographically smallest element in S_x is $x0^{p(|x|)-|s_A|}1s_A0$, where $s_A \in \Sigma^*$ is the smallest element in the ordering imposed by A, and the lexicographically largest element is $x0^{p(|x|)-|x|}1x0$. If $x0^{p(|x|)-|y_1|}1y_10, x0^{p(|x|)-|y_2|}1y_20 \in S_x$, then $x0^{p(|x|)-|y_1|}1y_10 \prec_{A'} x0^{p(|x|)-|y_2|}1y_20$ if and only if $y_1 \prec_A y_2$ (this is true because, for every $y \in \Sigma^*$ such that $y \leq_A x$, it holds

[1] As we do in many places, we take it here as tacitly clear that the length of a path is its number of nondeterministic guesses, all of which in this model must be binary guesses. Note also that in the context of our model "All computation paths of M on x have length exactly $p(|x|)$" certainly implies that M is balanced in the sense defined at the start of this section: It will have every (and only) path corresponding to a guess sequence from $\{0,1\}^{p(|x|)}$. We mention, however, that we do not require a machine to make nondeterministic guesses at each step.

that $x0^{p(|x|)-|y|}1y0 \in S_x$; and thus, for such y_1 and y_2, it is impossible for some string longer than $p(|x_0|)$ to be "wedged between" them). The lexicographically smallest element not in S_x is $x0^{p(|x|)+2}$ and the largest is $x1^{p(|x|)+2}$. For any $w_1, w_2 \in \Sigma^*$ and $b_1, b_2 \in \{0,1\}$ such that both w_1b_1 and w_2b_2 are lexicographically between $x0^{p(|x|)+2}$ and $x1^{p(|x|)+2}$ but neither is in S_x, $w_1b_1 \prec_A w_2b_2$ if and only if $(w_1b_1 \prec_{\text{lex}} w_2b_2)$ or $(w_1b_1 \not\prec_{\text{lex}} w_2b_2$ and $b_1 = b_2 = 1$ and $w_1 \prec_{\text{lex}} w_2$ and $w_20 \in S_x)$. All other cases are handled in the way obvious from the above, e.g., for any $w_1, w_2 \in \Sigma^*$ and $b_1, b_2 \in \{0,1\}$ such that both of w_1b_1 and w_2b_2 are lexicographically between $x0^{p(|x|)+2}$ and $x1^{p(|x|)+2}$, and exactly one of them—say w_1b_1—is in S_x, the above makes it clear that $w_1b_1 \prec_{A'} w_2b_2$ exactly if $w_1b_1 = x0^{p(|x|)-|x|}1x0$ and $w_2b_2 = x0^{p(|x|)+2}$.

Define M to be a Turing machine that, on input $x \in \Sigma^*$, guesses a string $w \in \Sigma^{p(|t(x)|)+2}$. If $t(x)w \notin S_{t(x)}$, then M rejects. Otherwise, M accepts if and only if $t(x)0^{p(|t(x)|)-|b(x)|}1b(x)0 <_{A'} t(x)w <_{A'} t(x)0^{p(|t(x)|)-|t(x)|}1t(x)0$. Clearly, M runs in polynomial time and has computation paths of length exactly $p(t(|x|)) + 2$. Also, the number of accepting paths of M on x equals $f(x)$. By construction, the set of accepting computation paths of M on x is an A'-cluster. Thus, $f \in \text{CL\#P}$. □

From Proposition 9.4 and Theorem 9.6, it is clear that CL#P is different from both c#P and IF_t unless some surprising complexity class collapses occur. In particular, the following holds.

COROLLARY 9.7.

1. *If* c#P = CL#P, *then* UP = PP.
2. *If* IF_t = CL#P, *then* P = UP.

Nonetheless, when considering only polynomially bounded functions, c#P and CL#P do coincide.

THEOREM 9.8. c#P ∩ PolyBounded = CL#P ∩ PolyBounded.

Proof. The inclusion "⊆" is immediate. For the inclusion "⊇," choose $f \in \text{CL\#P}$ via a nondeterministic polynomial-time Turing machine M, a polynomial p, and a length-respecting total order A having efficient adjacency checks, all three of which have the properties and behaviors described in Definition 9.5. Recall that all accepting paths of M on any input x will be of length $p(|x|)$. Let q be a polynomial such that, for all $x \in \Sigma^*$, $f(x) \le q(|x|)$. We now will define a nondeterministic polynomial-time Turing machine N that almost-uniquely computes f in the sense of Definition 9.2. Define N to be a Turing machine that, on input $x \in \Sigma^*$, does the following.

1. If ϵ is an accepting path of $M(x)$, then accept and output 1.
2. N nondeterministically guesses strings $y, z \in \Sigma^{p(|x|)}$, $y' \in \Sigma^{p(|x|)-1} \cup \Sigma^{p(|x|)}$, and $z' \in \Sigma^{p(|x|)} \cup \Sigma^{p(|x|)+1}$.
3. N checks whether all of the following hold.
   (a) $y' \prec_A y$ and $z \prec_A z'$.
   (b) $y' \notin acc_M(x)$.
   (c) $z' \notin acc_M(x)$.
   (d) $y \in acc_M(x)$ and $z \in acc_M(x)$.
4. If (3) does not hold, then N rejects, otherwise if $y = z$, N accepts and outputs 1.
5. If (3) does hold and $y \neq z$, then N proceeds as follows.
   (a) N nondeterministically guesses an integer r with $0 \le r \le q(|x|) - 2$.
   (b) N nondeterministically guesses r strings $v_1, \ldots, v_r \in \Sigma^{p(|x|)}$.
   (c) N checks whether $y \prec_A v_1 \prec_A v_2 \prec_A \cdots \prec_A v_r \prec_A z$.
   (d) If (5c) does not hold, then N rejects. Otherwise, N accepts and outputs $r + 2$.

N is a nondeterministic polynomial-time Turing machine that, on each input, has one accepting path if $f(x) > 0$ and no accepting paths if $f(x) = 0$. If $f(x) > 0$, then N on x outputs $f(x)$ on its accepting path. Thus, N almost-uniquely computes f, and so by Proposition 9.3.1 $f \in \text{c\#P}$. □

For a class $\mathcal{F}$ of functions, let $\exists! \cdot \mathcal{F}$ be the class of all sets L for which there exists a function $f \in \mathcal{F}$ such that, for all x, $x \in L \Leftrightarrow f(x) = 1$.

THEOREM 9.9.

1. $\exists! \cdot \text{IF}_\text{p} = \text{coNP}$.
2. $\exists! \cdot \text{c\#P} = \exists! \cdot \text{CL\#P} = \text{UP}$.

Proof. For (1), $\text{coNP} \subseteq \exists! \cdot \text{IF}_\text{p}$ follows from Corollary 5.4 and the observation that any language in coNP is also (via considering the NP machine for the language's complement but with one extra accepting path added on each input) in $\exists! \cdot (\#\text{P} \cap \text{Nonzero})$. To see $\exists! \cdot \text{IF}_\text{p} \subseteq \text{coNP}$, choose $L \in \exists! \cdot \text{IF}_\text{p}$, via $f \in \text{IF}_\text{p}$. Let boundary functions $b, t \in \text{FP}$ and partial, polynomial-time computable p-order A having efficient adjacency checks witness that $f \in \text{IF}_\text{p}$. Let M be a nondeterministic polynomial-time Turing machine that, on input x, (i) guesses $y, z \in \Sigma^*$ such that $y \neq z$ and (ii) accepts if $b(x) \prec_A t(x) \vee (b(x) <_A y <_A t(x) \wedge b(x) <_A z <_A t(x))$. It is easy to see that M accepts $\overline{L}$, thus $L \in \text{NP}$.

For (2), $\text{UP} \subseteq \exists! \cdot \text{c\#P}$ is obvious. To see that $\exists! \cdot \text{CL\#P} \subseteq \text{UP}$, choose $L \in \exists! \cdot \text{CL\#P}$. Thus there exists a function $f \in \text{CL\#P}$ such that, for all x, $x \in L \Leftrightarrow f(x) = 1$. Let M be a machine that computes f via total order A having efficient adjacency checks and polynomial p (where M, A, and p are in the sense of Definition 9.5). Recall that all accepting paths of $M(x)$ are of length $p(x)$. Let N be a nondeterministic polynomial-time Turing machine that, on input x, guesses strings $y \in \Sigma^{p(|x|)}$ and $x, z \in \Sigma^{p(|x|)-1} \cup \Sigma^{p(|x|)} \cup \Sigma^{p(|x|)+1}$, and accepts if and only if all the following hold.

1. $y \prec_A z \wedge y \in acc_M(x) \wedge (w \prec_A y \vee w = y = \epsilon)$.
2. $w \notin acc_M(x) \vee w = y = \epsilon$.
3. $z \notin acc_M(x)$.

Clearly, N has on any input at most one accepting path and N accepts L. □

The next result shows that CL#P is probably not powerful enough to capture #P.

THEOREM 9.10. *If* $\text{CL\#P} = \#\text{P}$, *then* $\text{UP} = \text{PH}$.

Proof. Using Theorem 5.2 and both parts of Theorem 9.9, we have $\text{coNP} \subseteq \exists! \cdot \#\text{P} = \exists! \cdot \text{CL\#P} = \text{UP}$. □

On the other hand, proving CL#P to be different from #P is at least as hard as proving that $\text{P} \neq \text{NP}$ and $\text{UP} \neq \text{PP}$.

PROPOSITION 9.11. *If* $\text{P} = \text{NP}$ *or* $\text{UP} = \text{PP}$, *then* $\text{CL\#P} = \#\text{P}$.

Proof. Suppose $\text{UP} = \text{PP}$. Then by Proposition 9.3.4 $\text{c\#P} = \#\text{P}$, and so (see Theorem 9.6) $\text{CL\#P} = \#\text{P}$. Suppose that $\text{P} = \text{NP}$. Then by Theorem 5.10 it holds that $\text{IF}_\text{t} = \#\text{P}$, and so (see Theorem 9.6) $\text{CL\#P} = \#\text{P}$. □

Unfortunately, the necessary and sufficient conditions we have obtained for the equality of #P and CL#P differ, i.e., they do not yield a complete characterization. However, if we consider polynomially bounded functions, then such a complete characterization can be established in terms of the classes UP [Val76] and Few [CH90] (see section 2 for a review of their definitions). Note that $\text{UP} = \text{Few} \Leftrightarrow \text{UP} = \text{coUP} = \text{FewP} = \text{Few}$ and so in light of Theorem 9.12 we easily have that $\text{CL\#P} \cap \text{PolyBounded} = \#\text{P} \cap \text{PolyBounded}$ implies $\text{UP} = \text{coUP} = \text{FewP}$.

THEOREM 9.12. $\text{CL\#P} \cap \text{PolyBounded} = \#\text{P} \cap \text{PolyBounded}$ *if and only if* $\text{UP} = \text{Few}$.

Proof. [⇒]: Suppose that $L \in \text{Few}$ via a function $f \in \#\text{P}$, a set $B \in \text{P}$, and a polynomial p such that, for all x, $f(x) \leq p(|x|)$, and $x \in L \Leftrightarrow (x, 1^{f(x)}) \in B$. Let $g(x) =_{\text{def}} 1+f(x)$. Then $g \in \#\text{P}$, and g is polynomially bounded. From our hypothesis and Theorem 9.8, we obtain $g \in \text{c}\#\text{P}$. Since $g(x) > 0$, by Proposition 9.3.3 we have that $g \in \text{UPSV}_{\text{t}}$ via some nondeterministic polynomial-time (function-computing) Turing machine M whose behavior is UPSV_{t}-like. Define N to be a Turing machine that, on input x, nondeterministically guesses a computation path y of M on input x, simulates M on input x along computation path y, and accepts (on its current path) if and only if y is an accepting path with output z satisfying $(x, 1^{z-1}) \in B$. Clearly, N is a nondeterministic polynomial-time Turing machine with at most one accepting path on each input. Furthermore, it holds that N on x has an accepting computation path if and only if $(x, 1^{f(x)}) \in B$. This gives $L \in \text{UP}$.

[⇐]: Let f be any polynomially bounded #P function. Define $A =_{\text{def}} \{(x, 1^y) \mid y \leq f(x)\}$. Note that $A \in \text{Few}$. So by our hypothesis $A \in \text{UP}$. Indeed, since Few is closed under complementation and Few = UP by hypothesis, $A \in \text{UP} \cap \text{coUP}$. Via binary search using A as an oracle, we can compute f in polynomial time. That is, f is in $\text{FP}^{\text{UP}\cap\text{coUP}} = \text{UPSV}_{\text{t}} \subseteq \text{c}\#\text{P}$. Thus, $\text{CL}\#\text{P} \cap \text{PolyBounded} = \#\text{P} \cap \text{PolyBounded}$. □

From Corollary 9.7, we know that CL#P and c#P probably are different classes. However, under the ∃ operator the difference disappears, since both are mapped to UP. (Recall that Proposition 9.4.1 established $\exists \cdot \text{c}\#\text{P} = \text{UP}$.)

THEOREM 9.13. $\exists \cdot \text{CL}\#\text{P} = \text{UP}$.

Proof. The inclusion $\text{UP} \subseteq \exists \cdot \text{CL}\#\text{P}$ is immediate from Proposition 9.4.1 and the fact that $\text{c}\#\text{P} \subseteq \text{CL}\#\text{P}$. To show the inclusion $\exists \cdot \text{CL}\#\text{P} \subseteq \text{UP}$, choose an arbitrary $L \in \exists \cdot \text{CL}\#\text{P}$. Let $L \in \exists \cdot \text{CL}\#\text{P}$ via some function $f \in \text{CL}\#\text{P}$ with $x \in L \Leftrightarrow f(x) > 0$. Let $f \in \text{CL}\#\text{P}$ be witnessed (in the sense of the M, p, and A of Definition 9.5) by some Turing machine M, polynomial p, and total order A with efficient adjacency checks. Define N to be a Turing machine that, on input $x \in \Sigma^*$, does the following.

1. N nondeterministically guesses $z \in \Sigma^{p(|x|)}$ and $z' \in \Sigma^{p(|x|)} \cup \Sigma^{p(|x|)+1}$.
2. N checks whether each of the following conditions holds.
   (a) $z \prec_A z'$.
   (b) $z \in acc_M(x)$.
   (c) $z' \notin acc_M(x)$.
3. N accepts if and only if 2 holds.

Clearly, N runs in polynomial time and always has at most one accepting path. Also, it holds that $\#acc_N(x) = 1 \Leftrightarrow x \in L$. Thus, $L \in \text{UP}$. □

It is known that c#P is not closed under increment unless UP = coUP [Kos99]. We note that CL#P displays the same behavior.

THEOREM 9.14. *If* CL#P *is closed under increment, then* UP = coUP.

Proof. Observe that $\text{co}(\exists \cdot \mathcal{F}) \subseteq \exists! \cdot (\mathcal{F} + 1)$ is true for every class $\mathcal{F}$ of total functions, where $\mathcal{F} + 1$ denotes $\{g \mid (\exists f \in \mathcal{F})(\forall x)[g(x) = f(x) + 1]\}$. Thus by our hypothesis and Theorem 9.13 we have $\text{coUP} = \text{co}(\exists \cdot \text{CL}\#\text{P}) \subseteq \exists! \cdot (\text{CL}\#\text{P} + 1) \subseteq \exists! \cdot \text{CL}\#\text{P} = \text{UP}$. □

As a corollary, we obtain that CL#P is incomparable to IF_{p} unless some unexpected complexity class collapse occurs.

COROLLARY 9.15.

1. *If* $\text{CL}\#\text{P} \subseteq \text{IF}_{\text{p}}$, *then* P = UP.
2. *If* $\text{IF}_{\text{p}} \subseteq \text{CL}\#\text{P}$, *then* UP = PH.

Proof. Regarding (1), from our hypothesis and Theorem 9.13 we have UP =

$\exists \cdot \mathrm{CL\#P} \subseteq \exists \cdot \mathrm{IF_p} = \mathrm{P}$. To verify (2), observe that from our hypothesis, Theorem 9.9.1, and Theorem 9.13 we obtain $\mathrm{coNP} \subseteq \exists! \cdot \mathrm{IF_p} \subseteq \exists! \cdot \mathrm{CL\#P} = \mathrm{UP}$. □

10. Conclusion and open problems. We introduced interval size functions over p-orders and used them to provide an alternate definition of #P as the set of all interval size functions over polynomial-time decidable p-orders. We also introduced the classes $\mathrm{IF_p}$ and $\mathrm{IF_t}$, the interval size functions over partial and total polynomial-time computable p-orders with efficient adjacency checks. We proved that $\mathrm{IF_p}$ is the class of all functions in #P whose support is in P. We also proved that $\mathrm{IF_t}$ - FP = #P - FP and $\mathrm{IF_p}$ - $\mathcal{O}(1)$ = #P - $\mathcal{O}(1)$, but that $\mathrm{IF_p}$ = #P if and only if P = NP, and that $\mathrm{IF_t} = \mathrm{IF_p}$ only if UP = PH.

We also introduced the classes $\mathrm{IF_p^*}$ and $\mathrm{IF_t^*}$, the interval size functions over partial and total p-orders with efficient adjacency checks. We proved that $\exists \cdot \mathrm{IF_t^*} = \exists \cdot \mathrm{IF_t^*} =$ PSPACE.

Finally, we introduced CL#P, the set of all functions that count the number of accepting paths of polynomial-time cluster machines whose underlying orders are total and have efficient adjacency checks, and we studied the relationship between CL#P and the previously studied cluster computing class c#P.

Reviewing all the results on the interval size function classes $\mathrm{IF_p}$, $\mathrm{IF_p^*}$, $\mathrm{IF_t}$, and $\mathrm{IF_t^*}$, it seems that we have a good understanding of the computational power of the classes $\mathrm{IF_p}$, $\mathrm{IF_p^*}$, and $\mathrm{IF_t^*}$. Regarding the class $\mathrm{IF_t}$, we commend as an open issue obtaining an understanding of the class $\mathrm{IF_t}$ - $\mathcal{O}(1)$, which can be loosely considered to be a kind of "total order" #P.

In section 8, we showed that #MONSAT is complete for $\mathrm{IF_t}$ under Turing reductions. Valiant showed that #2CNFMONSAT is complete for #P under Turing reductions (and thus of course #MONSAT is complete for #P under Turing reductions). In light of this, a referee suggested as interesting open issues such questions as the following: Can one more broadly determine which #P-complete problems fall in $\mathrm{IF_t}$ and which fall in $\mathrm{IF_p}$? And what can one say about the downward closure, under various reductions, of #MONSAT, of $\mathrm{IF_t}$, and of $\mathrm{IF_p}$? In particular, what reductions are sufficiently restrictive as to leave $\mathrm{IF_t}$ and/or $\mathrm{IF_p}$ closed downward under the reductions, and relatedly, which reductions are sufficiently restrictive as to have the class of sets reducing to #MONSAT under the reductions be a subset of $\mathrm{IF_t}$ and/or $\mathrm{IF_p}$?

Acknowledgments. We are grateful to J. Rothe, H. Spakowski, and M. Thakur for proofreading an earlier draft of this paper, and to E. Hemaspaandra and K.-J. Lange for helpful discussions. We also thank the anonymous referees for their careful, helpful reviews and for comments that improved the presentation of the paper.

REFERENCES

[AKS04] M. Agrawal, N. Kayal, and N. Saxena, *PRIMES is in P*, Ann. of Math. (2), 160 (2004), pp. 781–793.

[CH90] J.-Y. Cai and L. Hemachandra, *On the power of parity polynomial time*, Math. Systems Theory, 23 (1990), pp. 95–106.

[Coo71] S. Cook, *The complexity of theorem-proving procedures*, in Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, ACM, New York, 1971, pp. 151–158.

[FFK94] S. Fenner, L. Fortnow, and S. Kurtz, *Gap-definable counting classes*, J. Comput. System Sci., 48 (1994), pp. 116–148.

[GHJY91] J. Goldsmith, L. A. Hemachandra, D. Joseph, and P. Young, *Near-testable sets*, SIAM J. Comput., 20 (1991), pp. 506–523.

[Gil77] J. Gill, *Computational complexity of probabilistic Turing machines*, SIAM J. Comput., 6 (1977), pp. 675–695.

[GS88] J. Grollmann and A. L. Selman, *Complexity measures for public-key cryptosystems*, SIAM J. Comput., 17 (1988), pp. 309–335.

[GS91] A. V. Goldberg and M. Sipser, *Compression and ranking*, SIAM J. Comput., 20 (1991), pp. 524–536.

[GS92] M. Grigni and M. Sipser, *Monotone complexity*, in Boolean Function Complexity, London Math. Soc. Lecture Note Ser. 169, M. Paterson, ed., Cambridge University Press, Cambridge, UK, 1992, pp. 57–75.

[GW83] H. Galperin and A. Wigderson, *Succinct representations of graphs*, Inform. and Control, 56 (1983), pp. 183–198.

[HHK06] L. Hemaspaandra, C. Homan, and S. Kosub, *Cluster computing and the power of edge recognition*, in Proceedings of the Third Annual Conference on Theory and Applications of Models of Computation, Lecture Notes in Computer Sci. 3959, Springer-Verlag, Berlin, 2006, pp. 283–294.

[HHKW05] L. Hemaspaandra, C. Homan, S. Kosub, and K. Wagner, *The Complexity of Computing the Size of an Interval*, Technical report TR-856, University of Rochester, Department of Computer Science, Rochester, NY, February 2005, revised March 2005.

[HHW05] E. Hemaspaandra, L. Hemaspaandra, and O. Watanabe, *The Complexity of Kings*, Technical report TR-870, University of Rochester, Department of Computer Science, Rochester, NY, 2005.

[HKW01] L. Hemaspaandra, S. Kosub, and K. Wagner, *The complexity of computing the size of an interval*, in Proceedings of 28th International Colloquium on Algorithms, Languages and Programming, Lecture Notes in Computer Sci. 2076, Springer-Verlag, Berlin, 2001, pp. 1040–1051.

[HMU01] J. Hopcroft, R. Motwani, and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 2nd ed., Addison-Wesley, Boston, 2001.

[HO02] L. Hemaspaandra and M. Ogihara, *The Complexity Theory Companion*, Springer-Verlag, Berlin, 2002.

[HVW96] U. Hertrampf, H. Vollmer, and K. Wagner, *On balanced versus unbalanced computation trees*, Math. Systems Theory, 29 (1996), pp. 411–421.

[HW00] H. Hempel and G. Wechsung, *The operators min and max on the polynomial hierarchy*, Internat. J. Found. Comput. Sci., 11 (2000), pp. 315–342.

[Ko83] K. Ko, *On self-reducibility and weak P-selectivity*, J. Comput. System Sci., 26 (1983), pp. 209–221.

[Kos99] S. Kosub, *A note on unambiguous function classes*, Inform. Process. Lett., 72 (1999), pp. 197–203.

[KSTT92] J. Köbler, U. Schöning, S. Toda, and J. Torán, *Turing machines with few accepting computations and low sets for* PP, J. Comput. System Sci., 44 (1992), pp. 272–286.

[Lad89] R. E. Ladner, *Polynomial space counting problems*, SIAM J. Comput., 18 (1989), pp. 1087–1097.

[Lev75] L. Levin, *Universal sequential search problems*, Probl. Inf. Transm., 9 (1975), pp. 265–266.

[MP79] A. Meyer and M. Paterson, *With What Frequency are Apparently Intractable Problems Difficult?*, Technical report MIT/LCS/TM-126, MIT, Laboratory for Computer Science, Cambridge, MA, 1979.

[MS72] A. Meyer and L. Stockmeyer, *The equivalence problem for regular expressions with squaring requires exponential time*, in Proceedings of the 13th Symposium on Switching and Automata Theory, IEEE Press, Los Alamitos, CA, 1972, pp. 125–129.

[NT05] A. Nickelsen and T. Tantau, *The complexity of finding paths in graphs with bounded independence number*, SIAM J. Comput., 34 (2005), pp. 1176–1195.

[OH93] M. Ogiwara and L. Hemachandra, *A complexity theory of feasible closure properties*, J. Comput. System Sci., 46 (1993), pp. 295–325.

[OTTW96] M. Ogihara, T. Thierauf, S. Toda, and O. Watanabe, *On closure properties of* #P *in the context of* PF $\circ$ #P, J. Comput. System Sci., 53 (1996), pp. 171–179.

[PY86] C. Papadimitriou and M. Yannakakis, *A note on succinct representations of graphs*, Inform. and Control, 71 (1986), pp. 181–185.

[Sim75] J. Simon, *On Some Central Problems in Computational Complexity*, Ph.D. thesis, Cornell University, Ithaca, NY, 1975.

[Sto77] L. Stockmeyer, *The polynomial-time hierarchy*, Theoret. Comput. Sci., 3 (1977), pp. 1–22.

[Tan01] T. Tantau, *A Note on the Complexity of the Reachability Problem for Tournaments*, Technical report TR01-092, Electronic Colloquium on Computational Complexity, http://www.eccc.uni-trier.de/eccc/ (2001).

[Val76] L. Valiant, *Relative complexity of checking and evaluation*, Inform. Process. Lett., 5 (1976), pp. 20–23.

[Val79] L. G. Valiant, *The complexity of enumeration and reliability problems*, SIAM J. Comput., 8 (1979), pp. 410–421.

[VW95] H. Vollmer and K. Wagner, *Complexity classes of optimization functions*, Inform. and Comput., 120 (1995), pp. 198–219.

[Wag84] K. Wagner, *The complexity of problems concerning graphs with regularities*, in Proceedings of the 11th Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Sci. 176, Springer-Verlag, Berlin, 1984, pp. 544–552.

[Wag86] K. Wagner, *The complexity of combinatorial problems with succinct input representations*, Acta Inform., 23 (1986), pp. 325–356.

SIAM J. COMPUT.
Vol. 36, No. 5, pp. 1301–1328

CHOSEN-CIPHERTEXT SECURITY FROM IDENTITY-BASED ENCRYPTION*

DAN BONEH†, RAN CANETTI‡, SHAI HALEVI‡, AND JONATHAN KATZ§

Abstract. We propose simple and efficient CCA-secure public-key encryption schemes (i.e., schemes secure against adaptive chosen-ciphertext attacks) based on any identity-based encryption (IBE) scheme. Our constructions have ramifications of both theoretical and practical interest. First, our schemes give a new paradigm for achieving CCA-security; this paradigm avoids "proofs of well-formedness" that have been shown to underlie previous constructions. Second, instantiating our construction using known IBE constructions we obtain CCA-secure encryption schemes whose performance is competitive with the most efficient CCA-secure schemes to date. Our techniques extend naturally to give an efficient method for securing IBE schemes (even hierarchical ones) against adaptive chosen-ciphertext attacks. Coupled with previous work, this gives the first efficient constructions of CCA-secure IBE schemes.

Key words. identity-based encryption, public-key encryption, chosen-ciphertext security

AMS subject classifications. 94A60, 68P25

DOI. 10.1137/S009753970544713X

1. Introduction.

Security against adaptive chosen-ciphertext attacks [49, 50, 28, 2] is the de facto level of security required for public-key encryption schemes used in practice. This security notion is appropriate for encryption schemes used in the presence of *active* attackers who may potentially modify messages in transit, and schemes proven secure with respect to this notion may be securely "plugged in" to higher-level protocols deployed in unauthenticated networks that were designed and analyzed under the idealized assumption of "secure channels" (see, e.g., [15, 19]). Unfortunately, only a handful of approaches are known for constructing encryption schemes that meet this notion of security. In this work we put forward a new approach for constructing such schemes.

1.1. Background.

Security for public-key encryption was first defined formally by Goldwasser and Micali [38]. Their notion of *semantic security*, roughly speaking, requires that observation of a ciphertext does not enable an adversary to compute anything about the underlying plaintext message that it could not have computed on its own (i.e., prior to observing the ciphertext); this should hold even if the adversary has some a priori information about the message. Goldwasser and Micali (see also [47, 35, 36]) proved that semantic security is equivalent to the notion of *indistinguishability* that requires (roughly) the following: for any two messages, given a "challenge" ciphertext C that is an encryption of one of these messages it is infeasible to determine (with any noticeable advantage over a random guess) which message was

*Received by the editors March 3, 2005; accepted for publication (in revised form) April 29, 2006; published electronically December 21, 2006. This work appeared in preliminary form as *Chosen-ciphertext security from identity-based encryption*, in Eurocrypt 2004 [18] and *Improved efficiency for CCA-secure cryptosystems built using identity-based encryption*, in RSA-CT 2005 [12].
http://www.siam.org/journals/sicomp/36-5/44713.html

†Department of Computer Science, Stanford University, Stanford, CA 94305 (dabo@cs.stanford.edu). This author's research was supported by NSF grant CNS-0331640.

‡IBM T. J. Watson Research Center, Hawthorne, NY 10532 (canetti@watson.ibm.com, shaih@alum.mit.edu). The research of the second author was supported by NSF grant CNS-0430450.

§Department of Computer Science, University of Maryland, College Park, MD 20742 (jkatz@cs.umd.edu). This author's research was supported by NSF grant CNS-0310751.

actually encrypted. Because these definitions imply security even when the adversary can mount a *chosen-plaintext attack* to obtain encryptions of messages of its choice, we will refer to these notions using the commonly accepted term "CPA-security."

CPA-security does not guarantee any security against *chosen-ciphertext attacks* by which an adversary may obtain decryptions of ciphertexts of its choice (note that attacks of this sort may arise in practice [5, 53]). Indistinguishability-based definitions appropriate for this setting were given by Naor and Yung [49] and Rackoff and Simon [50]. Naor and Yung consider a *nonadaptive* chosen-ciphertext attack in which the adversary may request decryptions only *before* it obtains the challenge ciphertext. Rackoff and Simon define the stronger notion of security against *adaptive* chosen-ciphertext attacks whereby the adversary may request decryptions even after seeing the challenge ciphertext, under the natural limitation that the adversary may not request decryption of the challenge ciphertext itself. (We will refer to the latter notion as "CCA-security.") See [28, 2, 36] for further discussion of these definitions. Extensions of semantic security to the case of chosen-ciphertext attacks were considered in [36, 37, 55], where it is shown that, as in the case of CPA-security, these definitions are equivalent to the indistinguishability-based ones.

As we have already mentioned, CCA-security is now the de facto level of security for public-key encryption due to its numerous advantages (some of which were summarized earlier). Unfortunately, only a handful of public-key encryption schemes have been proven secure against adaptive chosen-ciphertext attacks without resorting to heuristics such as the random oracle methodology [3], a controversial and problematic approach [16].

In fact, prior to this work only two approaches were known for constructing CCA-secure cryptosystems. The first follows the paradigm introduced by Naor and Yung [49] to achieve nonadaptive chosen-ciphertext security, later extended to the case of adaptive chosen-ciphertext security by Dolev, Dwork, and Naor [28] and Sahai [51]. This technique uses as building blocks any CPA-secure public-key encryption scheme and any noninteractive zero-knowledge (NIZK) proof system for all of $\mathcal{NP}$ [6, 31]. Consequently, this approach can be based on general cryptographic assumptions [31]: specifically, the existence of enhanced trapdoor permutations [36, sect. C.4.1]. Encryption schemes resulting from this approach, however, are highly impractical because they employ generic NIZK proofs which in turn require a Karp reduction from the $\mathcal{NP}$ language of interest to some $\mathcal{NP}$-complete language. Thus, given the current state of the art, this approach serves as a feasibility result for the existence of CCA-secure cryptosystems based on general assumptions but does not lead to any practical constructions.

The second technique is due to Cramer and Shoup [21, 22] and is based on algebraic constructs with particular homomorphic properties (namely, those admitting "smooth hash proof systems" in the terminology of [22]). Algebraic constructs of the appropriate type are known to exist based on some specific number-theoretic assumptions [21, 22], including the decisional Diffie–Hellman (DDH) assumption. Other constructions relying on this technique have been given recently [32, 23, 42], leading to a number of practical schemes.

Interestingly, Elkind and Sahai have observed [29] that both the above approaches for constructing CCA-secure encryption schemes can be viewed as special cases of a single paradigm. In this paradigm one starts with a CPA-secure cryptosystem in which certain "ill-formed" ciphertexts are indistinguishable from honestly generated ciphertexts. A CCA-secure cryptosystem is then obtained by having the sender honestly generate a ciphertext using the underlying CPA-secure scheme, and then append

a "proof of well-formedness" (satisfying certain criteria) to this ciphertext. The NIZK proofs used by Sahai [51] as well as the smooth hash proof systems used by Cramer and Shoup [21, 22] are shown by Elkind and Sahai to satisfy the appropriate criteria.

1.2. Summary of our results. We propose two new approaches for constructing CCA-secure public-key encryption schemes based on any CPA-secure identity-based encryption (IBE) scheme[1] (see sections 2.1 and 3.1 for definitions of the latter notion). A number of IBE schemes based on specific number-theoretic assumptions are known [17, 7, 8, 56, 34]; thus, our techniques yield new constructions of CCA-secure encryption schemes based on these same assumptions.

From a theoretical perspective, our work offers new ways of constructing CCA-secure encryption schemes that do not use "proofs of well-formedness" and hence do not seem to fit within the Elkind–Sahai characterization mentioned above. From a practical perspective, we show that a specific instantiation of our construction yields a practical CCA-secure scheme with efficiency close to that of the best previous construction (cf. section 7). This efficient instantiation is based on the *decisional bilinear Diffie–Hellman (BDH) assumption*, described in section 7.3. Comparing this assumption to those used in prior constructions of CCA-secure encryption schemes, we note the following:

- The decisional BDH assumption seems incomparable to the assumption of enhanced trapdoor permutations that underlies the standard construction of generic NIZK as used in the schemes of [28, 51]. Nevertheless, the decisional BDH assumption is known to imply the existence of generic NIZK proof systems for all of $\mathcal{NP}$ [17, Appendix B] and hence was already known to imply CCA-secure encryption via [28, 51]. We stress again that the schemes shown here are orders of magnitude more efficient than schemes constructed via generic NIZK.
- The decisional BDH assumption implies the DDH assumption that underlies the schemes of [21, 42]. However, less efficient variants of our scheme can be proven secure using assumptions that are incomparable to the DDH assumption; see footnote 6 in section 7.3. In addition, our scheme lends itself to threshold encryption much more efficiently than previous schemes; see below.

Further extensions and applications. Both our approaches extend to give a transformation from any CPA-secure $(\ell+1)$-level hierarchical identity-based encryption (HIBE) scheme [41, 33] to a CCA-secure ℓ-level HIBE scheme (HIBE is described in section 3.2). In particular, applying our technique to any 2-level HIBE scheme gives a CCA-secure IBE scheme. Using this approach with known HIBE schemes [17, 7, 8, 56, 9] yields the first efficient constructions of CCA-secure IBE schemes.

Our first approach, when instantiated with an appropriate IBE scheme, serves as the basis for the first CCA-secure threshold encryption scheme with *noninteractive* decryption [7, 10]. (In a threshold encryption scheme [25] the secret key is shared among multiple servers, some fraction of whom must cooperate in order to decrypt a given ciphertext.) Security in this case crucially relies on specific properties of our construction and, in particular, on the feature that a certain class of "valid" ciphertexts can be efficiently recognized without knowledge of the global secret key; the reader is referred to [7, 10] for further discussion.

[1] Our constructions use other primitives, but these can all be constructed based on one-way functions which are in turn implied by CPA-secure encryption.

Our approaches are generic and can be used to construct a CCA-secure encryption scheme from an *arbitrary* IBE scheme. Extending our work, Boyen, Mei, and Waters [14] show that for some concrete IBE schemes (e.g., the one of Waters [56]) a more efficient and direct construction of a CCA-secure encryption scheme is possible.

1.3. Organization. In the following section, we provide an informal overview of IBE and HIBE as well as some high-level intuition regarding our techniques. Formal definitions of all relevant cryptographic notions (including IBE and CCA-secure public-key encryption) appear in section 3 and the appendix. The first of our transformations is discussed in section 4, and the second transformation is presented in section 5. We discuss the extension to HIBE in section 6.

The treatment in the above sections is generic and does not rely on any specific cryptographic assumptions. In section 7 we recall one specific number-theoretic assumption under which an IBE scheme is known to exist, describe a concrete instantiation of our construction based on this assumption, and compare the efficiency of the resulting CCA-secure encryption scheme to the most efficient such construction that was previously known (namely, the Kurosawa–Desmedt variant [42] of the Cramer–Shoup encryption scheme [21]).

2. Overview of our techniques.

2.1. Identity-based encryption. Before sketching our constructions, we first recall the notion of IBE as introduced by Shamir [52]. Informally, an IBE scheme is a public-key encryption scheme in which any string (i.e., identity) can serve as a public key. In more detail, a trusted authority called a *private-key generator* (PKG) is assumed to initialize the system by running a key-generation algorithm to generate "master" public and secret keys. The master public key PK is published, while the PKG stores the master secret key. Given the master secret key and an arbitrary string ID (viewed as the identity of a party in the system), the PKG can derive a "personal secret key" SK_{ID} and give it to this party. Any sender can encrypt a message for this party using only the master public key PK and the string ID; we denote such encryption by $\mathcal{E}_{PK}(ID, \cdot)$. The resulting ciphertext can be decrypted using the personal secret key SK_{ID}, but the following extension of CPA-security is required to hold:

> For any two messages and any identity ID, given a challenge ciphertext C that is an encryption of one of these messages (with respect to ID) it is infeasible to determine (with any noticeable advantage over a random guess) which message was actually encrypted. *This should hold even if the adversary is given $SK_{ID'}$ for multiple identities $ID' \neq ID$ chosen adaptively by the adversary.*

The first formal definition of security for IBE was given by Boneh and Franklin [11]. In their definition, the adversary may choose the "target identity" (ID in the above) in an adaptive manner, based on the master public key PK and any keys $\{SK_{ID'}\}$ the adversary has obtained thus far; we call such schemes "fully secure." A weaker notion, proposed by Canetti, Halevi, and Katz [17] and called "selective-ID" security there, requires the adversary to specify the target identity *in advance*, before the master public key is published. Fully secure IBE schemes in the random oracle model were first demonstrated by Boneh and Franklin [11] and Cocks [20]. Canetti, Halevi, and Katz [17], building on earlier work of Gentry and Silverberg [33], constructed an IBE scheme satisfying selective-ID security in the standard model; more efficient constructions were given by Boneh and Boyen [7]. More recently, Boneh

and Boyen [8] have shown a fully secure IBE scheme in the standard model, and more efficient constructions were subsequently given by Waters [56] and Gentry [34].

Both our constructions of CCA-secure encryption from IBE require an IBE scheme satisfying only the weaker notion of selective-ID security. Our transformation from any CPA-secure $(\ell + 1)$-level HIBE scheme to a CCA-secure ℓ-level HIBE scheme preserves the level of security in the above sense; i.e., if the original scheme is fully secure then so is the derived scheme, but selective-ID security of the original scheme is sufficient for selective-ID security of the derived scheme.

2.2. Our techniques.

Our first construction. Given an IBE scheme, we construct a CCA-secure public-key encryption scheme as follows: The public key of the new scheme is the master public key PK of the IBE scheme and the secret key is the corresponding master secret key. To encrypt a message with respect to public key PK, the sender first generates a key-pair (vk, sk) for a strong[2] one-time signature scheme, and then encrypts the message with respect to the "identity" vk. The resulting ciphertext $C \leftarrow \mathcal{E}_{PK}(vk, m)$ is then signed using sk to obtain a signature σ. The final ciphertext consists of the verification key vk, the IBE ciphertext C, and the signature σ. To decrypt a ciphertext $\langle vk, C, \sigma \rangle$, the receiver first verifies the signature on C with respect to vk and outputs $\perp$ if the verification fails. Otherwise, the receiver derives the secret key SK_{vk} corresponding to the "identity" vk, and uses SK_{vk} to decrypt the ciphertext C using the underlying IBE scheme.

Security of the above scheme against adaptive chosen-ciphertext attacks can be informally understood as follows. Say a ciphertext $\langle vk, C, \sigma \rangle$ is *valid* if σ is a valid signature on C with respect to vk. Now consider a challenge ciphertext $c^* = \langle vk^*, C^*, \sigma^* \rangle$ given to the adversary. We may first notice that any valid ciphertext $c = \langle vk, C, \sigma \rangle$ submitted by the adversary to its decryption oracle (implying $c \neq c^*$) must, except with negligible probability, have $vk \neq vk^*$ by the strong security of the one-time signature scheme. The crux of the security proof is then to show that (selective-ID) security of the IBE scheme implies that obtaining the decryption of C does not help the adversary in deciding which message the ciphertext C^* corresponds to. Intuitively, this is because the adversary cannot guess the message corresponding to C^* with probability better than $1/2$ *even if it were given the secret key* SK_{vk}. (This is so since $vk \neq vk^*$, and C^* was encrypted for "identity" vk^* using an IBE scheme.) But giving SK_{vk} to the adversary only makes the adversary more powerful, since it could then decrypt C itself.

Our use of a strong one-time signature scheme to force the adversary's decryption queries to differ from the challenge ciphertext in a specific way is reminiscent of prior work in the context of CCA-security [28, 51]. The key difference is that prior work used the verification key vk to implement "unduplicatable set selection" (cf. [51]) which requires $\Theta(k)$ invocations of some underlying encryption scheme, where k is the security parameter. Furthermore, prior work also required some sort of "proof of consistency" for the resulting ciphertext, leading (as described earlier) to an impractical scheme. In contrast, our construction gives a CCA-secure encryption scheme with relatively minimal overhead as compared to the original IBE scheme.

We note also independent work of MacKenzie, Reiter, and Yang [45], who introduce a weaker notion of CCA-secure encryption and use essentially the same con-

[2] A "strong" signature scheme has the property that it is infeasible to create a new, valid signature even for a previously signed message. A formal definition is given in the appendix.

struction to convert any scheme satisfying their weaker definition into a full-fledged CCA-secure encryption scheme. Their work, however, shows only efficient realizations of schemes in the random oracle model.

We remark that if the signature scheme used is only unforgeable in the standard sense (rather than *strongly* unforgeable), we obtain an encryption scheme satisfying the slightly weaker notion of *replayable* CCA-security [19]. Also, a simple modification of the above construction gives an encryption scheme secure against *nonadaptive* chosen-ciphertext attacks [49, 28, 2] but with essentially no overhead as compared to the underlying IBE scheme. Namely, replace the verification key vk by a randomly chosen string $r \in \{0,1\}^{\omega(\log k)}$; the resulting ciphertext is simply $\langle r, C\rangle$, where C is encrypted with respect to the "identity" r. Since an adversary cannot guess in advance (with better than negligible probability) which r will be used for the challenge ciphertext, an argument similar to the above shows that this scheme is secure against nonadaptive chosen-ciphertext attacks.

Improving the efficiency. Focusing again on security against *adaptive* chosen-ciphertext attacks, the previous construction—although conceptually simple and efficient—does add noticeable overhead in practice to the underlying IBE scheme: encryption requires the sender to generate signing/verification keys and sign a message; the ciphertext length is increased by the size of a verification key plus the size of a signature; and decryption requires the receiver to perform a signature verification. Although one-time signatures are "easy" to construct in theory and are more efficient than full-fledged signatures (i.e., those which are strongly unforgeable under adaptive chosen-message attack), they still have their price.

- Known one-time signature schemes based on general one-way functions [43, 30] allow very efficient *signing*; key generation and signature verification, on the other hand, require $\Theta(k)$ evaluations of the one-way function and are relatively expensive. More problematic, perhaps, is that such schemes have very long public keys and/or signatures (with combined length $\Theta(k^2)$), resulting in very long ciphertexts in our construction above.
- One-time signature schemes can of course be based on number-theoretic assumptions (say, by adapting full-fledged signature schemes); this yields schemes whose computational cost for key generation, signing, and verifying is more expensive, but which (may) have the advantage of short(er) public keys and signatures.

Motivated by the above, we modify the previous construction by using a message authentication code (MAC) in place of a one-time signature scheme in the following way: the secret signing key is replaced by a secret MAC key r and the public verification key (which was used as the "identity" for the IBE scheme) is replaced by a commitment to r. In more detail, encryption of a message m is performed by first committing to a random MAC key r, resulting in a commitment com and a corresponding decommitment dec. The ciphertext is $\langle \mathsf{com}, C, \mathsf{tag}\rangle$, where C is an encryption of the "message" $m \circ \mathsf{dec}$ with respect to the "identity" com (i.e., $C \leftarrow \mathcal{E}_{PK}(\mathsf{com}, m \circ \mathsf{dec})$) and tag is a message authentication code computed on C using key r. Decryption of ciphertext $\langle \mathsf{com}, C, \mathsf{tag}\rangle$ is done in the natural way: the receiver first decrypts C with respect to "identity" com to obtain $m \circ \mathsf{dec}$, and then recovers r using com and dec. The receiver then tries to verify tag using key r, outputting m if verification succeeds and $\perp$ otherwise.

In fact, a weaker form of commitment than the standard one suffices for our purposes (cf. section 5.1); we refer to this weaker notion as "encapsulation." The

advantage of the former is that encapsulation schemes can potentially be more efficient than full-fledged commitment schemes.

The intuition for the security of this construction is quite similar to that discussed previously. Consider a challenge ciphertext $\langle \mathsf{com}^*, C^*, \mathsf{tag}^* \rangle$ that was constructed using MAC key r^*. As before, decryption queries that use a different "identity" $\mathsf{com} \neq \mathsf{com}^*$ are useless to the attacker due to the security of the underlying IBE scheme. For decryption queries $\langle \mathsf{com}^*, C, \mathsf{tag} \rangle$ that use the same "identity" we note that either (1) C decrypts to $m \circ \mathsf{dec}$, where dec is a valid decommitment of com^* to some $r \neq r^*$, or (2) the attacker was able to compute a valid tag on $C \neq C^*$ with respect to the MAC key r^*. The first case is easily shown to violate the binding property of the commitment scheme. The second case can be shown to violate the secrecy of the commitment scheme (i.e., the adversary learns something about r^* from com^*), the secrecy of the encryption (i.e., the adversary learns something about r^* from C^*), or the security of the MAC (i.e., the adversary generates a valid tag without learning anything about r^*).

The actual proof for this scheme is more difficult than for the previous case due to the fact that here C must be decrypted *before* validity of the ciphertext as a whole can be checked. We thus must be careful to avoid the seeming circularity which arises since the MAC key r is used to authenticate a string (namely, C) that depends on r (via dec).

The idea of using a MAC and a commitment to the key was suggested previously in the context of nonmalleable commitment (e.g., [26, 27]), but our application of this technique is *qualitatively* different precisely due to the apparent circularity (and the resulting complications to the proof) discussed above. In particular, in the context of nonmalleable commitment the MAC key can be revealed by the sender during the decommitment phase and hence the key is not used to authenticate a message which depends on itself. In contrast, here the MAC key must be transmitted to the receiver as part of the ciphertext. The idea of encapsulating the MAC key (rather than using full-fledged commitment), as well as the encapsulation scheme we propose, are new to this work.

3. Definitions. We use the standard definitions of public-key encryption schemes and their security against adaptive chosen-ciphertext attacks strong one-time signature schemes, and MACs. For convenience and to fix notation, we recall these definitions in the appendix. Our definitions of IBE and HIBE schemes have also appeared previously; however, since these definitions are less familiar yet are central to our work, we include the appropriate definitions in this section. Encapsulation schemes are defined in section 5.1. Our definitions and proofs are phrased with respect to uniform adversaries but can be easily extended to the nonuniform setting. We let "PPT" stand for "probabilistic polynomial-time."

If Σ is a set then Σ^n denotes the set of n-tuples of elements of Σ, with Σ^0 denoting the set containing only the empty tuple. Thus, using this notation, $\{0,1\}^n$ denotes the set of binary strings of length n. We also define $\Sigma^{<n} \stackrel{\text{def}}{=} \bigcup_{0 \leq i < n} \Sigma^i$ and $\Sigma^{\leq n} \stackrel{\text{def}}{=} \bigcup_{0 \leq i \leq n} \Sigma^i$.

3.1. Identity-based encryption. We begin by reviewing the functional definition of an IBE scheme [11].

DEFINITION 1. *An* identity-based encryption scheme *for identities of length n (where n is a polynomially bounded function) is a tuple of* PPT *algorithms* $(\mathsf{Setup}, \mathsf{Der}, \mathcal{E}, \mathcal{D})$ *such that the following hold:*

- *The randomized* setup algorithm Setup *takes as input a security parameter* 1^k. *It outputs a master public key* PK *and a master secret key* msk. *(We assume that* k *and* $n = n(k)$ *are implicit in* PK *and* msk.*)*
- *The (possibly randomized)* key-derivation algorithm Der *takes as input the master secret key* msk *and an identity* $ID \in \{0,1\}^n$. *It returns the corresponding decryption key* SK_{ID}. *We write* $SK_{ID} \leftarrow \mathsf{Der}_{\mathsf{msk}}(ID)$.
- *The randomized* encryption algorithm $\mathcal{E}$ *takes as input the master public key* PK, *an identity* $ID \in \{0,1\}^n$, *and a message* m *in some implicit*[3] *message space; it outputs a ciphertext* C. *We write* $C \leftarrow \mathcal{E}_{PK}(ID, m)$.
- *The (possibly randomized)* decryption algorithm $\mathcal{D}$ *takes as input an identity* ID, *an associated decryption key* SK_{ID}, *and a ciphertext* C. *It outputs a message* m *or the symbol* $\perp$ *(which is not in the message space). We write* $m \leftarrow \mathcal{D}_{SK_{ID}}(ID, C)$.

We require that for all (PK, msk) *output by* Setup, *all* $ID \in \{0,1\}^n$, *all* SK_{ID} *output by* $\mathsf{Der}_{\mathsf{msk}}(ID)$, *all* m *in the message space, and all* C *output by* $\mathcal{E}_{PK}(ID, m)$ *we have* $\mathcal{D}_{SK_{ID}}(ID, C) = m$.

We now define security for IBE. As mentioned in the introduction, the definition we give is weaker than that considered by Boneh and Franklin [11] and conforms to "selective-ID" security [17] where the "target" identity is selected by the adversary before the public key is generated.

DEFINITION 2. *An identity-based encryption scheme* Π *for identities of length* n *is* selective-ID secure against chosen-plaintext attacks *if the advantage of any* PPT *adversary* $\mathcal{A}$ *in the following game is negligible in the security parameter* k*:*

1. $\mathcal{A}(1^k)$ *outputs a "target" identity* $ID^* \in \{0,1\}^{n(k)}$.
2. $\mathsf{Setup}(1^k)$ *outputs* (PK, msk). *The adversary is given* PK.
3. *The adversary* $\mathcal{A}$ *may make polynomially many queries to an oracle* $\mathsf{Der}_{\mathsf{msk}}(\cdot)$, *except that it may not request a secret key corresponding to the target identity* ID^*.
   The adversary is allowed to query this oracle repeatedly using the same identity*; if* Der *is randomized, then a different secret key may possibly be returned each time.*
4. *At some point,* $\mathcal{A}$ *outputs two messages* m_0, m_1 *with* $|m_0| = |m_1|$. *A bit* b *is randomly chosen and the adversary is given a "challenge" ciphertext* $C^* \leftarrow \mathcal{E}_{PK}(ID^*, m_b)$.
5. $\mathcal{A}$ *may continue to query its oracle* $\mathsf{Der}_{\mathsf{msk}}(\cdot)$ *as above. Finally,* $\mathcal{A}$ *outputs a guess* b'.

We say that $\mathcal{A}$ succeeds *if* $b' = b$ *and denote the probability of this event by* $\Pr^{\mathsf{IBE}}_{\mathcal{A},\Pi}[\mathsf{Succ}]$. *The adversary's* advantage *is defined as* $\mathsf{Adv}^{\mathsf{IBE}}_{\mathcal{A},\Pi}(k) \stackrel{\text{def}}{=} \left|\Pr^{\mathsf{IBE}}_{\mathcal{A},\Pi}[\mathsf{Succ}] - 1/2\right|$.

The definition may be extended to take into account security against adaptive chosen-ciphertext attacks. In this case, the adversary additionally has access to an oracle $\widehat{\mathcal{D}}(\cdot)$ such that $\widehat{\mathcal{D}}(C)$ returns $\mathcal{D}_{SK_{ID^*}}(C)$, where SK_{ID^*} is the secret key associated with the target identity ID^* (computed using $\mathsf{Der}_{\mathsf{msk}}(ID^*)$).[4] The adversary has access to this oracle throughout the entire game but cannot submit the challenge ciphertext C^* to $\widehat{\mathcal{D}}$.

On deterministic versus randomized key derivation. For simplicity, when

[3] For example, the message space may consist of all strings of length $p(k)$, where p is polynomially bounded.

[4] Note that decryption queries for identities $ID \neq ID^*$ are superfluous, as $\mathcal{A}$ may make the corresponding Der query itself and thereby obtain SK_{ID}.

dealing with chosen-ciphertext security for IBE schemes we will assume that Der is *deterministic*. If Der is not deterministic, a definition of chosen-ciphertext security is complicated by the question of whether different invocations of the decryption oracle $\widehat{\mathcal{D}}$ should use the *same* secret key SK_{ID^*} (computed using Der the first time $\widehat{\mathcal{D}}$ is invoked) or a *fresh* secret key (computed by running Der using fresh random coins each time). The resulting security definitions obtained in each case seem incomparable, and there does not appear to be any reason to prefer one over the other. A related difficulty arises in the case of HIBE (discussed next) even in the case of chosen-plaintext attacks. These distinctions all become irrelevant when Der is deterministic.

Assuming deterministic key derivation is anyway without much loss of generality: given an IBE scheme with randomized key-derivation algorithm Der we can construct an IBE scheme with deterministic key derivation by (1) including a random key sk for a pseudorandom function F as part of the master secret key msk; and (2) generating the decryption key for identity ID by running $\mathsf{Der}_{\mathsf{msk}}(ID)$ using "randomness" $F_{sk}(ID)$. A similar idea applies to the case of HIBE.

3.2. Hierarchical identity-based encryption. Hierarchical identity-based encryption (HIBE) is an extension of IBE suggested by Horwitz and Lynn [41]. In an ℓ-level HIBE scheme, there is again assumed to be a trusted authority who generates master public and secret keys. As in the case of IBE, it is possible to derive a personal secret key SK_{ID_1} for any identity ID_1 using the master secret key. The additional functionality provided by a HIBE scheme is that this personal secret key SK_{ID_1} may now be used to derive a personal secret key SK_{ID_1,ID_2} for the "ID-vector" (ID_1, ID_2), and so on, with the scheme supporting the derivation of keys in this way for ID-vectors of length at most ℓ. As in the case of IBE, any sender can encrypt a message for the ID-vector $v = (ID_1, \ldots, ID_L)$ using only the master public key and v; the resulting ciphertext can be decrypted by anyone who knows $SK_{ID_1,\ldots,ID_L}$. Security is defined as the natural analogue of security in the case of IBE: informally, indistinguishability should hold for ciphertexts encrypted with respect to a target ID-vector $v = (ID_1, \ldots, ID_L)$ as long as the adversary does not know the secret keys of any identity of the form $(ID_1, \ldots, ID_{L'})$ for $L' \leq L$.

Before formally defining an ℓ-level HIBE scheme, we first introduce some notation to deal with ID-vectors $v \in (\{0,1\}^n)^{\leq \ell}$. For an ID-vector $v = (v_1, \ldots, v_L)$ (with $v_i \in \{0,1\}^n$), we define the *length of* v as $|v| = L$ and let $v.r$ (for $r \in \{0,1\}^n$) denote the ID-vector $(v_1, \ldots, v_L, r)$ of length $|v|+1$. We let ε denote the ID-vector of length 0. Given v as above and an ID-vector $v' = (v'_1, \ldots, v'_{L'})$, we say that v *is a prefix of* v' if $|v| \leq |v'|$ and $v_i = v'_i$ for $i \leq |v|$.

Rephrased using the above notation, the functional property of an ℓ-level HIBE scheme is this: given the secret key SK_v associated with the ID-vector v it is possible to derive a secret key $SK_{v'}$ associated with the ID-vector v' (assuming $|v'| \leq \ell$) whenever v is a prefix of v'. Similarly, the security provided by a HIBE scheme is that indistinguishability should hold for ciphertexts encrypted with respect to an ID-vector v even if the adversary has multiple keys $\{SK_{v'}\}_{v' \in V}$ for some set V *as long as no* $v' \in V$ *is a prefix of* v.

Formal definitions follow. The functional definition is essentially from [33], although we assume for simplicity that the key derivation algorithm is deterministic (cf. the remark in the previous section). As in the case of IBE, the definition of security we give is the one proposed by Canetti, Halevi, and Katz [17], which is weaker than the one considered in [33].

DEFINITION 3. *An* ℓ*-level HIBE scheme for identities of length* n *(where* ℓ, n *are*

polynomially bounded functions) is a tuple of PPT *algorithms* $(\mathsf{Setup}, \mathsf{Der}, \mathcal{E}, \mathcal{D})$ *such that the following hold:*

- *The randomized* setup algorithm Setup *takes as input a security parameter* 1^k. *It outputs a master public key* PK *and a master secret key denoted* SK_ε. *(We assume that* $k, \ell = \ell(k)$, *and* $n = n(k)$ *are implicit in* PK *and all node secret keys.)*
- *The deterministic* key-derivation algorithm Der *takes as input an ID-vector* $v \in (\{0,1\}^n)^{<\ell}$, *its associated secret key* SK_v, *and a string* $r \in \{0,1\}^n$. *It returns the secret key* $SK_{v.r}$ *associated with the ID-vector* $v.r$. *We write this as* $SK_{v.r} := \mathsf{Der}_{SK_v}(v, r)$.
- *The randomized* encryption algorithm $\mathcal{E}$ *takes as input the master public key* PK, *an ID-vector* $v \in (\{0,1\}^n)^{\leq \ell}$, *and a message* m *in some implicit message space. It outputs a ciphertext* C. *We write this as* $C \leftarrow \mathcal{E}_{PK}(v, m)$.
- *The (possibly randomized)* decryption algorithm $\mathcal{D}$ *takes as input an ID-vector* $v \in (\{0,1\}^n)^{\leq \ell}$, *its associated secret key* SK_v, *and a ciphertext* C. *It returns a message* m *or the symbol* $\perp$ *(which is not in the message space). We write* $m \leftarrow \mathcal{D}_{SK_v}(v, C)$.

We require that for all (PK, SK_ε) *output by* Setup, *all* $v \in (\{0,1\}^n)^{\leq \ell}$, *any secret key* SK_v *correctly generated (in the obvious way) for* v, *and any message* m *we have* $m = \mathcal{D}_{SK_v}(v, \mathcal{E}_{PK}(v, M))$.

DEFINITION 4. *An* ℓ*-level HIBE scheme* Π *for identities of length* n *is* selective-ID secure against chosen-plaintext attacks *if the advantage of any* PPT *adversary* $\mathcal{A}$ *in the following game is negligible in the security parameter* k*:*

1. *Let* $\ell = \ell(k)$, $n = n(k)$. *Adversary* $\mathcal{A}(1^k)$ *outputs a "target" ID-vector* $v^* \in (\{0,1\}^n)^{\leq \ell}$.
2. *Algorithm* $\mathsf{Setup}(1^k)$ *outputs* (PK, SK_ε). *The adversary is given* PK.
3. *The adversary may adaptively ask for the secret key(s) corresponding to any ID-vector(s)* v, *as long as* v *is not a prefix of the target ID-vector* v^*. *The adversary is given the secret key* SK_v *correctly generated for* v *using* SK_ε *and (repeated applications of)* Der.
4. *At some point, the adversary outputs two messages* m_0, m_1 *with* $|m_0| = |m_1|$. *A bit* b *is randomly chosen, and the adversary is given a "challenge" ciphertext* $C^* \leftarrow \mathcal{E}_{PK}(v^*, m_b)$.
5. *The adversary can continue asking for secret keys as above. Finally,* $\mathcal{A}$ *outputs a guess* b'.

We say that $\mathcal{A}$ succeeds *if* $b' = b$, *and denote the probability of this event by* $\Pr^{\mathsf{HIBE}}_{\mathcal{A},\Pi}[\mathsf{Succ}]$. *The adversary's* advantage *is defined as* $\mathsf{Adv}^{\mathsf{HIBE}}_{\mathcal{A},\Pi}(k) \stackrel{\text{def}}{=} \left| \Pr^{\mathsf{HIBE}}_{\mathcal{A},\Pi}[\mathsf{Succ}] - 1/2 \right|$.

As in the case of IBE, it is easy to modify the above to take into account security against adaptive chosen-ciphertext attacks. Here, the adversary may additionally query an oracle $\widehat{\mathcal{D}}(\cdot,\cdot)$ such that $\widehat{\mathcal{D}}(v, C)$ returns $\mathcal{D}_{SK_v}(v, C)$ using key SK_v correctly generated for v. The only restriction is that the adversary may not query $\widehat{\mathcal{D}}(v^*, C^*)$ after receiving the challenge ciphertext C^*.

4. Chosen-ciphertext security from identity-based encryption. Given an IBE scheme $\Pi' = (\mathsf{Setup}, \mathsf{Der}, \mathcal{E}', \mathcal{D}')$ for identities of length n which is selective-ID secure against chosen-plaintext attacks, we construct a public-key encryption scheme $\Pi = (\mathsf{Gen}, \mathcal{E}, \mathcal{D})$ secure against adaptive chosen-ciphertext attacks. In the construction, we use a strong one-time signature scheme $\mathsf{Sig} = (\mathcal{G}, \mathsf{Sign}, \mathsf{Vrfy})$ (cf. Definition 11 in the appendix) in which the verification key output by $\mathcal{G}(1^k)$ has length $n = n(k)$.

The construction of Π proceeds as follows:

Key generation. $\mathsf{Gen}(1^k)$ runs $\mathsf{Setup}(1^k)$ to obtain (PK, msk). The public key is PK and the secret key is msk.

Encryption. To encrypt message m using public key PK, the sender first runs $\mathcal{G}(1^k)$ to obtain verification key vk and signing key sk (with $|vk| = n$). The sender then computes $C \leftarrow \mathcal{E}'_{PK}(vk, m)$ (i.e., the sender encrypts m with respect to the "identity" vk) and $\sigma \leftarrow \mathsf{Sign}_{sk}(C)$. The final ciphertext is $\langle vk, C, \sigma \rangle$.

Decryption. To decrypt ciphertext $\langle vk, C, \sigma \rangle$ using secret key msk, the receiver first checks whether $\mathsf{Vrfy}_{vk}(C, \sigma) \stackrel{?}{=} 1$. If not, the receiver simply outputs $\perp$. Otherwise, the receiver computes $SK_{vk} \leftarrow \mathsf{Der}_{\mathsf{msk}}(vk)$ and outputs $m \leftarrow \mathcal{D}'_{SK_{vk}}(vk, C)$.

It is clear that the above scheme satisfies correctness. We give some intuition as to why Π is secure against chosen-ciphertext attacks. Let $\langle vk^*, C^*, \sigma^* \rangle$ be the challenge ciphertext (cf. Definition 8). It should be clear that, without any decryption oracle queries, the plaintext corresponding to this ciphertext remains "hidden" to the adversary; this is so because C^* is output by Π' which is CPA-secure (and the additional components of the ciphertext provide no additional help).

We claim that decryption oracle queries cannot further help the adversary in determining the plaintext (i.e., guessing the value of b; cf. Definition 8). On one hand, if the adversary submits to its decryption oracle a ciphertext $\langle vk, C, \sigma \rangle$ that is different from the challenge ciphertext but with $vk = vk^*$ then (with all but negligible probability) the decryption oracle will reply with $\perp$ since the adversary is unable to forge new, valid signatures with respect to vk. On the other hand, if $vk \neq vk^*$ then (informally) the decryption query will not help the adversary since the eventual decryption using $\mathcal{D}'$ (in the underlying scheme Π') will be done with respect to a different "identity" vk. In the proof below, we formalize these ideas.

THEOREM 1. *If Π' is an identity-based encryption scheme which is selective-ID secure against chosen-plaintext attacks and Sig is a strong one-time signature scheme, then Π is a public-key encryption scheme secure against adaptive chosen-ciphertext attacks.*

Proof. Assume we are given a PPT adversary $\mathcal{A}$ attacking Π in an adaptive chosen-ciphertext attack. Say a ciphertext $\langle vk, C, \sigma \rangle$ is *valid* if $\mathsf{Vrfy}_{vk}(C, \sigma) = 1$. Let $\langle vk^*, C^*, \sigma^* \rangle$ denote the challenge ciphertext received by $\mathcal{A}$ during a particular run of the experiment, and let Forge denote the event that $\mathcal{A}$ submits a valid ciphertext $\langle vk^*, C, \sigma \rangle$ to the decryption oracle. (We may assume that vk^* is chosen at the outset of the experiment so this event is well defined even before $\mathcal{A}$ is given the challenge ciphertext. Recall also that $\mathcal{A}$ is disallowed from submitting the challenge ciphertext to the decryption oracle once the challenge ciphertext is given to $\mathcal{A}$.) We prove the following claims.

CLAIM 1. $\Pr^{\mathsf{PKE}}_{\mathcal{A},\Pi}[\mathsf{Forge}]$ *is negligible.*

CLAIM 2. $\left| \Pr^{\mathsf{PKE}}_{\mathcal{A},\Pi}[\mathsf{Succ} \wedge \overline{\mathsf{Forge}}] + \frac{1}{2} \Pr^{\mathsf{PKE}}_{\mathcal{A},\Pi}[\mathsf{Forge}] - \frac{1}{2} \right|$ *is negligible.*

To see that these imply the theorem, note that

$$\begin{aligned}
&\left| \Pr^{\mathsf{PKE}}_{\mathcal{A},\Pi}[\mathsf{Succ}] - \tfrac{1}{2} \right| \\
&\leq \left| \Pr^{\mathsf{PKE}}_{\mathcal{A},\Pi}[\mathsf{Succ} \wedge \mathsf{Forge}] - \tfrac{1}{2} \Pr^{\mathsf{PKE}}_{\mathcal{A},\Pi}[\mathsf{Forge}] \right| \; + \; \left| \Pr^{\mathsf{PKE}}_{\mathcal{A},\Pi}[\mathsf{Succ} \wedge \overline{\mathsf{Forge}}] + \tfrac{1}{2} \Pr^{\mathsf{PKE}}_{\mathcal{A},\Pi}[\mathsf{Forge}] - \tfrac{1}{2} \right| \\
&\leq \tfrac{1}{2} \Pr^{\mathsf{PKE}}_{\mathcal{A},\Pi}[\mathsf{Forge}] + \left| \Pr^{\mathsf{PKE}}_{\mathcal{A},\Pi}[\mathsf{Succ} \wedge \overline{\mathsf{Forge}}] + \tfrac{1}{2} \Pr^{\mathsf{PKE}}_{\mathcal{A},\Pi}[\mathsf{Forge}] - \tfrac{1}{2} \right|,
\end{aligned}$$

which is negligible given the stated claims. (A concrete security bound can be derived easily.)

Proof of Claim 1. The proof is quite straightforward. We construct a PPT forger $\mathcal{F}$ who forges a signature with respect to signature scheme Sig (in the sense of Definition 11) with probability exactly $\mathrm{Pr}^{\mathsf{PKE}}_{\mathcal{A},\Pi}[\mathsf{Forge}]$. Security of Sig implies the claim.

$\mathcal{F}$ is defined as follows: given input 1^k and verification key vk^* (output by $\mathcal{G}$), $\mathcal{F}$ first runs $\mathsf{Setup}(1^k)$ to obtain (PK, msk), and then runs $\mathcal{A}(1^k, PK)$. Note that $\mathcal{F}$ can answer any decryption queries of $\mathcal{A}$. If $\mathcal{A}$ happens to submit a valid ciphertext $\langle vk^*, C, \sigma\rangle$ to its decryption oracle before requesting the challenge ciphertext, then $\mathcal{F}$ simply outputs the forgery (C, σ) and stops. Otherwise, when $\mathcal{A}$ outputs messages m_0, m_1, forger $\mathcal{F}$ proceeds as follows: it chooses a random bit b, computes $C^* \leftarrow \mathcal{E}'_{PK}(vk^*, m_b)$, and obtains (from its signing oracle) a signature σ^* on the "message" C^*. Finally, $\mathcal{F}$ hands the challenge ciphertext $\langle vk^*, C^*, \sigma^*\rangle$ to $\mathcal{A}$. If $\mathcal{A}$ submits a valid ciphertext $\langle vk^*, C, \sigma\rangle$ to its decryption oracle, note that we must have $(C, \sigma) \neq (C^*, \sigma^*)$. In this case, $\mathcal{F}$ simply outputs (C, σ) as its forgery. It is easy to see that $\mathcal{F}$'s success probability (in the sense of Definition 11) is exactly $\mathrm{Pr}^{\mathsf{PKE}}_{\mathcal{A},\Pi}[\mathsf{Forge}]$. □

Proof of Claim 2. We use $\mathcal{A}$ to construct a PPT adversary $\mathcal{A}'$ which attacks the IBE scheme Π' in the sense of Definition 2. Relating the advantages of $\mathcal{A}$ and $\mathcal{A}'$ gives the desired result.

Define adversary $\mathcal{A}'$ as follows:

1. $\mathcal{A}'(1^k)$ runs $\mathcal{G}(1^k)$ to generate (vk^*, sk^*) and outputs the "target" identity $ID^* = vk^*$.
2. $\mathcal{A}'$ is given a master public key PK. Adversary $\mathcal{A}'$, in turn, runs $\mathcal{A}(1^k, PK)$.
3. When $\mathcal{A}$ makes decryption oracle query $\mathcal{D}(\langle vk, C, \sigma\rangle)$, adversary $\mathcal{A}'$ proceeds as follows:
   (a) If $vk = vk^*$ then $\mathcal{A}'$ checks whether $\mathsf{Vrfy}_{vk^*}(C, \sigma) = 1$. If so, $\mathcal{A}'$ aborts and outputs a random bit. Otherwise, it simply responds with $\perp$.
   (b) If $vk \neq vk^*$ and $\mathsf{Vrfy}_{vk}(C, \sigma) = 0$ then $\mathcal{A}'$ responds with $\perp$.
   (c) If $vk \neq vk^*$ and $\mathsf{Vrfy}_{vk}(C, \sigma) = 1$ then $\mathcal{A}'$ makes the oracle query $\mathsf{Der}_{\mathsf{msk}}(vk)$ to obtain SK_{vk}. It then computes $m \leftarrow \mathcal{D}'_{SK_{vk}}(vk, C)$ and responds with m.
4. At some point, $\mathcal{A}$ outputs two equal-length messages m_0, m_1. These messages are output by $\mathcal{A}'$ as well. In return, $\mathcal{A}'$ is given a challenge ciphertext C^*; adversary $\mathcal{A}'$ then computes $\sigma^* \leftarrow \mathsf{Sign}_{vk^*}(C^*)$ and returns $\langle vk^*, C^*, \sigma^*\rangle$ to $\mathcal{A}$.
5. $\mathcal{A}$ may continue to make decryption oracle queries, and these are answered by $\mathcal{A}'$ as before.
6. Finally, $\mathcal{A}$ outputs a guess b'; this same guess is output by $\mathcal{A}'$.

Note that $\mathcal{A}'$ represents a legal adversarial strategy for attacking Π'; in particular, $\mathcal{A}'$ never requests the secret key corresponding to the "target" identity vk^*. Furthermore, $\mathcal{A}'$ provides a perfect simulation for $\mathcal{A}$ until event Forge occurs. It is thus easy to see that

$$\left|\mathrm{Pr}^{\mathsf{IBE}}_{\mathcal{A}',\Pi'}[\mathsf{Succ}] - \tfrac{1}{2}\right| = \left|\mathrm{Pr}^{\mathsf{PKE}}_{\mathcal{A},\Pi}[\mathsf{Succ} \wedge \overline{\mathsf{Forge}}] + \tfrac{1}{2}\,\mathrm{Pr}^{\mathsf{PKE}}_{\mathcal{A},\Pi}[\mathsf{Forge}] - \tfrac{1}{2}\right|,$$

and the left-hand side of the above is negligible by the assumed security of Π'. □

This concludes the proof of the theorem. □

5. A more efficient construction. We show here how the idea from the previous section can be implemented using a MAC and a primitive we call an "encapsulation

scheme" instead of a one-time signature scheme. As argued in the introduction, this results in more efficient constructions of CCA-secure encryption schemes than the previous approach. However, using MACs rather than signatures—which, in particular, will imply that ciphertext validity can no longer be determined efficiently without an appropriate decryption key—complicates the security proof somewhat.

5.1. Encapsulation. We begin by defining a notion of "encapsulation" which may be viewed as a weak variant of commitment. In terms of functionality, an encapsulation scheme commits the sender to a *random string* as opposed to a string chosen by the sender as in the case of commitment. In terms of security, encapsulation only requires binding to hold for *honestly generated encapsulations*; this is analogous to assuming an honest sender during the first phase of a commitment scheme.

DEFINITION 5. *An* encapsulation scheme *is a triple of* PPT *algorithms* $(\mathsf{Init}, \mathcal{S}, \mathcal{R})$ *such that the folllowing hold:*

- Init *takes as input the security parameter* 1^k *and outputs a string* pub.
- $\mathcal{S}$ *takes as input* 1^k *and* pub *and outputs* $(r, \mathsf{com}, \mathsf{dec})$ *with* $r \in \{0,1\}^k$. *We refer to* com *as the* commitment string *and* dec *as the* decommitment string.
- $\mathcal{R}$ *takes as input* $(\mathsf{pub}, \mathsf{com}, \mathsf{dec})$ *and outputs* $r \in \{0,1\}^k \cup \{\perp\}$.

We require that for all pub *output by* Init *and for all* $(r, \mathsf{com}, \mathsf{dec})$ *output by* $\mathcal{S}(1^k, \mathsf{pub})$, *we have* $\mathcal{R}(\mathsf{pub}, \mathsf{com}, \mathsf{dec}) = r$. *We also assume for simplicity that* com *and* dec *have fixed lengths for any given value of the security parameter.*

As in the case of commitment, an encapsulation scheme satisfies notions of binding and hiding. Informally, "hiding" requires that com should not reveal information about r; formally, r should be indistinguishable from random even when given com (and pub). "Binding" requires that an honestly generated com can be "opened" to only a single (legal) value of r; see below.

DEFINITION 6. *An encapsulation scheme is* secure *if it satisfies both hiding and binding as follows:*

Hiding: *The following is negligible for all* PPT $\mathcal{A}$*:*

$$\left| \Pr\left[\begin{array}{c} \mathsf{pub} \leftarrow \mathsf{Init}(1^k); r_0 \leftarrow \{0,1\}^k; \\ (r_1, \mathsf{com}, \mathsf{dec}) \leftarrow \mathcal{S}(1^k, \mathsf{pub}); b \leftarrow \{0,1\} : \mathcal{A}(1^k, \mathsf{pub}, \mathsf{com}, r_b) = b \end{array} \right] - \frac{1}{2} \right|.$$

Binding: *The following is negligible for all* PPT $\mathcal{A}$*:*

$$\Pr\left[\begin{array}{c} \mathsf{pub} \leftarrow \mathsf{Init}(1^k); \\ (r, \mathsf{com}, \mathsf{dec}) \leftarrow \mathcal{S}(1^k, \mathsf{pub}); \quad : \quad \mathcal{R}(\mathsf{pub}, \mathsf{com}, \mathsf{dec}') \notin \{\perp, r\} \\ \mathsf{dec}' \leftarrow \mathcal{A}(1^k, \mathsf{pub}, \mathsf{com}, \mathsf{dec}) \end{array} \right].$$

Both hiding and binding are required to hold only computationally. The encapsulation scheme we will later construct achieves *statistical* hiding (and computational binding).

Since encapsulation is a weaker primitive than commitment, we could use any commitment scheme as an encapsulation scheme. We will be interested, however, in optimizing the efficiency of the construction (in particular, the lengths of com and dec for a fixed value of k) and therefore focus on satisfying only the weaker requirements given above. See further discussion in section 7.2.

5.2. The construction. Let $\Pi' = (\mathsf{Setup}, \mathsf{Der}, \mathcal{E}', \mathcal{D}')$ be an IBE scheme for identities of length $n = n(k)$ which is selective-ID secure against chosen-plaintext attacks, let $(\mathsf{Init}, \mathcal{S}, \mathcal{R})$ be a secure encapsulation scheme in which commitments com output by $\mathcal{S}$ have length n, and let $(\mathsf{Mac}, \mathsf{Vrfy})$ be a MAC. We construct a public-key encryption scheme Π as follows:

Key generation. Keys for our scheme are generated by running $\mathsf{Setup}(1^k)$ to generate (PK, msk) and $\mathsf{Init}(1^k)$ to generate pub. The public key is (PK, pub), and the secret key is msk.

Encryption. To encrypt a message m using public key (PK, pub), a sender first encapsulates a random value by running $\mathcal{S}(1^k, \mathsf{pub})$ to obtain $(r, \mathsf{com}, \mathsf{dec})$. The sender then encrypts the "message" $m \circ \mathsf{dec}$ with respect to the "identity" com; that is, the sender computes $C \leftarrow \mathcal{E}'_{PK}(\mathsf{com}, m \circ \mathsf{dec})$. The resulting ciphertext C is then authenticated by using r as a key for a message authentication code; i.e., the sender computes $\mathsf{tag} \leftarrow \mathsf{Mac}_r(C)$. The final ciphertext is $\langle \mathsf{com}, C, \mathsf{tag} \rangle$.

Decryption. To decrypt a ciphertext $\langle \mathsf{com}, C, \mathsf{tag} \rangle$, the receiver derives the secret key SK_{com} corresponding to the "identity" com, and uses this key to decrypt the ciphertext C as per the underlying IBE scheme; this yields a "message" $m \circ \mathsf{dec}$ (if decryption fails, the receiver outputs $\perp$). Next, the receiver runs $\mathcal{R}(\mathsf{pub}, \mathsf{com}, \mathsf{dec})$ to obtain a string r; if $r \neq \perp$ and $\mathsf{Vrfy}_r(C, \mathsf{tag}) = 1$, the receiver outputs m. Otherwise, the receiver outputs $\perp$.

THEOREM 2. *If Π' is an identity-based encryption scheme which is selective-ID secure against chosen-plaintext attacks, the encapsulation scheme is secure (in the sense of Definition 6), and $(\mathsf{Mac}, \mathsf{Vrfy})$ is a strong one-time message authentication code, then Π a public-key encryption scheme secure against adaptive chosen-ciphertext attacks.*

Proof. Let $\mathcal{A}$ be a PPT adversary attacking Π in an adaptive chosen-ciphertext attack. On an intuitive level, the proof here is the same as the proof of Theorem 1 in the following sense: Say a ciphertext $\langle \mathsf{com}, C, \mathsf{tag} \rangle$ is *valid* if decryption of this ciphertext (using msk) does *not* result in $\perp$. Let $\langle \mathsf{com}^*, C^*, \mathsf{tag}^* \rangle$ denote the challenge ciphertext received by $\mathcal{A}$. We will show that (1) $\mathcal{A}$ submits to its decryption oracle a valid ciphertext $\langle \mathsf{com}^*, C, \mathsf{tag} \rangle$ (with $\langle C, \mathsf{tag} \rangle \neq \langle C^*, \mathsf{tag}^* \rangle$) only with negligible probability; and (2) assuming that the previous event does not occur, the decryption queries made by $\mathcal{A}$ do not help $\mathcal{A}$ to "learn" the underlying plaintext. The second statement is relatively easy to prove based on the security of Π'; the first, however, is now more challenging to prove since validity of a ciphertext cannot be determined without knowledge of msk. Because of this, we structure the proof as a sequence of games to make it easier to follow. We let $\Pr_i[\cdot]$ denote the probability of a particular event occurring in game i.

Game 0 is the original game in which $\mathcal{A}$ attacks Π in a chosen-ciphertext attack as described in Definition 8. Let $r^*, \mathsf{com}^*, \mathsf{dec}^*$ denote the values that are used in computing the challenge ciphertext, and notice that we may assume these values are generated at the outset of the experiment (since these values are generated independently of $\mathcal{A}$'s actions). We are interested in upper-bounding $\left|\Pr_0[\mathsf{Succ}] - \frac{1}{2}\right|$, where (recall) Succ denotes the event that $\mathcal{A}$'s output bit b' is identical to the bit b used in constructing the challenge ciphertext.

In Game 1, we modify the experiment as follows: on input of a ciphertext of the form $\langle \mathsf{com}^*, C, \mathsf{tag} \rangle$, the decryption oracle simply outputs $\perp$. Let Valid denote the event that $\mathcal{A}$ submits a ciphertext $\langle \mathsf{com}^*, C, \mathsf{tag} \rangle$ to its decryption oracle which is valid, and note that

$$\left|\Pr_1[\mathsf{Succ}] - \Pr_0[\mathsf{Succ}]\right| \leq \Pr_0[\mathsf{Valid}] = \Pr_1[\mathsf{Valid}].$$

The above holds since Games 0 and 1 are identical until Valid occurs.

Let NoBind denote the event that $\mathcal{A}$ at some point submits a ciphertext $\langle \mathsf{com}^*, C, \mathsf{tag}\rangle$ to its decryption oracle such that: (1) C decrypts to $m \circ \mathsf{dec}$ (using the secret key SK_{com^*} derived from msk) and (2) $\mathcal{R}(\mathsf{pub}, \mathsf{com}^*, \mathsf{dec}) = r$ with $r \notin \{r^*, \perp\}$. Let Forge denote the event that $\mathcal{A}$ at some point submits a ciphertext $\langle \mathsf{com}^*, C, \mathsf{tag}\rangle$ to its decryption oracle such that $\mathsf{Vrfy}_{r^*}(C, \mathsf{tag}) = 1$. We clearly have $\Pr_1[\mathsf{Valid}] \leq \Pr_1[\mathsf{NoBind}] + \Pr_1[\mathsf{Forge}]$.

It is relatively easy to see that $\Pr_1[\mathsf{NoBind}]$ is negligible assuming the binding property of the encapsulation scheme. Formally, consider an adversary $\mathcal{B}$ acting as follows: given input $(1^k, \mathsf{pub}, \mathsf{com}^*, \mathsf{dec}^*)$, adversary $\mathcal{B}$ generates (PK, msk) by running $\mathsf{Setup}(1^k)$ and then runs $\mathcal{A}$ on inputs 1^k and (PK, pub). Whenever $\mathcal{A}$ makes a query to its decryption oracle, $\mathcal{B}$ can respond to this query as required by Game 1; specifically, $\mathcal{B}$ simply responds with $\perp$ to a decryption query of the form $\langle \mathsf{com}^*, C, \mathsf{tag}\rangle$, and responds to other queries using msk. When $\mathcal{A}$ submits its two messages m_0, m_1, adversary $\mathcal{B}$ simply chooses $b \in \{0, 1\}$ at random and encrypts m_b in the expected way to generate a completely valid challenge ciphertext $\langle \mathsf{com}^*, C^*, \mathsf{tag}^*\rangle$. (Note that $\mathcal{B}$ can easily do this since it has dec^* and can compute r^*.) At the end of the experiment, $\mathcal{B}$ can decrypt every query of the form $\langle \mathsf{com}^*, C, \mathsf{tag}\rangle$ that $\mathcal{A}$ made to its decryption oracle to see whether NoBind occurred and, if so, to learn a value dec such that $\mathcal{R}(\mathsf{pub}, \mathsf{com}^*, \mathsf{dec}) \notin \{r^*, \perp\}$. But this exactly violates the binding property of encapsulation scheme $(\mathsf{Init}, \mathcal{S}, \mathcal{R})$, implying that $\Pr_1[\mathsf{NoBind}]$ must be negligible.

Game 2 is derived by modifying the way the challenge ciphertext is computed. Specifically, when $\mathcal{A}$ submits its two messages m_0, m_1 we now compute $C^* \leftarrow \mathcal{E}'_{PK}(\mathsf{com}^*, 0^{|m_0|} \circ 0^n)$ followed by $\mathsf{tag}^* \leftarrow \mathsf{Mac}_{r^*}(C^*)$. The challenge ciphertext is $\langle \mathsf{com}^*, C^*, \mathsf{tag}^*\rangle$. (A random bit b is still chosen, but is only used to define event Succ.) Since the challenge ciphertext is independent of b, it follows immediately that $\Pr_2[\mathsf{Succ}] = \frac{1}{2}$.

We claim that $|\Pr_2[\mathsf{Succ}] - \Pr_1[\mathsf{Succ}]|$ is negligible. To see this, consider the following adversary $\mathcal{A}'$ attacking the IBE scheme Π' via a chosen-plaintext attack:

- Algorithm $\mathcal{A}'(1^k)$ first runs $\mathsf{Init}(1^k)$ to generate pub and then runs $\mathcal{S}(1^k, \mathsf{pub})$ to obtain $(r^*, \mathsf{com}^*, \mathsf{dec}^*)$. It outputs com^* as the target identity and is then given the master public key PK. Finally, $\mathcal{A}'$ runs $\mathcal{A}$ on inputs 1^k and (PK, pub).
- Decryption queries of $\mathcal{A}$ are answered in the natural way:
  - Queries of the form $\langle \mathsf{com}^*, C, \mathsf{tag}\rangle$ are answered with $\perp$.
  - Queries of the form $\langle \mathsf{com}, C, \mathsf{tag}\rangle$ with $\mathsf{com} \neq \mathsf{com}^*$ are answered by first querying $\mathsf{Der}_{\mathsf{msk}}(\mathsf{com})$ to obtain SK_{com}, and then decrypting in the usual way.
- Eventually, $\mathcal{A}$ submits two equal-length messages m_0, m_1. $\mathcal{A}'$ selects a bit b at random, and sends $m_b \circ \mathsf{dec}^*$ and $0^{|m_0|} \circ 0^n$ to its encryption oracle. It receives in return a challenge ciphertext C^*, and uses this to generate a ciphertext $\langle \mathsf{com}^*, C^*, \mathsf{tag}^*\rangle$ in the natural way.
- Further decryption queries of $\mathcal{A}$ are answered as above.
- Finally, $\mathcal{A}$ outputs a bit b'. If $b = b'$, then $\mathcal{A}'$ outputs 0; otherwise, $\mathcal{A}'$ outputs 1.

Note that $\mathcal{A}'$ is a valid adversary. When the encryption query of $\mathcal{A}'$ is answered with an encryption of $m_b \circ \mathsf{dec}^*$, then the view of $\mathcal{A}$ is exactly as in Game 1; on the other hand, when the encryption query of $\mathcal{A}'$ is answered with an encryption of $0^{|m_0|} \circ 0^n$,

then the view of $\mathcal{A}$ is exactly as in Game 2. Thus,

$$\begin{aligned}\mathsf{Adv}^{\mathsf{IBE}}_{\mathcal{A}',\Pi'}(k) &= \left|\tfrac{1}{2}\Pr_1[\mathsf{Succ}] + \tfrac{1}{2}\Pr_2[\overline{\mathsf{Succ}}] - \tfrac{1}{2}\right| \\ &= \tfrac{1}{2}\cdot\left|\Pr_1[\mathsf{Succ}] - \Pr_2[\mathsf{Succ}]\right|.\end{aligned}$$

Security of Π' implies that $\mathsf{Adv}^{\mathsf{IBE}}_{\mathcal{A}',\Pi'}$ is negligible, implying that $|\Pr_2[\mathsf{Succ}]-\Pr_1[\mathsf{Succ}]|$ is negligible. An exactly analogous argument shows that $|\Pr_2[\mathsf{Forge}] - \Pr_1[\mathsf{Forge}]|$ is negligible as well. (The only difference is that $\mathcal{A}'$ runs $\mathcal{A}$ to completion and then checks whether $\mathcal{A}$ has made any decryption query of the form $\langle \mathsf{com}^*, C, \mathsf{tag}\rangle$ for which $\mathsf{Vrfy}_{r^*}(C,\mathsf{tag}) = 1$. If so, then $\mathcal{A}'$ outputs 1; otherwise, it outputs 0.)

In Game 3, we introduce one final change. The components com^* and C^* of the challenge ciphertext are computed as in Game 2; however, the component tag^* is computed by choosing a random key $r \in \{0,1\}^k$ and setting $\mathsf{tag}^* = \mathsf{Mac}_r(C^*)$. Event Forge in this game is defined as before, but using the key r; that is, Forge is now the event that $\mathcal{A}$ makes a decryption query of the form $\langle \mathsf{com}^*, C, \mathsf{tag}\rangle$ for which $\mathsf{Vrfy}_r(C,\mathsf{tag}) = 1$.

We claim that $|\Pr_3[\mathsf{Forge}] - \Pr_2[\mathsf{Forge}]|$ is negligible. To see this, consider the following algorithm $\mathcal{B}$ attacking the hiding property of the encapsulation scheme:

- $\mathcal{B}$ is given input 1^k and $(\mathsf{pub}, \mathsf{com}^*, \tilde{r})$. It then runs $\mathsf{Setup}(1^k)$ to generate (PK, msk), and runs $\mathcal{A}$ on inputs 1^k and (PK, pub).
- Decryption queries of $\mathcal{A}$ are answered in the natural way.
- Eventually, $\mathcal{A}$ submits messages m_0, m_1. $\mathcal{B}$ computes $C^* \leftarrow \mathcal{E}'_{PK}(\mathsf{com}^*, 0^{|m_0|} \circ 0^n)$, computes $\mathsf{tag}^* = \mathsf{Mac}_{\tilde{r}}(C^*)$, and returns the challenge ciphertext $\langle \mathsf{com}^*, C^*, \mathsf{tag}^*\rangle$ to $\mathcal{A}$.
- Further decryption queries of $\mathcal{A}$ are answered as above.
- When $\mathcal{A}$ halts, $\mathcal{B}$ checks whether $\mathcal{A}$ has made any decryption query of the form $\langle \mathsf{com}^*, C, \mathsf{tag}\rangle$ for which $\mathsf{Vrfy}_{\tilde{r}}(C,\mathsf{tag}) = 1$. If so, $\mathcal{B}$ outputs 1; otherwise, it outputs 0.

Now, if $\tilde{r}$ is such that $(\tilde{r}, \mathsf{com}^*, \mathsf{dec}^*)$ was output by $\mathcal{S}(1^k, \mathsf{pub})$, then the view of $\mathcal{A}$ is exactly as in Game 2 and so $\mathcal{B}$ outputs 1 with probability $\Pr_2[\mathsf{Forge}]$. On the other hand, if $\tilde{r}$ is chosen at random independently of com^*, then the view of $\mathcal{A}$ is exactly as in Game 3 and so $\mathcal{B}$ outputs 1 with probability $\Pr_3[\mathsf{Forge}]$. The hiding property of the encapsulation scheme thus implies that $|\Pr_3[\mathsf{Forge}] - \Pr_2[\mathsf{Forge}]|$ is negligible.

To complete the proof, we show that $\Pr_3[\mathsf{Forge}]$ is negligible. This follows rather easily from the security of the MAC, but we sketch the details here. Let $q = q(k)$ be an upper bound on the number of decryption oracle queries made by $\mathcal{A}$, and consider the following forging algorithm $\mathcal{F}$: first, $\mathcal{F}$ chooses a random index $j \leftarrow \{1,\ldots,q\}$. Next, $\mathcal{F}$ begins simulating Game 3 for $\mathcal{A}$ in the natural way. If the jth decryption query $\langle \mathsf{com}_j, C_j, \mathsf{tag}_j\rangle$ occurs before $\mathcal{A}$ makes its encryption query, then $\mathcal{F}$ simply outputs (C_j, tag_j) and halts. Otherwise, in response to the encryption query (m_0, m_1) of $\mathcal{A}$, forger $\mathcal{F}$ computes $(r^*, \mathsf{com}^*, \mathsf{dec}^*) \leftarrow \mathcal{S}(1^k, \mathsf{pub})$ followed by $C^* \leftarrow \mathcal{E}'_{PK}(\mathsf{com}^*, 0^{|m_0|} \circ 0^n)$. Next, $\mathcal{F}$ submits C^* to its Mac oracle and receives in return tag^*. Forger $\mathcal{F}$ then gives the challenge ciphertext $\langle \mathsf{com}^*, C^*, \mathsf{tag}^*\rangle$ to $\mathcal{A}$ and continues running $\mathcal{A}$ until $\mathcal{A}$ submits its jth decryption query $\langle \mathsf{com}_j, C_j, \mathsf{tag}_j\rangle$. At this point, $\mathcal{F}$ outputs (C_j, tag_j) and halts.

It is not difficult to see that the success probability of $\mathcal{F}$ in outputting a valid forgery is at least $\Pr_3[\mathsf{Forge}]/q$. Since $(\mathsf{Mac}, \mathsf{Vrfy})$ is a strong one-time MAC and q is polynomial, this shows that $\Pr_3[\mathsf{Forge}]$ is negligible.

Putting everything together, we have

$$\begin{aligned}\left|\Pr_0[\mathsf{Succ}] - \tfrac{1}{2}\right| &\le \left|\Pr_0[\mathsf{Succ}] - \Pr_1[\mathsf{Succ}]\right| + \left|\Pr_1[\mathsf{Succ}] - \tfrac{1}{2}\right| \\ &\le \Pr_1[\mathsf{NoBind}] + \Pr_1[\mathsf{Forge}] + \left|\Pr_1[\mathsf{Succ}] - \Pr_2[\mathsf{Succ}]\right| + \left|\Pr_2[\mathsf{Succ}] - \tfrac{1}{2}\right| \\ &= \Pr_1[\mathsf{NoBind}] + \Pr_1[\mathsf{Forge}] + \left|\Pr_1[\mathsf{Succ}] - \Pr_2[\mathsf{Succ}]\right| \\ &\le \Pr_1[\mathsf{NoBind}] + \Pr_3[\mathsf{Forge}] + \left|\Pr_2[\mathsf{Forge}] - \Pr_3[\mathsf{Forge}]\right| \\ &\quad + \left|\Pr_1[\mathsf{Forge}] - \Pr_2[\mathsf{Forge}]\right| + \left|\Pr_1[\mathsf{Succ}] - \Pr_2[\mathsf{Succ}]\right|,\end{aligned}$$

and all terms in the final equation are negligible. (A concrete security analysis follows easily from the above.) □

6. Chosen-ciphertext security for IBE and HIBE schemes. The techniques of the previous two sections extend relatively easily to enable construction of an ℓ-level HIBE scheme secure against chosen-ciphertext attacks based on any $(\ell+1)$-level HIBE scheme secure against chosen-plaintext attacks. (Note that an IBE scheme is simply a 1-level HIBE scheme.) We give the details for the signature-based approach of section 4. For arbitrary $\ell \ge 1$, let $\Pi' = (\mathsf{Setup}', \mathsf{Der}', \mathcal{E}', \mathcal{D}')$ be an $(\ell+1)$-level HIBE scheme handling identities of length $n+1$, and let $\mathsf{Sig} = (\mathcal{G}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme in which the verification key output by $\mathcal{G}(1^k)$ has length $n = n(k)$. We construct an ℓ-level HIBE scheme Π handling identities of length n. The intuition behind the construction is simple: the ID-vector $v = (v_1, \ldots, v_L) \in (\{0,1\}^n)^L$ in Π will be mapped to the ID-vector

$$\mathsf{Encode}(v) \stackrel{\text{def}}{=} (0v_1, \ldots, 0v_L) \in (\{0,1\}^{n+1})^L$$

in Π'. We will maintain the invariant that the secret key SK_v for ID-vector v in Π will be the secret key $SK'_{\hat{v}}$ for ID-vector $\hat{v} = \mathsf{Encode}(v)$ in Π'. When encrypting a message m to ID-vector v in Π, the sender will generate a verification key vk and then encrypt m to the ID-vector $\hat{v}.(1vk)$ using Π'. (The resulting ciphertext will then be signed as in section 4.) The extra 0 and 1 bits used as "padding" ensure that any decryption queries asked by an adversary (in Π) correspond (in Π') to nodes that are not ancestors of the target ID-vector.

In more detail, Π is constructed as follows:

Setup. The Setup algorithm is the same as in Π'. (Note that $\mathsf{Encode}(\varepsilon) = \varepsilon$ so the master secret key $SK_\varepsilon = SK'_\varepsilon$ satisfies the desired invariant.)

Key derivation. $\mathsf{Der}_{SK_v}(v, r)$ runs as follows: let $\hat{v} = \mathsf{Encode}(v)$ and $\hat{r} = \mathsf{Encode}(r)$. Run $\mathsf{Der}'_{SK_v}(\hat{v}, \hat{r})$ and output the result as $SK_{v.r}$. (To see that key derivation maintains the desired invariant given that $SK_v = SK'_{\hat{v}}$, note that $\mathsf{Encode}(v.r) = \mathsf{Encode}(v).\mathsf{Encode}(r)$.)

Encryption. $\mathcal{E}_{PK}(v, m)$ first runs $\mathcal{G}(1^k)$ to obtain (vk, sk). Let $\hat{v} = \mathsf{Encode}(v).(1vk)$. The algorithm then computes $C \leftarrow \mathcal{E}'_{PK}(\hat{v}, m)$ and $\sigma \leftarrow \mathsf{Sign}_{sk}(C)$. The final ciphertext is $\langle vk, C, \sigma \rangle$.

Decryption. $\mathcal{D}_{SK_v}(v, \langle vk, C, \sigma \rangle)$ proceeds as follows: first check whether $\mathsf{Vrfy}_{vk}(C, \sigma) \stackrel{?}{=} 1$. If not, output $\perp$. Otherwise, let $\hat{v} = \mathsf{Encode}(v)$ and run $\mathsf{Der}'_{SK_v}(\hat{v}, (1vk))$ to generate the key $SK^* = SK'_{\hat{v}.(1vk)}$. Then output $m := \mathcal{D}'_{SK^*}(\hat{v}, C)$.

It can be verified easily that the above scheme is correct. An analogous construction can be given using the MAC-based construction of section 5. We now state the main result of this section.

THEOREM 3. *If* Π' *is selective-ID secure against chosen-plaintext attacks and* Sig *is a strong one-time signature scheme, then* Π *is selective-ID secure against chosen-ciphertext attacks.*

Proof. The proof is similar to that of Theorem 1. Given any PPT adversary $\mathcal{A}$ attacking Π in a selective-ID chosen-ciphertext attack, we define an event Forge and then prove the analogues of Claims 1 and 2 in our setting. For visual comfort, we use $\Pr[\cdot]$ instead of $\Pr^{\mathsf{HIBE}}_{\mathcal{A},\Pi}[\cdot]$.

Let v^* denote the "target" ID-vector initially output by $\mathcal{A}$, and let $\langle vk^*, C^*, \sigma^*\rangle$ be the challenge ciphertext received by $\mathcal{A}$. Let Forge be the event that $\mathcal{A}$ makes a decryption query $\widehat{\mathcal{D}}(v^*, \langle vk^*, C, \sigma\rangle)$ with $\mathsf{Vrfy}_{vk^*}(C, \sigma) = 1$. (As in the previous proof, we may assume vk^* is chosen at the beginning of the experiment and so this event is defined even before $\mathcal{A}$ receives the challenge ciphertext. Recall again that $\mathcal{A}$ is disallowed from submitting the challenge ciphertext to its decryption oracle once this ciphertext has been given to $\mathcal{A}$.) A proof exactly as in the case of Claim 1, relying again on the fact that Sig is a strong one-time signature scheme, shows that $\Pr[\mathsf{Forge}]$ is negligible.

We next show that $|\Pr[\mathsf{Succ} \wedge \overline{\mathsf{Forge}}] + \frac{1}{2}\Pr[\mathsf{Forge}] - \frac{1}{2}|$ is negligible. To do so, we define adversary $\mathcal{A}'$ attacking Π' in a selective-ID chosen-plaintext attack. $\mathcal{A}'$ is defined as follows:

1. $\mathcal{A}'(1^k)$ runs $\mathcal{A}(1^k)$ who, in turn, outputs an ID-vector $v^* \in (\{0,1\}^n)^{\leq \ell}$. Adversary $\mathcal{A}'$ runs $\mathcal{G}(1^k)$ to generate (vk^*, sk^*) and outputs the target ID-vector $V^* = \mathsf{Encode}(v^*).(1vk^*)$.
2. $\mathcal{A}'$ is given PK, which it gives to $\mathcal{A}$.
3. When $\mathcal{A}$ requests the secret key for ID-vector v, $\mathcal{A}'$ requests the secret key $SK'_{\hat{v}}$ for ID-vector $\hat{v} = \mathsf{Encode}(v)$ and returns this secret key to $\mathcal{A}$. Note that since v is not a prefix of the target ID-vector v^* of $\mathcal{A}$, it follows that $\hat{v}$ is not a prefix of the target ID-vector V^* of $\mathcal{A}'$.
4. When $\mathcal{A}$ makes a decryption query $\widehat{\mathcal{D}}(v, \langle vk, C, \sigma\rangle)$, adversary $\mathcal{A}'$ proceeds as follows:
   (a) If $v = v^*$ then $\mathcal{A}'$ checks whether $\mathsf{Vrfy}_{vk}(C, \sigma) = 1$. If so, then $\mathcal{A}'$ aborts and outputs a random bit. Otherwise, it simply responds with $\perp$.
   (b) If $v \neq v^*$, or if $v = v^*$ and $vk \neq vk^*$, then $\mathcal{A}'$ sets $\hat{v} = \mathsf{Encode}(v)$ and requests the secret key $SK'_{\hat{v}.(1vk)}$. (Note that $\hat{v}.(1vk)$ is not a prefix of the target ID-vector V^* of $\mathcal{A}'$, so $\mathcal{A}'$ is allowed to submit this request.) It then honestly decrypts the submitted ciphertext and returns the result to $\mathcal{A}$.
5. When $\mathcal{A}$ outputs its two messages m_0, m_1, these same messages are output by $\mathcal{A}'$. In return, $\mathcal{A}'$ receives a challenge ciphertext C^*. Adversary $\mathcal{A}'$ computes $\sigma^* \leftarrow \mathsf{Sign}_{sk^*}(C^*)$ and returns challenge ciphertext $\langle vk^*, C^*, \sigma^*\rangle$ to $\mathcal{A}$.
6. Any of $\mathcal{A}$'s subsequent decryption queries, or requests for secret keys, are answered as before.
7. Finally, $\mathcal{A}$ outputs a guess b'; this same guess is output by $\mathcal{A}'$.

Note that $\mathcal{A}'$ represents a legal adversarial strategy for attacking Π'. As in the proof of Claim 2, it follows from the security of Π' that $|\Pr[\mathsf{Succ} \wedge \overline{\mathsf{Forge}}] + \frac{1}{2}\Pr[\mathsf{Forge}] - \frac{1}{2}|$ is negligible. This completes the proof. □

We remark that when Π' is fully secure against chosen-plaintext attacks [11, 33], then Π is fully secure against chosen-ciphertext attacks. A proof for this case is easily derived from the proof above.

Canetti, Halevi, and Katz [17] define a slightly stronger notion of HIBE which

requires the HIBE scheme to support an arbitrary (polynomial) number of levels ℓ and identities of arbitrary (polynomial) length n (where ℓ, n are provided as input to the initial Setup algorithm). We refer to HIBE schemes of this type as *unbounded*. Security is defined as in Definition 4, except that the adversary's advantage must be negligible for all adversaries $\mathcal{A}$ as well as for all polynomially bounded functions ℓ, n. Since the above construction requires only a strong one-time signature scheme, which can be constructed based on any one-way function (and hence from any secure HIBE scheme), we have the following.

COROLLARY 1. *If there exists an unbounded HIBE scheme which is selective-ID secure (resp., fully secure) against chosen-plaintext attacks, then there exists an unbounded HIBE scheme which is selective-ID secure (resp., fully secure) against adaptive chosen-ciphertext attacks.*

The analogous result for the case of (standard) public-key encryption is not known.

7. An efficient instantiation. Here, we describe one instantiation of our generic construction of CCA-secure cryptosystems from section 5. We then compare the efficiency of this construction with the most efficient previously known CCA-secure scheme. To instantiate our construction, we need to specify a message authentication code, an encapsulation scheme, and an IBE scheme which is selective-ID secure against chosen-plaintext attacks. We consider each of these in turn.

7.1. Message authentication code. A number of efficient (strong, one-time) MACs are known. Since the computational cost of these schemes will be dominated by the computational cost of the IBE scheme, we focus instead on minimizing the lengths of the key and the tag. For concreteness, we suggest using CBC-MAC with 128-bit AES as the underlying block cipher. (In this scheme, both the secret key and the tag are 128 bits long.) We remark, however, that strong one-time MACs with information-theoretic security [57, 54] could also be used.

7.2. Encapsulation scheme. Adapting earlier work of Damgård, Pedersen, and Pfitzmann [24] and Halevi and Micali [39], we propose an encapsulation scheme based on any universal one-way hash function (UOWHF) family $\{H_s : \{0,1\}^{k_1} \to \{0,1\}^k\}$ (where $k_1 \geq 3k$ is a function of the security parameter k). Our scheme works as follows:

- Init chooses a hash function h from a family of pairwise-independent hash functions mapping k_1-bit strings to k-bit strings, and also chooses at random a key s defining UOWHF H_s. It outputs $\mathsf{pub} = (h, s)$.
- The encapsulation algorithm $\mathcal{S}$ takes pub as input, chooses a random $x \in \{0,1\}^{k_1}$, and then outputs $(r = h(x), \mathsf{com} = H_s(x), \mathsf{dec} = x)$.
- The recovery algorithm $\mathcal{R}$ takes as input $((h, s), \mathsf{com}, \mathsf{dec})$ and outputs $h(\mathsf{dec})$ if $H_s(\mathsf{dec}) = \mathsf{com}$, and $\perp$ otherwise.

We prove the following regarding the above scheme.

THEOREM 4. *The scheme above is a secure encapsulation scheme. Specifically, the scheme is computationally binding under the assumption that $\{H_s\}$ is a UOWHF family, and statistically hiding (without any assumptions).*

Proof. The binding property is easy to see. In particular, violation of the binding property implies that an adversary finds $\mathsf{dec}' \neq \mathsf{dec}$ for which $H_s(\mathsf{dec}') = H_s(\mathsf{dec})$. Since dec is chosen independently of the key s in an honest execution of $\mathcal{S}$, security of the UOWHF family implies that binding can be violated with only negligible probability. We omit the straightforward details. (Note, however, that a UOWHF rather

than a collision-resistant hash function is sufficient here since the binding property we require is weaker than that required by a standard commitment scheme.)

We next prove the following claim, which immediately implies statistical hiding.

CLAIM 3. *For the encapsulation scheme described above, the statistical difference between the following distributions is at most* $2 \cdot 2^{\frac{2k-k_1}{3}} \leq 2 \cdot 2^{-k/3}$*:*

$$(1)\{\mathsf{pub} \leftarrow \mathsf{Setup}; (r, \mathsf{com}, \mathsf{dec}) \leftarrow \mathcal{S}(\mathsf{pub}) : (\mathsf{pub}, \mathsf{com}, r)\},$$
$$(2)\{\mathsf{pub} \leftarrow \mathsf{Setup}; (r, \mathsf{com}, \mathsf{dec}) \leftarrow \mathcal{S}(\mathsf{pub}); r' \leftarrow \{0,1\}^k : (\mathsf{pub}, \mathsf{com}, r')\}.$$

The proof of this claim is loosely based on [24, 39], but our proof is much simpler. Let $\alpha \stackrel{\text{def}}{=} \frac{2k_1-k}{3}$, and assume for simplicity that k_1, k are multiples of 3. Fix an arbitrary s for the remainder of the discussion. For any fixed $x \in \{0,1\}^{k_1}$, let $N_x \stackrel{\text{def}}{=} \{x' \mid H_s(x') = H_s(x)\}$; this is simply the set of elements hashing to $H_s(x)$. Call x *good* if $|N_x| \geq 2^\alpha$, and *bad* otherwise. Since the output length of H_s is k bits, there are at most $2^\alpha \cdot 2^k = 2^{\alpha+k}$ bad x's; thus, the probability that an x chosen uniformly at random from $\{0,1\}^{k_1}$ is bad is at most

$$2^{\alpha+k-k_1} \;=\; 2^{\frac{2k-k_1}{3}} \;\leq\; 2^{-k/3}$$

(using the fact that $k_1 \geq 3k$).

When x is good, the min-entropy of x—given (h, s) and $H_s(x)$—is at least α since every $\tilde{x} \in N_x$ is equally likely. Let U_k represent the uniform distribution over $\{0,1\}^k$. Viewing h as a strong extractor (or, equivalently, applying the leftover-hash lemma [40]) we see that the statistical difference between $\{h, s, H_s(x), h(x)\}$ and $\{h, s, H_s(x), U_k\}$ is at most

$$2^{-(\alpha-k)/2} \;=\; 2^{\frac{2k-k_1}{3}} \;\leq\; 2^{-k/3}.$$

The claim, and hence the theorem, follows. □

A practical setting of the above parameters (and one that we will use when discussing the efficiency of our scheme, below) is $k_1 = 448$, $k = 128$ which yields a 128-bit r with statistical difference at most $2 \cdot 2^{\frac{256-448}{3}} = 2^{-63}$ from uniform.[5] Also, in practice one would likely replace the UOWHF by a suitable modification of a cryptographic hash function such as SHA-1.

7.3. IBE scheme. Boneh and Boyen [7] recently proposed two efficient IBE schemes satisfying the definition of security needed for our purposes. In the interest of space, we explore an instantiation of our construction using their first scheme only. (Of course, their second scheme could also be used. Doing so yields a mild efficiency improvement at the expense of requiring a stronger cryptographic assumption.)

We briefly discuss the cryptographic assumption on which the Boneh–Boyen IBE scheme is based. Let $\mathcal{IG}$ denote an efficient algorithm which, on input 1^k, outputs descriptions of two cyclic groups $\mathbb{G}, \mathbb{G}_1$ of prime order q (with $|q| = k$), a generator $g \in \mathbb{G}$, and an efficiently computable function $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$ which is a *nontrivial bilinear map*; namely, (1) for all $\mu, \nu \in \mathbb{G}$ and $a, b \in \mathbb{Z}_q$ we have $\hat{e}(\mu^a, \nu^b) = \hat{e}(\mu, \nu)^{ab}$ and (2) $\hat{e}(g, g)$ is a generator of $\mathbb{G}_1$. (See [11] for a discussion about realizing an algorithm $\mathcal{IG}$ with these properties.) Following the standard terminology, we refer to $\hat{e}$ as a *pairing*.

[5] Note that since only second-preimage resistance is needed to achieve the binding property, a 128-bit output length provides sufficient security.

The *computational bilinear Diffie-Hellman (BDH) problem with respect to* $\mathcal{IG}$ is the following: given $(\mathbb{G}, \mathbb{G}_1, g, \hat{e})$ as output by $\mathcal{IG}$ along with g^α, g^β, and g^γ (for random $\alpha, \beta, \gamma \in \mathbb{Z}_q$), compute $\hat{e}(g,g)^{\alpha\beta\gamma}$. Informally, we say that $\mathcal{IG}$ *satisfies the computational BDH assumption* if the computational BDH assumption with respect to $\mathcal{IG}$ is hard for any PPT algorithm.

The *decisional BDH problem with respect to* $\mathcal{IG}$ is to distinguish between tuples of the form $(g^\alpha, g^\beta, g^\gamma, \hat{e}(g,g)^{\alpha\beta\gamma})$ and $(g^\alpha, g^\beta, g^\gamma, \hat{e}(g,g)^\mu)$ for random $\alpha, \beta, \gamma, \mu \in \mathbb{Z}_q$ (note that $\hat{e}(g,g)^\mu$ is simply a random element of $\mathbb{G}_1$). Informally, we say $\mathcal{IG}$ *satisfies the decisional BDH assumption* if no PPT algorithm can solve the decisional BDH problem with respect to $\mathcal{IG}$ with probability significantly better than $\frac{1}{2}$. We refer to [11] for formal definitions and further discussion.

A concrete IBE scheme. We refer to [7] for the full details and content ourselves with giving only a high-level description of their first IBE scheme here, modified slightly for our eventual application. We assume for simplicity that system parameters $(\mathbb{G}, \mathbb{G}_1, g, \hat{e})$ have already been established by running $\mathcal{IG}(1^k)$ (of course, it is also possible for $\mathcal{IG}$ to be run during key generation). Let $G : \mathbb{G}_1 \to \{0,1\}^k$ be a function whose output is indistinguishable from uniform when its input is uniformly distributed in $\mathbb{G}_1$ (however, G need not expand its input). The IBE scheme is defined as follows:

Setup. Pick random generators $g_1, g_2 \in \mathbb{G}$ and a random $x \in \mathbb{Z}_q$. Set $g_3 = g^x$ and $Z = \hat{e}(g_1, g_3)$. The master public key is $PK = (g, g_1, g_2, g_3, Z)$ and the master secret key is $\mathsf{msk} = x$.

Derive. To derive the secret key for the identity $ID \in \mathbb{Z}_q$ using $\mathsf{msk} = x$, choose a random $t \in \mathbb{Z}_q$, and return the key $SK_{ID} = (g_1^x g_2^t g_3^{t \cdot ID},\ g^t)$.

Encrypt. To encrypt a message $M \in \{0,1\}^k$ with respect to the identity $ID \in \mathbb{Z}_q$, choose a random $s \in \mathbb{Z}_q$, and output the ciphertext $(g^s,\ g_2^s g_3^{s \cdot ID},\ G(Z^s) \oplus M)$.

Decrypt. To decrypt ciphertext (A, B, C) using private key (K_1, K_2), output

$$C \oplus G(\hat{e}(A, K_1)/\hat{e}(B, K_2)). \tag{7.1}$$

Correctness can be easily verified. Security of the above scheme is based on the decisional[6] BDH assumption. For efficiency, the master secret key msk may also contain the discrete logarithms of g_1, g_2 (with respect to g), in which case the key-derivation algorithm requires only two exponentiations with respect to the fixed base g.

7.4. Putting it all together. Given the above, we now fully describe a CCA-secure encryption scheme. In describing the scheme, we focus on the case of encrypting "long" messages (say, 10^4 bits or longer). Focusing on this case allows for a more accurate comparison with the scheme of [42] (which also focuses on this case).

Let $(\mathsf{Mac}, \mathsf{Vrfy})$ denote the CBC-MAC using 128-bit AES as the underlying block cipher. Let $H : \{0,1\}^{448} \to \{0,1\}^{128}$ represent a hash function assumed to be second-preimage resistant (constructed, e.g., via a suitable modification of SHA-1). Let $G : \mathbb{G}_1 \to \{0,1\}^*$ denote a pseudorandom generator with sufficiently long output length (constructed, e.g., by first hashing elements of $\mathbb{G}_1$ and then using a suitable modification of a block/stream cipher). We assume that $|q| > 128$ so that strings in $\{0,1\}^{128}$ may be mapped to $\mathbb{Z}_q$ in a one-to-one manner. Using the IBE scheme

[6] Note that if G instead represents a hard-core predicate for the *computational* BDH assumption, we obtain a scheme (encrypting a single bit) secure under this, possibly weaker, assumption. Running the scheme in parallel we obtain a scheme encrypting longer messages, as needed by our construction.

TABLE 7.1
Efficiency comparison for CCA-secure encryption schemes. See text for discussion.

	Encryption	Decryption	Key generation	Ciphertext overhead
Our scheme	3.5 f-exps.	1.5 exp. + 1 pairing	4 f-exps.	$2 \cdot L_{\text{BG}} + 704$
KD-CS [42]	3.5 f-exps.	1.5 exps.	3 f-exps.	$2 \cdot L_{\text{DDH}} + 128$

outlined above, we obtain the following (we assume that $G, H, q, \mathbb{G}, \mathbb{G}_1, \hat{e}, g$, and $\hat{e}(g,g)$ are provided as universal parameters):

Key generation. Choose $\alpha_1, \alpha_2, x \leftarrow \mathbb{Z}_q$ and set $g_1 = g^{\alpha_1}$, $g_2 = g^{\alpha_2}$, and $g_3 = g^x$. Also set $Z = \hat{e}(g,g)^{\alpha_1 x}$. Finally, choose hash function h from a family of pairwise-independent hash functions. The public key is $PK = (g_1, g_2, g_3, Z, h)$ and the secret key is $SK = (\alpha_1, \alpha_2, x)$.

Encryption. To encrypt message M using public key (g_1, g_2, g_3, Z, h), first choose random $r \in \{0,1\}^{448}$ and set $k_1 = h(r)$ and $ID = H(r)$. Choose random $s \in \mathbb{Z}_q$ and then set $C = (g^s, g_2^s g_3^{s \cdot ID}, G(Z^s) \oplus (M \circ r))$. Output the ciphertext

$$\langle ID, C, \mathsf{Mac}_{k_1}(C) \rangle.$$

Decryption. To decrypt ciphertext $\langle ID, C, \mathsf{tag} \rangle$, first parse C as $(A, B, \hat{C})$. Then pick a random $t \in \mathbb{Z}_q$ and compute the values $(M \circ r) = \hat{C} \oplus G(\hat{e}(A^{\alpha_1 x + t(\alpha_2 + x \cdot ID)} B^{-t}, g))$. Set $k_1 = h(r)$. If $\mathsf{Vrfy}_{k_1}(C, \mathsf{tag}) \stackrel{?}{=} 1$ and $H(r) \stackrel{?}{=} ID$ output M; otherwise, output $\perp$.

We have changed the steps used in decryption for efficiency purposes, but it is easily checked that decryption yields the same result as deriving a secret key for identity ID and then using this key to decrypt C.

We tabulate the efficiency of our scheme, and compare it to the Kurosawa–Desmedt variant of Cramer–Shoup encryption [42, 21] (which we refer to as KD–CS) in Table 7.1. In tabulating computational efficiency, "private-key" operations (that is, evaluations of G, H, and h) and group multiplications are ignored; "exp" stands for exponentiation; "f-exp" refers to exponentiation relative to a fixed base (where efficiency can be improved using precomputation); and one multiexponentiation is counted as 1.5 exponentiations [46, p. 618]. Ciphertext overhead is the difference (in bits) between the lengths of the ciphertext and the message. L_{BG} is the bit-length of an element in a group $\mathbb{G}$ suitable for our scheme, and L_{DDH} is the bit-length of an element in a group suitable for the KD–CS scheme.

Although performance of the two systems looks similar, efficiency of the KD–CS scheme scales better with the security parameter. To see why, fix the measure of hardness to be the difficulty of computing discrete logarithms in the respective groups. (Although the underlying computational problems used to prove security of the above two schemes are not known to be equivalent to the discrete logarithm problem, in each case the best currently known algorithms for solving the problem rely on a discrete logarithm computation.) Consider two concrete settings:

80-bit security. Suppose we wish to use groups in which solving the discrete logarithm problem (using the best currently known algorithms) is roughly equivalent to the security attained by 80-bit symmetric-key cryptography.

- Our scheme can use groups based on the elliptic curves suggested by Miyaji, Nakabayashi, and Takano[48]. In this case, the discrete logarithm problem in a group $\mathbb{G}$ in which elements can be written using $L_{\text{BG}} = \log q$ bits (q prime) can be reduced to a discrete logarithm problem in $\mathbb{F}^*_{q^6}$. (See [13, section

4.3].) 80-bit security for the latter is obtained by setting $q^6 \approx 2^{1024}$ [44, 1] (specifically, our numbers throughout this discussion are taken from [1, Table 2]). We thus need $L_{\text{BG}} \approx 1024/6 \approx 171$.

- The KD–CS scheme can use standard elliptic curve groups, for which the best-known algorithm for computing discrete logarithms is Pollard's rho algorithm that runs in time proportional to $\sqrt{q}$ for groups of order q. This gives $L_{\text{DDH}} = \log q \approx 160$ [1].

We see that for this level of security, both schemes have ciphertexts of roughly the same length and group operations take roughly the same amount of time. Still, the KD–CS scheme outperforms our scheme (even if by a relatively small margin in some cases) in all parameters; this is especially true for decryption since a pairing calculation is computationally expensive compared to an exponentiation.

256-bit security. Now, the KD–CS scheme performs even better relative to ours.

- For our scheme, we can use groups based on a certain class of elliptic curves suggested by Barreto and Naehrig [4] (note that such curves will give worse performance for the case of 80-bit security considered above). Now, the discrete logarithm problem in a group $\mathbb{G}$ in which elements can be written using $L_{\text{BG}} = \log q$ bits (q prime) can be reduced to a discrete logarithm problem in $\mathbb{F}^*_{q^{12}}$. 256-bit security for the latter is obtained by setting $q^{12} \approx 2^{15360}$. We thus need $L_{\text{BG}} \approx 15386/12 \approx 1280$.
- Using the same the analysis as before, the KD–CS scheme can use $L_{\text{DDH}} = \log q \approx 512$.

We conclude that for currently acceptable settings of the security parameter the schemes have comparable performance, though the KD–CS scheme is more efficient. We stress that our goal is not to displace the KD–CS system but rather to show another approach to building practical CCA-secure systems. Our construction also has advantages not present in the KD–CS scheme, such as being readily amenable to a threshold implementation [10].

8. Conclusions. We presented in this paper new paradigms for constructing CCA-secure public-key encryption schemes using IBE as a building block. Our paradigms extend to enable constructions of CCA-secure (hierarchical) IBF schemes as well. Instantiating our constructions with an existing IBE system yields a CCA-secure encryption scheme whose performance, for standard settings of the security parameter, is competitive with the best CCA-secure schemes known to date.

Appendix. Review of standard definitions. We provide the standard definitions of public-key encryption schemes and their security against adaptive chosen-ciphertext attacks, as well as appropriate definitions for strong one-time signature schemes and message authentication codes.

A.1. Public-key encryption.

DEFINITION 7. *A* public-key encryption scheme *is a triple of* PPT *algorithms* $(\mathsf{Gen}, \mathcal{E}, \mathcal{D})$ *such that the following hold:*

- *The randomized* key generation algorithm Gen *takes as input a security parameter* 1^k *and outputs a public key* PK *and a secret key* SK.
- *The randomized* encryption algorithm $\mathcal{E}$ *takes as input a public key* PK *and a message* m *(in some implicit message space), and outputs a ciphertext* C. *We write* $C \leftarrow \mathcal{E}_{PK}(m)$.

- *The (possibly randomized)* decryption algorithm $\mathcal{D}$ *takes as input a ciphertext* C *and a secret key* SK*. It returns a message* m *or the symbol* $\perp$ *(which is not in the message space). We write* $m \leftarrow \mathcal{D}_{SK}(C)$*.*

We require that for all (PK, SK) *output by* Gen*, all* m *in the message space, and all* C *output by* $\mathcal{E}_{PK}(m)$ *we have* $\mathcal{D}_{SK}(C) = m$*.*

We recall the standard definition of security against adaptive chosen-ciphertext attacks (cf. [2]).

DEFINITION 8. *A public-key encryption scheme* Π *is* secure against adaptive chosen-ciphertext attacks *(i.e., "CCA-secure") if the advantage of any* PPT *adversary* $\mathcal{A}$ *in the following game is negligible in the security parameter* k*:*

1. $\mathsf{Gen}(1^k)$ *outputs* (PK, SK)*. Adversary* $\mathcal{A}$ *is given* 1^k *and* PK*.*
2. *The adversary may make polynomially many queries to a decryption oracle* $\mathcal{D}_{SK}(\cdot)$*.*
3. *At some point,* $\mathcal{A}$ *outputs two messages* m_0, m_1 *with* $|m_0| = |m_1|$*. A bit* b *is randomly chosen and the adversary is given a "challenge ciphertext"* $C^* \leftarrow \mathcal{E}_{PK}(m_b)$*.*
4. $\mathcal{A}$ *may continue to query its decryption oracle* $\mathcal{D}_{SK}(\cdot)$ *except that it may not request the decryption of* C^**.*
5. *Finally,* $\mathcal{A}$ *outputs a guess* b'*.*

We say that $\mathcal{A}$ succeeds *if* $b' = b$*, and we denote the probability of this event by* $\Pr^{\mathsf{PKE}}_{\mathcal{A},\Pi}[\mathsf{Succ}]$*. The adversary's* advantage *is defined as* $\mathsf{Adv}^{\mathsf{PKE}}_{\mathcal{A},\Pi}(k) \stackrel{\text{def}}{=} \left|\Pr^{\mathsf{PKE}}_{\mathcal{A},\Pi}[\mathsf{Succ}] - 1/2\right|$*.*

A.2. Signatures and MACs. We remind the reader of the standard functional definitions for signature schemes and MACs, followed by a definition of strong one-time security appropriate for each.

DEFINITION 9. *A* signature scheme *is a triple of* PPT *algorithms* $(\mathcal{G}, \mathsf{Sign}, \mathsf{Vrfy})$ *such that the following hold:*

- *The randomized* key generation algorithm $\mathcal{G}$ *takes as input the security parameter* 1^k *and outputs a verification key* vk *and a signing key* sk*. We assume for simplicity that the length of* vk *is fixed for any given value of* k*.*
- *The* signing algorithm Sign *takes as input a signing key* sk *and a message* m *(in some implicit message space), and outputs a signature* σ*. We write* $\sigma \leftarrow \mathsf{Sign}_{sk}(m)$*.*
- *The* verification algorithm Vrfy *takes as input a verification key* vk*, a message* m*, and a signature* σ*, and outputs a bit* $b \in \{0,1\}$ *(where* $b = 1$ *signifies "acceptance" and* $b = 0$ *signifies "rejection"). We write this as* $b := \mathsf{Vrfy}_{vk}(m, \sigma)$*.*

We require that for all (vk, sk) *output by* $\mathcal{G}$*, all* m *in the message space, and all* σ *output by* $\mathsf{Sign}_{sk}(m)$*, we have* $\mathsf{Vrfy}_{vk}(m, \sigma) = 1$*.*

A MAC is similar in spirit to a signature scheme, except that here the signing key and verification key are identical. We review the definition for convenience.

DEFINITION 10. *A* message authentication code *is a pair of* PPT *algorithms* $(\mathsf{Mac}, \mathsf{Vrfy})$ *such that the following hold:*

- *The* tagging algorithm Mac *takes as input a key* $sk \in \{0,1\}^k$ *(where* k *is the security parameter) and a message* m *(in some implicit message space). It outputs a tag* tag*, and we denote this by* $\mathsf{tag} \leftarrow \mathsf{Mac}_{sk}(m)$*.*
- *The* verification algorithm Vrfy *takes as input a key* sk*, a message* m*, and a tag* tag*; it outputs a bit* $b \in \{0,1\}$ *(where* $b = 1$ *signifies "acceptance" and* $b = 0$ *signifies "rejection"). We write this as* $b := \mathsf{Vrfy}_{sk}(m, \mathsf{tag})$*.*

We require that for all sk, all m in the message space, and all tag *output by* $\mathsf{Mac}_{sk}(m)$, *we have* $\mathsf{Vrfy}_{sk}(m, \mathsf{tag}) = 1$.

We next turn to definitions of security for signature schemes and MACs. The definition of security is analogous in each case: the adversary should be unable to forge a valid message/signature (resp., message/tag) pair, after receiving a signature (resp., tag) on any *single* message m of the adversary's choice. Note that we require so-called *strong* security in each case, so that it should be infeasible for the adversary to generate even a different signature (resp., tag) on the same message m.

DEFINITION 11. *A signature scheme* Sig *is a* strong one-time signature scheme *if the success probability of any* PPT *adversary* $\mathcal{A}$ *in the following game is negligible in the security parameter* k:

1. $\mathcal{G}(1^k)$ *outputs* (vk, sk) *and the adversary is given* 1^k *and* vk.
2. $\mathcal{A}(1^k, vk)$ *may do one of the following:*
   (a) $\mathcal{A}$ *may output a pair* (m^*, σ^*) *and halt. In this case* (m, σ) *are undefined.*
   (b) $\mathcal{A}$ *may output a message* m, *and is then given in return* $\sigma \leftarrow \mathsf{Sign}_{sk}(m)$. *Following this,* $\mathcal{A}$ *outputs* (m^*, σ^*).

We say the adversary succeeds *if* $\mathsf{Vrfy}_{vk}(m^*, \sigma^*) = 1$ *but* $(m^*, \sigma^*) \neq (m, \sigma)$ *(assuming* (m, σ) *are defined). We stress that the adversary may succeed even if* $m^* = m$.

DEFINITION 12. *A message authentication code* $(\mathsf{Mac}, \mathsf{Vrfy})$ *is a* strong one-time message authentication code *if the success probability of any* PPT *adversary* $\mathcal{A}$ *in the following game is negligible in the security parameter* k:

1. *A random key* $sk \in \{0,1\}^k$ *is chosen.*
2. $\mathcal{A}(1^k)$ *may do one of the following:*
   (a) $\mathcal{A}$ *may output* (m^*, tag^*). *In this case,* (m, tag) *are undefined.*
   (b) $\mathcal{A}$ *may output a message* m *and is then given in return* $\mathsf{tag} \leftarrow \mathsf{Mac}_{sk}(m)$. *Following this,* $\mathcal{A}$ *outputs* (m^*, tag^*).

We say the adversary succeeds *if* $\mathsf{Vrfy}_{sk}(m^*, \mathsf{tag}^*) = 1$ *but* $(m^*, \mathsf{tag}^*) \neq (m, \mathsf{tag})$ *(assuming* (m, tag) *are defined). We stress that the adversary may succeed even if* $m^* = m$.

Acknowledgments. We thank Eu-Jin Goh for pointing out that our techniques imply a conversion from IBE to nonadaptive CCA-security with essentially no overhead. We also thank the anonymous referees for their helpful comments, and in particular for suggesting a way to simplify the presentation of the proof of Theorem 2.

REFERENCES

[1] E. BARKER, W. BARKER, W. BURR, W. POLK, AND M. SMID, *Recommendation for key management—Part 1: General*, NIST Special Publication 800-57, August 2005, National Institute of Standards and Technology, available online at http://csrc.nist.gov/publications/nistpubs/800-57/SP800-57-Part1.pdf.

[2] M. BELLARE, A. DESAI, D. POINTCHEVAL, AND P. ROGAWAY, *Relations among notions of security for public-key encryption schemes*, in Advances in Cryptology—Crypto '98, Lecture Notes in Comput. Sci. 1462, Springer-Verlag, New York, 1998, pp. 26–45.

[3] M. BELLARE AND P. ROGAWAY, *Random oracles are practical: A paradigm for designing efficient protocols*, in Proceedings of the 1st ACM Conference on Computer and Communications Security, ACM, New York, 1993, pp. 62–73.

[4] P. BARRETO AND M. NAEHRIG, *Pairing-friendly elliptic curves of prime order*, in Selected Areas in Cryptography—SAC 2005, Lecture Notes in Comput. Sci. 3897, Springer-Verlag, New York, 2006, pp. 319–331.

[5] D. Bleichenbacher, *Chosen-ciphertext attacks against protocols based on the RSA encryption standard PKCS #1*, in Advances in Cryptology—Crypto '98, Lecture Notes in Comput. Sci. 1462, Springer-Verlag, New York, 1998, pp. 1–12.

[6] M. Blum, P. Feldman, and S. Micali, *Noninteractive zero-knowledge and its applications*, in Proceedings of the 20th ACM Symposium on Theory of Computing, ACM, New York, 1988, pp. 103–112.

[7] D. Boneh and X. Boyen, *Efficient selective-ID secure identity-based encryption without random oracles*, in Advances in Cryptology—Eurocrypt 2004, Lecture Notes in Comput. Sci. 3027, Springer-Verlag, New York, 2004, pp. 223–238. Full version available at http://eprint.iacr.org/2004/172.

[8] D. Boneh and X. Boyen, *Secure identity-based encryption without random oracles*, in Advances in Cryptology—Crypto 2004, Lecture Notes in Comput. Sci. 3152, Springer-Verlag, New York, 2004, pp. 443–459. Full version available at http://eprint.iacr.org/2004/173.

[9] D. Boneh, X. Boyen, and E.-J. Goh, *Hierarchical identity-based encryption with constant-size ciphertexts*, in Advances in Cryptology—Eurocrypt 2005, Lecture Notes in Comput. Sci. 3494, Springer-Verlag, New York, 2005, pp. 440–456. Full version available at http://eprint.iacr.org/2005/015.

[10] D. Boneh, X. Boyen, and S. Halevi, *Chosen-ciphertext secure public-key threshold encryption without random oracles*, in Topics in Cryptology—CT-RSA 2006, Lecture Notes in Comput. Sci. 3860, Springer-Verlag, New York, 2006, pp. 226–243.

[11] D. Boneh and M. Franklin, *Identity-based encryption from the Weil pairing*, SIAM J. Comput., 32 (2003), pp. 586–615.

[12] D. Boneh and J. Katz, *Improved efficiency for CCA-secure cryptosystems built using identity-based encryption*, in Topics in Cryptology—CT-RSA 2005, Lecture Notes in Comput. Sci. 3376, Springer-Verlag, New York, 2005, pp. 87–103.

[13] D. Boneh, B. Lynn, and H. Shacham, *Short signatures from the Weil pairing*, J. Cryptology, 17 (2004), pp. 297–319.

[14] X. Boyen, Q. Mei, and B. Waters, *Direct chosen ciphertext security from identity-based techniques*, in Proceedings of the 12th ACM Conference on Computer and Communications Security, ACM, New York, 2005, pp. 320–329.

[15] R. Canetti, *Universally composable security: A new paradigm for cryptographic protocols*, in Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2001, pp. 136–145. Full version available at http://eprint.iacr.org/2000/067.

[16] R. Canetti, O. Goldreich, and S. Halevi, *The random oracle methodology, revisited*, J. ACM, 51 (2004), pp. 557–594.

[17] R. Canetti, S. Halevi, and J. Katz, *A forward-secure public-key encryption scheme*, in Advances in Cryptology—Eurocrypt 2003, Lecture Notes in Comput. Sci. 2656, Springer-Verlag, New York, 2003, pp. 255–271. Full version available at http://eprint.iacr.org/2003/083. J. Cryptology, to appear.

[18] R. Canetti, S. Halevi, and J. Katz, *Chosen-ciphertext security from identity-based encryption*, in Advances in Cryptology—Eurocrypt 2004, Lecture Notes in Comput. Sci. 3027, Springer-Verlag, New York, 2004, pp. 207–222.

[19] R. Canetti, H. Krawczyk, and J.B. Nielsen, *Relaxing chosen ciphertext security*, in Advances in Cryptology—Crypto 2003, Lecture Notes in Comput. Sci. 2656, Springer-Verlag, New York, 2003, pp. 565–582.

[20] C. Cocks, *An identity-based encryption scheme based on quadratic residues*, in Cryptography and Coding, Lecture Notes in Comput. Sci. 2260, Springer-Verlag, New York, 2001, pp. 360–363.

[21] R. Cramer and V. Shoup, *Design and analysis of practical public-key encryption schemes secure against adaptive chosen-ciphertext attack*, SIAM J. Comput., 33 (2003), pp. 167–226.

[22] R. Cramer and V. Shoup, *Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption*, in Advances in Cryptology—Eurocrypt 2002, Lecture Notes in Comput. Sci. 2332, Springer-Verlag, New York, 2002, pp. 45–64.

[23] J. Camenisch and V. Shoup, *Practical verifiable encryption and decryption of discrete logarithms*, in Advances in Cryptology—Crypto 2003, Lecture Notes in Comput. Sci. 2729, Springer-Verlag, New York, 2003, pp. 126–144.

[24] I. Damgård, T. P. Pedersen, and B. Pfitzmann, *On the existence of statistically-hiding bit commitment schemes and fail-stop signatures*, in Advances in Cryptology—Crypto '93, Lecture Notes in Comput. Sci. 773, Springer-Verlag, New York, 1993, pp. 250–265.

[25] Y. Desmedt and Y. Frankel, *Threshold cryptosystems*, in Advances in Cryptology—Crypto '89, Lecture Notes in Comput. Sci. 435, Springer-Verlag, New York, 1990, pp. 307–315.

[26] G. Di Crescenzo, Y. Ishai, and R. Ostrovsky, *Noninteractive and nonmalleable commitment*, in Proceedings of the ACM Symposium on Theory of Computing, ACM, New York, 1998, pp. 141–150.

[27] G. Di Crescenzo, J. Katz, R. Ostrovsky, and A. Smith, *Efficient and noninteractive nonmalleable commitment*, in Advances in Cryptology—Eurocrypt 2001, Lecture Notes in Comput. Sci. 2045, Springer-Verlag, New York, 2001, pp. 40–59.

[28] D. Dolev, C. Dwork, and M. Naor, *Nonmalleable cryptography*, SIAM J. Comput., 30 (2000), pp. 391–437.

[29] E. Elkind and A. Sahai, *A unified methodology for constructing public-key encryption schemes secure against adaptive chosen-ciphertext attack*, available online at http://eprint.iacr.org/2002/042.

[30] S. Even, O. Goldreich, and S. Micali, *On-line/off-line digital signatures*, J. Cryptology, 9 (1996), pp. 35–67.

[31] U. Feige, D. Lapidot, and A. Shamir, *Multiple noninteractive zero knowledge proofs under general assumptions*, SIAM J. Comput., 29 (1999), pp. 1–28.

[32] R. Gennaro and Y. Lindell, *A framework for password-based authenticated key exchange*, in Advances in Cryptology—Eurocrypt 2003, Lecture Notes in Comput. Sci. 2656, Springer-Verlag, New York, 2003, pp. 524–543.

[33] C. Gentry and A. Silverberg, *Hierarchical identity-based cryptography*, in Advances in Cryptology—Asiacrypt 2002, Lecture Notes in Comput. Sci. 2501, Springer-Verlag, New York, 2002, pp. 548–566.

[34] C. Gentry, *Practical identity-based encryption without random oracles*, in Advances in Cryptology—Eurocrypt 2006, Lecture Notes in Comput. Sci. 4004, Springer-Verlag, New York, 2006, pp. 445–464.

[35] O. Goldreich, *A uniform complexity treatment of encryption and zero-knowledge*, J. Cryptology, 6 (1993), pp. 21–53.

[36] O. Goldreich, *Foundations of Cryptography, Vol.* 2: *Basic Applications*, Cambridge University Press, Cambridge, UK, 2004.

[37] O. Goldreich, Y. Lustig, and M. Naor, *On chosen-ciphertext security of multiple encryptions*, available online at http://eprint.iacr.org/2002/089.

[38] S. Goldwasser and S. Micali, *Probabilistic encryption*, J. Comput. System Sci., 28 (1984), pp. 270–299.

[39] S. Halevi and S. Micali, *Practical and provably-secure commitment schemes from collision-free hashing*, in Advances in Cryptology—Crypto '96, Lecture Notes in Comput. Sci. 1109, Springer-Verlag, New York, 1996, pp. 201–215.

[40] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby, *A pseudorandom generator from any one-way function*, SIAM J. Comput., 28 (1999), pp. 1364–1396.

[41] J. Horwitz and B. Lynn, *Toward hierarchical identity-based encryption*, in Advances in Cryptology—Eurocrypt 2002, Lecture Notes in Comput. Sci. 2332, Springer-Verlag, New York, 2002, pp. 466–481.

[42] K. Kurosawa and Y. Desmedt, *A new paradigm of hybrid encryption scheme*, in Advances in Cryptology—Crypto 2004, Lecture Notes in Comput. Sci. 3152, Springer-Verlag, New York, 2004, pp. 426–442.

[43] L. Lamport, *Constructing Digital Signatures from a One-Way Function*, Technical Report CSL-98, SRI International, Menlo Park, CA, 1979.

[44] A. K. Lenstra and E. R. Verheul, *Selecting cryptographic key sizes*, J. Cryptology, 14 (2001), pp. 255–293.

[45] P. MacKenzie, M. Reiter, and K. Yang, *Alternatives to nonmalleability: Definitions, constructions, and applications*, in Proceedings of the 1st Theory of Cryptography Conference—TCC 2004, Lecture Notes in Comput. Sci. 2951, Springer-Verlag, New York, 2004, pp. 171–190.

[46] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, FL, 1997.

[47] S. Micali, C. Rackoff, and B. Sloan, *The notion of security for probabilistic cryptosystems*, SIAM J. Comput., 17 (1988), pp. 412–426.

[48] A. Miyaji, M. Nakabayashi, and S. Takano, *New explicit constructions of elliptic curve traces for FR-reduction*, IEICE Trans. Fundamentals, E84-A (2001), pp. 1234–1243.

[49] M. Naor and M. Yung, *Public-key cryptosystems provably-secure against chosen-ciphertext attacks*, in Proceedings of the 22nd ACM Symposium on Theory of Computing, ACM, New York, 1990, pp. 427–437.

[50] C. Rackoff and D. Simon, *Noninteractive zero-knowledge proof of knowledge and chosen ciphertext attack*, in Advances in Cryptology—Crypto '91, Lecture Notes in Comput. Sci. 576, Springer-Verlag, New York, 1992, pp. 433–444.

[51] A. Sahai, *Nonmalleable noninteractive zero knowledge and adaptive chosen-ciphertext security*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1999, pp. 543–553.

[52] A. Shamir, *Identity-based cryptosystems and signature schemes*, in Advances in Cryptology—Crypto '84, Lecture Notes in Comput. Sci. 196, Springer-Verlag, New York, 1985, pp. 47–53.

[53] V. Shoup, *Why chosen ciphertext security matters*, IBM Research Report RZ 3076, IBM, Armonk, NY, 1998. Available online at http://www.shoup.net/papers.

[54] D. R. Stinson, *Universal hashing and authentication codes*, Des. Codes Cryptogr., 4 (1994), pp. 369–380.

[55] Y. Watanabe, J. Shikata, and H. Imai, *Equivalence between semantic security and indistinguishability against chosen-ciphertext attacks*, in Public-Key Cryptography 2003, Lecture Notes in Comput. Sci. 2567, Springer-Verlag, New York, 2003, pp. 71–84.

[56] B. Waters, *Efficient identity-based encryption without random oracles*, in Advances in Cryptology—Eurocrypt 2005, Lecture Notes in Comput. Sci. 3494, Springer-Verlag, New York, 2005, pp. 114–127.

[57] M. N. Wegman and J. L. Carter, *New hash functions and their use in authentication and set equality*, J. Comput. System Sci., 22 (1981), pp. 265–279.

SIAM J. COMPUT.
Vol. 36, No. 5, pp. 1329–1341

A DETERMINISTIC ALGORITHM FOR FINDING ALL MINIMUM k-WAY CUTS*

YOKO KAMIDOI†, NORIYOSHI YOSHIDA†, AND HIROSHI NAGAMOCHI‡

Abstract. Let $G = (V, E)$ be an edge-weighted undirected graph with n vertices and m edges. We present a deterministic algorithm to compute a minimum k-way cut of G for a given k. Our algorithm is a divide-and-conquer method based on a procedure that reduces an instance of the minimum k-way cut problem to $O(n^{2k-5})$ instances of the minimum $(\lfloor (k+\sqrt{k})/2 \rfloor + 1)$-way cut problem, and can be implemented to run in $O(n^{4k/(1-1.71/\sqrt{k})-31})$ time. With a slight modification, the algorithm can find all minimum k-way cuts in $O(n^{4k/(1-1.71/\sqrt{k})-16})$ time.

Key words. minimum cut, multiway cut, divide-and-conquer

AMS subject classifications. 05C85, 68R10, 68W05

DOI. 10.1137/050631616

1. Introduction.

For an edge-weighted graph $G = (V, E)$, a subset F of edges is called a *k-way cut* if removal of F from G results in at least k connected components. The *minimum k-way cut problem* asks to find a minimum weight k-way cut in G. Given k vertices (called *terminals*), a k-way cut F is called a *k-terminal cut* if no two terminals are in the same connected component after removal of F. The problem of finding a minimum weight k-terminal cut is called the *minimum k-terminal cut problem*. These problems have several important applications such as VLSI design [1, 6, 26], task allocation in distributed computing systems [25, 34], graph strength [4, 9, 32], and network reliability [3, 35].

For $k = 2$, the minimum 2-terminal cut problem in a graph can be solved by applying a maximum flow algorithm. Let $F(n, m)$ denote the time complexity of a maximum flow algorithm in an edge-weighted graph with n vertices and m edges. The complexity $F(n, m)$ was found to be $O(n^3)$ in [24] and $O(nm \log(n^2/m))$ in [7]. Dahlhaus et al. [5] proved that the minimum k-terminal cut problem is NP-hard for any fixed $k \geq 3$. Several approximation algorithms have been proposed [2, 5, 21], among which a 1.3438-approximation algorithm is obtained by Karger et al. [21]. An extension of this problem to a general setting defined by submodular set functions can be found in the articles by Zhao, Nagamochi, and Ibaraki [39, 40]. For planar graphs, the minimum k-terminal cut problem admits a polynomial time algorithm [5], and currently an $O(k4^k n^{2k-4} \log n)$ time algorithm in [14] and an $O((k-\frac{3}{2})^{k-1}(n-k)^{2k-4}[nk - \frac{3}{2}k^2 + \frac{1}{2}k]\log(n-k))$ time algorithm in [37] are known.

On the other hand, Goldschmidt and Hochbaum [8] proved that the minimum k-way cut problem is NP-hard if k is an input parameter but admits a polynomial time algorithm if k is regarded as a constant. The minimum 2-way cut problem (i.e.,

*Received by the editors May 15, 2005; accepted for publication (in revised form) June 12, 2006; published electronically December 21, 2006. A preliminary version of this paper appeared in *Proceedings of the 4th Japanese–Hungarian Symposium on Discrete Mathematics and Its Applications*, Budapest, Hungary, 2005, pp. 224–233. This research was supported by the Scientific Grant-in-Aid from Ministry of Education, Science, Sports and Culture of Japan.

http://www.siam.org/journals/sicomp/36-5/63161.html

†Faculty of Information Sciences, Hiroshima City University, 3-4-1, Ozuka-Higashi, Asaminami-ku, Hiroshima 731-3194, Japan (yoko@ce.hiroshima-cu.ac.jp, nyoshida@khaki.plala.or.jp).

‡Department of Applied Mathematics and Physics, Kyoto University, Sakyo, Kyoto 606-8501, Japan (nag@amp.i.kyoto-u.ac.jp).

the problem of computing edge-connectivity) can be solved by $O(nm + n^2 \log n)$ and $O(nm \log(n^2/m))$ time deterministic algorithms [12, 28] and by $O(n^2(\log n)^3)$ and $O(m(\log n)^3)$ time randomized algorithms [20, 22, 23]. Approximation algorithms for a minimum k-way cut problem of G have been proposed in [16, 33, 38]. Saran and Vazirani [33] first proposed a $2(1-1/k)$-approximation algorithm for the minimum k-way cut problem, which runs in $O(nF(n,m))$ time. Kapoor [16] also gave a $2(1-1/k)$-approximation algorithm for the minimum k-way cut problem of G, which requires $O(k(nm+n^2 \log n))$ time. Zhao et al. [38] also presented an approximation algorithm by using a set of minimum 3-way cuts. Their algorithm has the performance ratio $2-3/k$ for an odd k and $2-(3k-4)/(k^2-k)$ for an even k, and runs in $O(kmn^3 \log(n^2/m))$ time. Approximation algorithms for a multiway cut problem defined by submodular set functions are discussed in the articles by Zhao, Nagamochi, and Ibaraki [39, 40].

Goldschmidt and Hochbaum [8] presented an $O(n^{k^2/2-3k/2+4}F(n,m))$ time algorithm for solving the minimum k-way cut problem. This running time is polynomial for any fixed k. The algorithm is based on a divide-and-conquer approach. Suppose that we can choose a family $\mathcal{X}$ of subsets of V such that at least one subset $X \in \mathcal{X}$ has a property that a minimum $(k-1)$-way cut $\{V_1, \ldots, V_{k-1}\}$ in the subgraph $G[V-X]$ induced by $V-X$ gives rise to a minimum k-way cut $\{X, V_1, \ldots, V_{k-1}\}$ in the original graph G. Then we can find a minimum k-way cut in G by solving the $(k-1)$-way cut problem instances $G[V-X]$ for all $X \in \mathcal{X}$. Goldschmidt and Hochbaum [8] proved that such a family $\mathcal{X}$ of $O(n^{2k-3})$ subsets of V can be found in polynomial time, implying an $O(n^{O(k^2)})$ time algorithm for the minimum k-way cut problem.

For small $k \leq 6$ or planar graphs, faster algorithms have been obtained [16, 17, 18, 29, 30, 31]. For $k \leq 6$, the above family $\mathcal{X}$ can be constructed in polynomial time by collecting $O(n)$ subsets of V, and an $O(mn^k \log(n^2/m))$ time algorithm is known [29, 30, 31]. For planar graphs, Hartvigsen [13] gave an $O(n^{2k-1})$ time algorithm, and Nagamochi and Ibaraki [30] and Nagamochi, Katayama, and Ibaraki [31] showed that the problem can be solved in $O(n^k)$ time if $k \leq 6$. The case of unweighted planar graphs with $k = 3$ can be solved in $O(n \log n)$ time [15].

Randomized algorithms have been developed for the k-way cut problem. Karger and Stein [23] proposed a Monte Carlo algorithm for the minimum k-way cut problem which runs in $O(n^{2(k-1)} \log^3 n)$ time. Afterward, Levine [27] gave a Monte Carlo algorithm for $k \leq 6$ that runs in $O(mn^{k-2} \log^3 n)$ time. However, for a general k and a general graph G, no faster deterministic algorithm has been discovered since Goldschmidt and Hochbaum [8] found an $O(n^{O(k^2)})$ time algorithm.

In this paper, we present the first $O(n^{O(k)})$ time deterministic algorithm to compute a minimum k-way cut of G. Our algorithm is based on a divide-and-conquer method which consists of a procedure that reduces an instance of the minimum k-way cut problem to $O(n^{2k-5})$ instances of the minimum $(\lfloor (k+\sqrt{k})/2 \rfloor + 1)$-way cut problem, and can be implemented to run in $O(n^{4k/(1-1.71/\sqrt{k})-31})$ time. With a slight modification, we can also find all minimum k-way cuts in $O(n^{4k/(1-1.71/\sqrt{k})-16})$ time.

The paper is organized as follows. Section 2 introduces notation and reviews basic properties of 2-way cuts. Section 3 presents our divide-and-conquer algorithm, assuming an efficient procedure for computing a family $\mathcal{X}$ of subsets required to reduce a given problem instance, which is discussed in section 5, after proving a key property on crossing 2-way cuts in section 4. Section 6 analyzes the runtime of our algorithm. Section 7 shows how to modify the algorithm so that all minimum k-way cuts can be computed, and section 8 makes some concluding remarks.

2. Preliminaries. Let $G = (V, E)$ stand for an edge-weighted undirected graph consisting of a vertex set V and an edge set E with an edge weight function $cost : E \to R^+$, where R^+ is the set of nonnegative real numbers. Let $n = |V|$ be the number of vertices and $m = |E|$ be the number of edges. We may simply call G a graph. Let $comp(G)$ denote the number of connected components in G. An edge $e \in E$ with end vertices u and v may be denoted by $e = (u, v)$, and its weight is denoted by $cost(e)$. For a nonempty subset $F \subseteq E$, we let $cost(F)$ denote $\sum_{e \in F} cost(e)$. Let $X_1, X_2, \ldots, X_p$ be mutually disjoint subsets of V.

We denote the set of edges $e = (u, v)$ with $u \in X_i$ and $v \in X_j$ for some $i \neq j$ by $(X_1; X_2; \ldots; X_p)$, and the sum of the weights of these edges by $cost(X_1; X_2; \ldots; X_p)$, which is defined to be 0 if $(X_1; X_2; \ldots; X_p) = \emptyset$. For a subset X of V, we may denote $f(X) = cost(X; V-X)$, where f is called a *cut function* of G and satisfies the following identities:

$$f(X) + f(Y) = f(X \cap Y) + f(X \cup Y) + 2cost(X - Y, Y - X) \quad \text{for all } X, Y \subseteq V, \tag{2.1}$$

$$f(X) + f(Y) = f(X - Y) + f(Y - X) + 2cost(X \cap Y, V - (X \cup Y)) \quad \text{for all } X, Y \subseteq V. \tag{2.2}$$

Let F be a subset of E in G. We denote by $G - F$ the graph obtained from G by deleting edges in F. We call F a *k-way cut* if $comp(G - F) \geq k$. A k-way cut F is *minimum* if it has the minimum $cost(F)$ over all k-way cuts. Given a graph G and an integer $k(\geq 2)$, the *minimum k-way cut problem* asks to find a minimum k-way cut in G. We denote the cost of a minimum k-way cut in G by $opt(G, k)$. Note that $opt(G, k) = 0$ if and only if the set of edges with positive weights induces a subgraph of G with at least k connected components. Any inclusionwise minimal k-way cut F is given by $F = (X_1; X_2; \ldots; X_k)$ for some partition $\{X_1, X_2, \ldots, X_k\}$ of V. Conversely, for any partition $\{V_1, V_2, \ldots, V_k\}$ of V, $F' = (V_1; V_2; \ldots; V_k)$ is a k-way cut, where possibly $comp(G - F') > k$. For a set $\mathcal{C}$ of subsets F of E, we denote the union $\cup_{F \in \mathcal{C}} F$ by $E(\mathcal{C})$.

Given a nonempty vertex subset X, let $G[X] = (X, E_X)$ be the subgraph of G induced by X, where $G[X]$ has the edge weight function $cost_X : E_X \to R^+$, which is defined such that $cost_X(e) = cost(e)$ for every edge $e \in E_X$. For a subset Y of vertices of V, we denote $V - Y$ by $\overline{Y}$ if V is clear from the context.

Given p mutually disjoint nonempty subsets $T_1, T_2, \ldots, T_p$ of V, called *terminal sets*, a subset $F \subseteq E$ is called a *$(T_1, T_2, \ldots, T_p)$-terminal cut* of G if the removal of F from G disconnects each terminal set from the others. A $(T_1, T_2, \ldots, T_p)$-terminal cut is called minimum if it has the minimum $cost(F)$ among all $(T_1, T_2, \ldots, T_p)$-terminal cuts.

3. Divide-and-conquer algorithm. Each of the previously known deterministic algorithms reduces a minimum k-way cut problem instance to a set of minimum $(k-1)$-way cut problem instances, where the target k on the number of components is reduced only by 1. In this paper, we reduce a minimum k-way cut problem instance to a set of minimum k'-way cut problem instances with k' nearly equal to $k/2$. For this, we first observe the following property.

LEMMA 3.1. *Let $(V_1; V_2; \ldots; V_k)$ be a minimum k-way cut in a graph $G = (V, E)$, where $k \in [2, n]$.*

Then for any integer $p \in [1, k-1]$, there is a union X of p subsets in $\{V_1, V_2, \ldots, V_k\}$ such that

$$f(X) \leq \frac{2(kp - p^2)}{(k^2 - k)} opt(G, k).$$

Proof. Let $\mathcal{X}$ be the family of all such unions X. Then $|\mathcal{X}| = \binom{k}{p}$. For each edge $e = (u, v) \in (V_1; V_2; \ldots; V_k)$, there are $2\binom{k-2}{p-1}$ unions $X \in \mathcal{X}$ such that $u \in X$ and $v \in \overline{X}$ or $u \in \overline{X}$ and $v \in X$. Therefore it holds that $\sum_{X \in \mathcal{X}} f(X) = 2\binom{k-2}{p-1} opt(G, k)$, and the average of $f(X)$ over all $X \in \mathcal{X}$ is $[2\binom{k-2}{p-1} opt(G, k)]/\binom{k}{p} = 2(kp - p^2)/(k^2 - k) opt(G, k)$. This implies the lemma. □

Let $p = \lceil (k - \sqrt{k})/2 \rceil - 1$, which satisfies $2(kp - p^2)/(k^2 - k) < 1/2$ for $k \geq 5$. Then there exists a set $X \subseteq V$ such that

$$f(X) < opt(G, k)/2$$

and X is a union of p subsets in $\{V_1, V_2, \ldots, V_k\}$ for a minimum k-way cut $(V_1; V_2; \ldots; V_k)$ in G. For such a subset X, we can reduce the current instance (G, k) into two instances $(G[X], p)$ and $G([V - X], k - p)$, where a minimum k-way cut F for (G, k) is obtained from a minimum p-way cut F' for $(G[X], p)$ and a minimum $(k - p)$-way cut F'' for $(G[V - X], k - p)$ by constructing a k-way cut $F = (X; V - X) \cup F' \cup F''$. Note that the size k is reduced to at most $k - \lceil (k - \sqrt{k})/2 \rceil + 1 = \lfloor (k + \sqrt{k})/2 \rfloor + 1$, which is nearly a half of k for a large k.

Section 5 shows that the number of such subsets $X \in V$ with $f(X) < opt(G, k)/2$ is at most n^{2k-5} and a family $\mathcal{X}$ of n^{2k-5} subsets including these subsets X (possibly together with some other subsets) can be obtained in $O(n^{2k-5} F(n, m))$ time. With this property, our divide-and-conquer algorithm can be described as follows.

Algorithm MULTIWAY(G, k)
Input: A graph $G = (V, E)$ and an integer $k \in [1, |V|]$.
Output: A minimum k-way cut F in G.

```
1.  if opt(G,k) = 0 then Return F := ∅
2.  else /* opt(G,k) > 0 */
3.    if k ≤ 2 then Return a minimum k-way cut F of G in the time of O(1)
        maximum flow computations
4.    else if k ≤ 6 then Return a minimum k-way cut F of G by O(|V|^{k-1})
        maximum flow computations
5.    else /* k ≥ 7 */
6.      Compute a set X of at most |V|^{2k-5} subsets of V such that any 2-way cut
        with cost less than opt(G,k)/2 is given by (X; V−X) for some X ∈ X;
7.      p := ⌈(k − √k)/2⌉ − 1;
8.      for each X ∈ X with |X| ≥ p and |V−X| ≥ k − p do
9.        F_X := (X; V−X) ∪ MULTIWAY(G[X], p) ∪ MULTIWAY(G[V−X], k − p);
10.     end /* for */
11.     Choose a k-way cut F_X with the minimum cost over all X, and return
        F := F_X
12.   end /* if */
13. end. /* if */
```

For the correctness of algorithm MULTIWAY, we have only to give a procedure in line 5, which will be discussed in section 5. The runtime of MULTIWAY will be analyzed in section 6.

4. A crossing property. This section provides a property on crossing 2-way cuts, based on which a procedure for collecting all subsets $X \subset V$ with $f(X) < opt(G,k)/2$ is designed in section 5.

LEMMA 4.1. *For a graph $G = (V, E)$, let $\{Y_1, Y_2, \ldots, Y_q, W, Z\}$ be a partition of V, and let Q be a subset of V such that each subset in $\{Y_1 - Q, Y_2 - Q, \ldots, Y_q - Q, W \cap Q, Q \cap Z, Z - Q\}$ is nonempty. Then partition $\{Y'_i = Y_i - Q \ (i = 1, 2, \ldots, q), Y'_{q+1} = Q \cap Z, W' = (W \cup Q) - Z, Z' = Z - Q\}$ of V satisfies*

$$\begin{aligned} &2cost(Y_1; Y_2; \ldots; Y_q; W; Z) - f(Y_{1,q}) + f(Q) \\ &\quad \geq 2cost(Y'_1; Y'_2; \ldots; Y'_q; Y'_{q+1}; W'; Z') - f(Y'_{1,q+1}) + f(W \cap Q), \end{aligned}$$

where we denote $Y_{1,i} = Y_1 \cup Y_2 \cup \cdots \cup Y_i$ and $Y'_{1,j} = Y'_1 \cup Y'_2 \cup \cdots \cup Y'_j$.

Proof. We obtain

(4.1)
$$\begin{aligned} &f(Y_1) + f(Y_2) + \cdots + f(Y_q) + f(Y'_{1,q+1}) \\ &\quad \geq f(Y'_1) + f(Y'_2) + \cdots + f(Y'_q) + f(Y_{1,q} \cup (Q \cap Z)) + 2cost(Y_{1,q} \cap Q; Q \cap Z), \end{aligned}$$

by summing up the following q inequalities implied by (2.1):

$$\begin{aligned} &f(Y_1) + f((Y_{1,q} - Q) \cup (Q \cap Z)) \\ &\quad \geq f(Y_1 - Q) + f(Y_1 \cup (Y_{1,q} - Q) \cup (Q \cap Z)) + 2cost(Y_1 \cap Q; Q \cap Z), \\ &f(Y_2) + f(Y_1 \cup (Y_{1,q} - Q) \cup (Q \cap Z)) \\ &\quad \geq f(Y_2 - Q) + f(Y_{1,2} \cup (Y_{1,q} - Q) \cup (Q \cap Z)) + 2cost(Y_2 \cap Q; Q \cap Z) \\ &\qquad\qquad \cdots \\ &f(Y_q) + f(Y_{1,q-1} \cup (Y_{1,q} - Q) \cup (Q \cap Z)) \\ &\quad \geq f(Y_q - Q) + f(Y_{1,q} \cup (Y_{1,q} - Q) \cup (Q \cap Z)) + 2cost(Y_q \cap Q; Q \cap Z). \end{aligned}$$

On the other hand, (2.1) and (2.2) mean

(4.2)
$$\begin{aligned} &f(Z) + f(W) + f(Q) \geq f(Z) + f(W \cap Q) + f(W \cup Q) \\ &\quad \geq f((W \cup Q) - Z) + f(Z - Q) + 2cost(Y_{1,q} - Q; Q \cap Z) + f(W \cap Q). \end{aligned}$$

From (4.1) and (4.2), we have

$$\begin{aligned} &f(Y_1) + f(Y_2) + \cdots + f(Y_q) + f(W) + f(Z) + f(Y'_{1,q+1}) + f(Q) \\ &\geq f(Y'_1) + f(Y'_2) + \cdots + f(Y'_q) + f((W \cup Q) - Z) + f(Z - Q) \\ &\quad + f(Y_{1,q} \cup (Q \cap Z)) + 2cost(Y_{1,q} \cap Q; Q \cap Z) + 2cost(Y_{1,q} - Q; Q \cap Z) + f(W \cap \mathrm{Q}) \\ &= f(Y'_1) + f(Y'_2) + \cdots + f(Y'_q) + f(W') + f(Z') + f(Q \cap Z) + f(Y_{1,q}) + f(W \cap Q), \end{aligned}$$

implying the lemma. ∎

LEMMA 4.2. *For a graph $G = (V, E)$ and an integer $k \in [5, n-1]$, let $(X; \overline{X})$ be a 2-way cut of G, R be a nonempty subset of $\overline{X}$, and $T = \{t_1, t_2, \ldots, t_p\}$ be a set of $p \geq 2$ vertices in $\overline{X} - R$. Assume that, for each t_i, there exists a minimum $(X, T \cup R - \{t_i\})$-terminal cut $(X_i; \overline{X_i})$ which satisfies $X \cup \{t_i\} \subseteq X_i$ (see Figure 4.1). Let $\mathcal{C} = \{(X_i; \overline{X_i}) \mid 1 \leq i \leq p\}$. Then $E(\mathcal{C})$ is a $(p+2)$-way cut which partitions V into $p+2$ subsets*

$$Z = \cap_{1 \leq i \leq p} \overline{X_i}, \quad W = \cup_{1 \leq i < j \leq p} (X_i \cap X_j), \ \text{and} \ \ Y_i = X_i - W \ (i = 1, 2, \ldots, p).$$

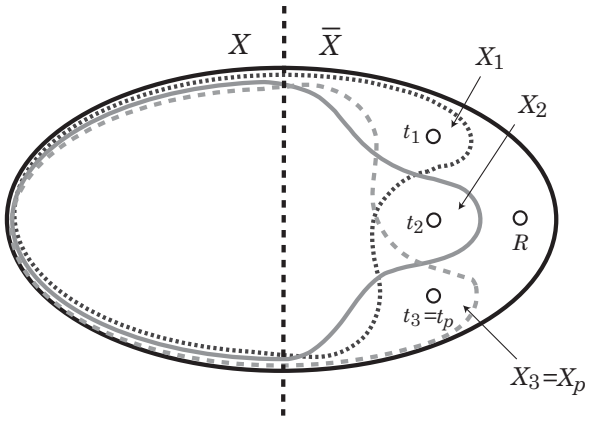


FIG. 4.1. *Illustration for a 2-way cut* $(X;\overline{X})$ *and a minimum* $(X;T\cup R-\{t_i\})$*-terminal cut* $(X_i;\overline{X_i})$*,* $i=1,2,3\ (=p)$*.*

Furthermore $(p+2)$*-way cut* $E(\mathcal{C})=(Y_1;Y_2;\ldots;Y_p;Z;W)$ *satisfies*

$$cost(E(\mathcal{C}))+cost(Z;W)+cost(Y_1;Y_2;\ldots;Y_p)\le f(X_1)+f(X_2). \tag{4.3}$$

Proof. Since $X\subseteq W$, $t_i\in Y_i$ $(1\le i\le p)$, and $R\subseteq Z$ hold, we see that $E(\mathcal{C})$ is a $(p+2)$-way cut. We prove (4.3) by an induction on p.

Basis case. For $p=2$, $Z=V-(X_1\cup X_2)$, $W=X_1\cap X_2$, $Y_1=X_1-X_2$, and $Y_2=X_2-X_1$. Then it holds that $cost(Y_1;Y_2;Z;W)+cost(Z;W)+cost(Y_1;Y_2)$ $=cost(X_1-X_2;X_2-X_1;V-(X_1\cup X_2);X_1\cap X_2)+cost(V-(X_1\cup X_2);X_1\cap X_2)$ $+cost(X_1-X_2;X_2-X_1)=f(X_1)+f(X_2)$, as required.

Inductive case. Let $q\ge 2$. Assuming that (4.3) holds for $p=q$, we prove that (4.3) holds for $p=q+1$. Let R', $T'=\{t_1,t_2,\ldots,t_{q+1}\}\subset\overline{X}-R'$ and $\mathcal{C}'$ be subsets of $\overline{X}$ and a set of $q+1$ 2-way cuts satisfying the condition of the lemma for $q+1$. We here consider $R=R'\cup\{t_{p+1}\}$, $T=T'-\{t_{q+1}\}$, and $\mathcal{C}=\mathcal{C}'-\{(X_{q+1};\overline{X_{q+1}})\}$, which satisfy the condition of the lemma for q (see Figure 4.2). Hence, by the induction

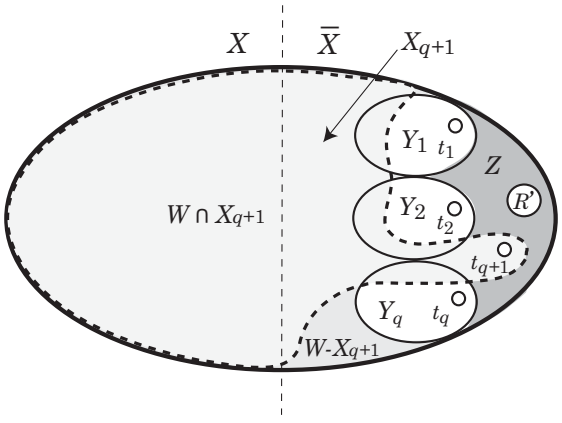


FIG. 4.2. *Illustration for a minimum* $(X,T'\cup R'-\{t_p\})$*-terminal cut* $(X_p;\overline{X_p})$ *and subsets* $Y_1,Y_2,\ldots,Y_{p-1}$*.*

hypothesis, we have

$$\begin{aligned} & f(X_1) + f(X_2) \\ & \quad \geq cost(E(\mathcal{C})) + cost(Z; W) + cost(Y_1; Y_2; \ldots; Y_q) \\ & \quad = 2cost(E(\mathcal{C})) - f(Y_{1,q}), \end{aligned} \tag{4.4}$$

where $Z = \cap_{1 \leq i \leq q} \overline{X_i}$, $W = \cup_{1 \leq i < j \leq q} (X_i \cap X_j)$, and $Y_i = X_i - W$ $(i = 1, 2, \ldots, q)$. By Lemma 4.1 to $Y_1, \ldots, Y_q$, W, Z, and $Q = X_{q+1}$, we obtain

$$\begin{aligned} 2cost(E(\mathcal{C})) - f(Y_{1,q}) &\geq 2cost(E(\mathcal{C}')) - f(Y'_{1,q+1}) + f(W \cap X_{q+1}) - f(X_{q+1}) \\ &\geq 2cost(E(\mathcal{C}')) - f(Y_{1,q+1}), \end{aligned}$$

where $f(W \cap X_{q+1}) \geq f(X_{q+1})$ holds since $(W \cap X_{q+1}; \overline{W \cap X_{q+1}})$ is an $(X, T' \cup R' - \{t_{q+1}\})$-terminal cut in G. This implies that (4.3) holds for p. ∎

5. Computing small cuts. With Lemma 4.2, we are ready to present an $O(n^{2k-5}F(n,m))$ time procedure for collecting all subsets $X \in V$ with $f(X) < opt(G,k)/2$.

THEOREM 5.1. *For a graph $G = (V, E)$ and an integer $k \in [5, n-1]$, let $(X; \overline{X})$ be a 2-way cut with $f(X) < opt(G,k)/2$. Then, for any vertices $s^* \in X$ and $t^* \in \overline{X}$, there are subsets $S \subseteq X$ and $T \subseteq \overline{X}$ with $|S| \leq k-3$ and $|T| \leq k-3$ such that $(X; \overline{X})$ is a unique minimum $(S \cup \{s^*\}, T \cup \{t^*\})$-terminal cut in G.*

Proof. Let $(X; \overline{X})$ be a 2-way cut with $f(X) \leq opt(G,k)/2$. We first prove the next claim.

CLAIM 5.2. *A set T of at most $k-3$ vertices in $\overline{X}$ can be chosen so that $(X; \overline{X})$ becomes a unique minimum $(X, T \cup \{t^*\})$-terminal cut.*

Proof. Let $\mathcal{Y}$ be the family of all subsets Y with $X \subset Y \subseteq V - \{t^*\}$ such that

$$f(Y) \leq f(X).$$

We choose a subset T of $\overline{X} - \{t^*\}$ so that T becomes a minimal transversal of $\mathcal{Y}$ (i.e., T is an inclusion-wise minimal subset of $\overline{X} - \{t^*\}$ such that $Y \cap T \neq \emptyset$ for all $Y \in \mathcal{Y}$). Since T is a transversal of $\mathcal{Y}$, no other $(X, T \cup \{t^*\})$-terminal cut than $(X; \overline{X})$ has cost less than or equal to $f(X)$. Hence, to prove the claim, it suffices to show that $|T| \leq k-3$.

For each $t \in T$, let $(X_t; \overline{X}_t)$ denote a minimum $(X, (T - \{t\}) \cup \{t^*\})$-terminal cut. We show that

$$f(X_t) \leq f(X), \quad t \in X_t.$$

By the minimality of T, each vertex $t \in T$ has a subset $Y' \in \mathcal{Y}$ such that $t \in Y'$ and $Y' \cap (T - \{t\}) \cup \{t^*\}) = \emptyset$. Hence $f(X_t) \leq f(Y') \leq f(X)$. This also implies that $X_t \in \mathcal{Y}$ and hence t must belong to X_t (since otherwise $X_t \cap T = \emptyset$ would hold). See Figure 4.1, where $R = \{t^*\}$.

The above sets $R = \{t^*\}$, T, $\mathcal{C} = \{(X_t; \overline{X}_t) \mid t \in T\}$ satisfy the condition of Lemma 4.2. By Lemma 4.2 and the assumption on $f(X)$, $E(\mathcal{C})$ is a $(|T|+2)$-way cut with $cost(E(\mathcal{C})) \leq 2\max\{f(X_t) \mid t \in T\} \leq 2f(X) < opt(G,k)$. Therefore $|T| \leq k-3$ holds, since otherwise $comp(G - E(\mathcal{C})) \geq |T| + 2 \geq k$ would hold, contradicting the definition of $opt(G,k)$. This proves the claim. ∎

By applying the above claim to X, we see that a set S of at most $k-3$ vertices in X can be chosen so that $(X; \overline{X})$ becomes a unique minimum $(S \cup \{s^*\}, \overline{X})$-terminal cut.

Finally we show that $(X;\overline{X})$ is a unique minimum $(S\cup\{s^*\},T\cup\{t^*\})$-terminal cut in G. Assume indirectly that G has another minimum $(S\cup\{s^*\},T\cup\{t^*\})$-terminal cut $(Z;\overline{Z})$. By the property of S and T, neither $Z\subseteq X$ nor $Z\supseteq X$; the remaining case is $X-Z\neq\emptyset\neq Z-X$. In this case, by the submodularity of cost function,

$$f(X)+f(Z)\geq f(X\cap Z)+f(Z\cup X)$$

holds, and we see that at least one of $(X\cap Z;\overline{X\cap Z})$ and $(Z\cup X;\overline{Z\cup X})$ is a minimum $(S\cup\{s^*\},T\cup\{t^*\})$-terminal cut. This, however, contradicts the above property of S and T. This completes the proof of the theorem. □

Based on this theorem, we can find all 2-way cuts $(X;\overline{X})$ with $f(X)<opt(G,k)/2$ by $O(n^{2k-5})$ maximum flow computations. For this, choose a vertex $s^*\in V$, and execute the following procedure for each vertex $t^*\in V-\{s^*\}$: Choose disjoint sets $S,T\subseteq V-\{s^*,t^*\}$ with $2\leq|S|\leq k-3$ and $2\leq|T|\leq k-3$, and compute a minimum $(S\cup\{s^*\},T\cup\{t^*\})$-terminal cut $(X;\overline{X})$ in G. Then the set of these 2-way cuts $(X;\overline{X})$ for all $t^*\in V-\{s^*\}$ include those with cost less than $opt(G,k)/2$. For fixed s^* and t^*, there are at most n^{2k-6} such pairs of S and T. Hence, we need $O(n^{2k-6}\cdot n)=O(n^{2k-5})$ maximum flow computations. We also have the following corollary.

COROLLARY 5.3. *Let $G=(V,E)$ be a graph, $k\in[5,n]$ be an integer, and $opt(G,k)>0$. Then the number of subsets $X\subset V$ such that $f(X)<(1/2)opt(G,k)$ is $O(n^{2k-5})$.*

6. Runtime of MULTIWAY. In this section, we analyze the runtime of algorithm MULTIWAY.

THEOREM 6.1. *For a graph $G=(V,E)$ with n vertices, m edges, and an integer $k\in[1,n]$,* MULTIWAY(G,k) *runs in $O(n^{k-1}F(n,m))$ time for $k\leq 6$ and in $O(n^{4k/(1-1.71/\sqrt{k})-34}F(n,m))$ time for $k\geq 7$, where $F(n,m)$ denotes the time complexity for computing a maximum flow in a graph with n vertices and m edges.*

Proof. We derive an upper bound $N(k,n)$ on the number of maximum flow computations to execute MULTIWAY(G,k) for a graph G with n vertices, where we assume that $N(k,n)$ is an increasing function with respect to k and n. For $k\leq 6$, it is known that a minimum k-way cut can be obtained by at most 1 (resp., $2n^2$, $4n^3$, $60n^4$, and $900n^5$) maximum flow computations for $k=2$ [12] (resp., $k=3,4,5,6$ [29, 30, 31]). For $k\geq 5$, it suffices to consider $N(k,n)$ such that

$$\begin{aligned}N(k,n)&\leq n^{2k-5}+n^{2k-5}N(k-\lceil(k-\sqrt{k})/2\rceil+1,n-\lceil(k-\sqrt{k})/2\rceil+1)\\&\leq n^{2k-5}N(\lfloor(k+\sqrt{k})/2\rfloor+1,n).\end{aligned}$$

Then we define $M(k)$ by a recursive formula $M(k)=2k-5+M(\lfloor(k+\sqrt{k})/2\rfloor+1)$ for $k\geq 7$ and $M(2)=0$, $M(3)=2$, $M(4)=3$, $M(5)=4$, and $M(6)=5$. We see that $900n^{M(k)}$ gives an upper bound on the number of maximum flow computations needed to execute MULTIWAY(G,k). We see that $M(k)\leq 4k/(1-1.71/\sqrt{k})-34$ holds for $k\leq 1.3\times 10^6$ by generating all those $M(k)$ with a computer program. We prove that $M(k)\leq 4k/(1-1.71/\sqrt{k})-34$ holds for $k>1.3\times 10^6$ with the recursive formula. Let $a=1.71$, and $k'=\lfloor(k+\sqrt{k})/2\rfloor+1\leq(k+\sqrt{k}+2)/2$. Then by the induction hypothesis we have

$$M(k)=2k-5+M(k')\leq 2k-5+4k'\sqrt{k'}/(\sqrt{k'}-a)-34.$$

Then it suffices to show that

$$4k\sqrt{k}/(\sqrt{k}-a)-34-(2k-5)-2(k+\sqrt{k}+2)\sqrt{k+\sqrt{k}+2}/(\sqrt{k+\sqrt{k}+2}-\sqrt{2}a)+34$$

is nonnegative for $k > 1.3 \times 10^6$. For this, we prove the following is nonnegative:

$$\begin{aligned} & 4k\sqrt{k}(\sqrt{k+\sqrt{k}+2}-\sqrt{2}a) - (\sqrt{k}-a)(2k-5)(\sqrt{k+\sqrt{k}+2}-\sqrt{2}a) \\ & \quad - (\sqrt{k}-a)2(k+\sqrt{k}+2)\sqrt{k+\sqrt{k}+2} \\ & = \Big((4a-2)k+(2a+1)\sqrt{k}-a\Big)\sqrt{k+\sqrt{k}+2} \\ & \quad + \sqrt{2}a(-2k\sqrt{k}-2ak-5\sqrt{k}+5a). \end{aligned} \tag{6.1}$$

Since $(4a-2)k+(2a+1)\sqrt{k}-a > 0$ for $k > 1.3 \times 10^6$, (6.1) is at least

$$\begin{aligned} & ((4a-2)k+(2a+1)\sqrt{k}-a)\sqrt{k}+\sqrt{2}a(-2k\sqrt{k}-2ak-5\sqrt{k}) \\ & \quad = \sqrt{k}\Big[((4-2\sqrt{2})a-2)k+(2a(1-\sqrt{2}a)+1)\sqrt{k}-(1+5\sqrt{2})a\Big], \end{aligned}$$

which is nonnegative, since $4a-2\sqrt{2}a-2 > 0$ holds for $k = 1.3 \times 10^6$ and in this case it holds that

$$\begin{aligned} & ((4-2\sqrt{2}) \times 1.71 - 2) \times 1.3 \times 10^6 + (2 \cdot 1.71 \times (1-\sqrt{2} \cdot 1.71)+1) \times \sqrt{1.3 \times 10^6} \\ & \qquad - (1+5\sqrt{2}) \times 1.71 > 0. \end{aligned}$$

This proves that $M(k) \le 4k/(1-1.71/\sqrt{k}) - 34$ holds for $k > 1.3 \times 10^6$. □

7. Enumerating all minimum k-way cuts. In this section, we show that algorithm MULTIWAY can be modified so that all minimum k-way cuts in G can be enumerated in nearly the same time complexity. Given a graph $G = (V, E)$ and integer k, we can construct all minimum k-way cuts F by combining a minimum p-way cut F' in $G[X]$ and a minimum $(k-p)$-way cut F'' in $G[V-X]$ for all possible subsets $X \subset V$ such that X is a union of p subsets in $\{V_1, \ldots, V_k\}$ for a minimum k-way cut in G.

However, we cannot apply Corollary 5.3 to instances (G, k) with $k \le 4$. From Lemma 3.1 with $p = 1$, we see that for any minimum k-way cut $(V_1; V_2; \ldots; V_k)$ in a graph $G = (V, E)$, there is a subset $X \in \{V_1, V_2, \ldots, V_k\}$ such that

$$f(X) \le (2/k)opt(G, k).$$

For $k \in \{2, 3, 4\}$, we derive an upper bound on the number of subsets $X \subset V$ with $f(X) \le (2/k)opt(G, k)$.

LEMMA 7.1. *Let $G = (V, E)$ be a graph, $k \in [2, 4]$ be an integer, and $opt(G, k) > 0$. Then the number of subsets $X \subset V$ such that $f(X) \le (2/k)opt(G, k)$ for $k = 2$ (resp., $k = 3, 4$) is $O(n^2)$, (resp., $O(n^4)$ and $O(n^4)$).*

Proof. Let $\mathcal{X}_k$ be the family of subsets X with $f(X) \le (2/k)opt(G, k)$. Let $\lambda(G)$ denote the edge-connectivity of G (i.e., $\lambda(G) = opt(G, 2)$). It is known [19] that, for $\lambda(G) > 0$, there are $O(n^{2\alpha})$ subsets X with $f(X) \le \alpha\lambda(G)$.

(i) Let $k = 2$. Then $\lambda(G) > 0$ holds by assumption $opt(G, 2) > 0$. This proves that $|\mathcal{X}_2| = O(n^{2\alpha}) = O(n^2)$ holds for $\alpha = 1$.

For $k \in \{3, 4\}$, we assume that $\mathcal{X}_k$ contains two subsets X_1 and X_2 such that each of $Y_1 = X_1 \cap X_2$, $Y_2 = X_1 - X_2$, $Y_3 = V - (X_1 \cup X_2)$, and $Y_4 = X_2 - X_1$ is nonempty (otherwise $|\mathcal{X}_k| \le 2n$ holds).

(ii) Let $k = 3$. If $\lambda(G) = 0$, then the edges with positive weights induce from G exactly two connected components G_1 and G_2, where we see that a minimum 3-way cut in G is a minimum 2-way cut in G_1 or G_2. By the result of (i) we have $|\mathcal{X}_k| = O(n_1^2 + n_2^2) = O(n^2)$, where n_i is the number of vertices in G_i. We next assume $\lambda(G) > 0$. Then for the above two crossing subsets $X_1, X_2 \in \mathcal{X}_3$, we have

$$\begin{aligned} 4opt(G,3) &\leq cost(Y_1; Y_2; V - (Y_1 \cup Y_2)) + cost(Y_2; Y_3; V - (Y_2 \cup Y_3)) \\ &\quad + cost(Y_3; Y_4; V - (Y_3 \cup Y_4)) + cost(Y_4; Y_1; V - (Y_4 \cup Y_1)) \\ &= 3(f(X_1) + f(X_2)) - 2cost(Y_1; Y_3) - 2cost(Y_2; Y_4) \\ &\leq 3((2/3)opt(G,3) + (2/3)opt(G,3)) - 2cost(Y_1; Y_3) - 2cost(Y_2; Y_4) \\ &= 4opt(G,3) - 2cost(Y_1; Y_3) - 2cost(Y_2; Y_4), \end{aligned}$$

implying that $f(X_1) = f(X_2) = (2/3)opt(G,3)$ and $cost(Y_1; Y_3) = cost(Y_2; Y_4) = 0$. Hence no two subsets $X, X' \in \mathcal{X}_3$ with $f(X) < (2/3)opt(G,3)$ and $f(X') < (2/3)opt(G,3)$ cross each other, indicating that the number of subsets X with $f(X) < (2/3)opt(G,3)$ is $O(n)$. If $\lambda(G) < (1/3)opt(G,3)$, then there is a subset $Z \subset V$ with $f(Z) = \lambda(G) < (1/3)opt(G,3) (< f(X_1))$, where $Z \neq X_1 \neq V - Z$ holds, and $F = (Z; V-Z) \cup (X_1; V-X_1)$ is a 3-way cut with $cost(F) < (1/3)opt(G,3) + (2/3)opt(G,3) = opt(G,3)$, which is a contradiction. Therefore, $\lambda(G) \geq (1/3)opt(G,3)$ holds, and $f(X) = (2/3)opt(G,3) \leq 2\lambda(G)$ holds for every subset X with $f(X) = (2/3)opt(G,3)$, implying that $|\mathcal{X}_3| = O(n^{2\alpha}) = O(n^4)$ holds for $\alpha = 2$.

(iii) Let $k = 4$. If $\lambda(G) = 0$, then it is not difficult to see that $|\mathcal{X}_3| = O(n^4)$ holds by using the results in (i)–(ii). Assume $\lambda(G) > 0$. For the above two crossing subsets $X_1, X_2 \in \mathcal{X}_4$, we have

$$\begin{aligned} opt(G,4) &\leq cost(Y_1; Y_2; Y_3; Y_4) \\ &= f(X_1) + f(X_2) - cost(Y_1; Y_3) - cost(Y_2; Y_4) \\ &\leq (1/2)opt(G,4) + (1/2)opt(G,4) - cost(Y_1; Y_3) - cost(Y_2; Y_4) \\ &= opt(G,4) - cost(Y_1; Y_3) - cost(Y_2; Y_4), \end{aligned}$$

implying that $f(X_1) = f(X_2) = (1/2)opt(G,4)$ and $cost(Y_1; Y_3) = cost(Y_2; Y_4) = 0$. From this, the number of subsets X with $f(X) < (1/2)opt(G,4)$ is $O(n)$. Let $\mathcal{X}_4' = \{X \in \mathcal{X}_4 \mid f(X) = (1/2)opt(G,4)\}$. If $\lambda(G) \geq (1/4)opt(G,4)$, then $f(X) = (1/2)opt(G,4) \leq 2\lambda(G)$ holds for every subset X with $f(X) = (1/2)opt(G,4)$, implying that $|\mathcal{X}_4'| = O(n^{2\alpha}) = O(n^4)$ holds for $\alpha = 2$. Assume $\lambda(G) < (1/4)opt(G,4)$. Let Z be a subset of V with $f(Z) = \lambda(G)$. Consider two crossing subsets $X_1, X_2 \in \mathcal{X}_4'$ with $Z \cap X_1 \neq \emptyset \neq Z \cap X_2$. As observed above, we have $f(X_1) = f(X_2) = (1/2)opt(G,4)$ and $cost(Y_1; Y_3) = cost(Y_2; Y_4) = 0$, indicating that one of the 3-way cuts $F_1 = (Y_1; Y_2; Y_3 \cup Y_4)$, $F_2 = (Y_1; Y_2 \cup Y_3; Y_4)$, $F_3 = (Y_1 \cup Y_2; Y_3; Y_4)$, and $F_4 = (Y_1 \cup Y_4; Y_2; Y_3)$ has cost at most $(3/4)opt(G,4)$. Such a 3-way cut F_i and $(Z; V-X)$ have $cost(F_i) + cost(Z; V-X) \leq (3/4)opt(G,4) + \lambda(G) < (3/4)opt(G,4) + (1/4)opt(G,4) = opt(G,4)$, and this means that $F_i \cup (Z; V-X)$ remains a 3-way cut, i.e., $X_1 \cap X_2 = Z$ (otherwise $F_i \cup (Z; V-X)$ would be a 4-way cut with cost less than $opt(G,4)$). Hence we have a property that, for two crossing subsets $X, X' \in \mathcal{X}_4'$ with $Z \cap X \neq \emptyset \neq Z \cap X'$, $X - Z$ and $X' - Z$ are disjoint, and we see that there are $O(n)$ subsets $X \in \mathcal{X}_4'$ with $Z \cap X \neq \emptyset$. Since there are $O(n^2)$ subsets Z with $f(Z) = \lambda(G)$, we have $|\mathcal{X}_4'| = O(n^2 \cdot n) = O(n^3)$. This completes the proof of the lemma. □

It is known that the h minimum 2-way cuts can be enumerated in $O(hnF(n,m))$ time [10, 11, 36]. Thus, for $k = 2, 3, 4$ (resp., $k \geq 5$), all subsets X with $f(X) \leq$

$(2/k)opt(G,k)$ (resp., $f(X) < (1/2)opt(G,k)$) can be obtained in $O(n^3F(n,m))$ time for $k=2$, in $O(n^5F(n,m))$ time for $k=3,4$ (by Lemma 7.1), and in $O(n^{2k-5}F(n,m))$ time for $k \geq 5$ (by Theorem 5.1).

We are ready to describe our algorithm for enumerating all minimum k-way cuts.

Algorithm ALL_CUTS(G,k)
Input: A graph $G=(V,E)$ and an integer $k \in [1,|V|]$.
Output: The set $\mathcal{F}$ of all minimum k-way cuts in G.

1. **if** $opt(G,k)=0$ **then** Return $\mathcal{F} := \emptyset$
2. **else** /* $opt(G,k) > 0$ */
3. **if** $k=2$ **then** Return $\{(X; V-X) \mid f(X) = \lambda(G)\}$
4. **else if** $k \in \{3,4\}$ **then**
   Compute the set $\mathcal{X}_k$ of all subsets $X \subset V$ such that $f(X) \leq (2/k)opt(G,k)$ and $|V-X| \geq k-1$;
5. $\mathcal{F} := \{(X; V-X) \cup F_2 \mid X \in \mathcal{X}_k,\ F_2 \in \text{ALL\_CUTS}(G[V-X], k-1)\}$;
6. Return $\mathcal{F} := \mathcal{F} - \{F \in \mathcal{F} \mid cost(F) > opt(G,k)\}$ /* $\min_{F\in\mathcal{F}} cost(F) = opt(G,k)$ */
7. **else** /* $k \geq 5$ */
8. Compute the set $\mathcal{X}_k$ of all subsets $X \subset V$ such that $f(X) < (1/2)opt(G,k)$;
9. $p := \lceil (k-\sqrt{k})/2 \rceil - 1$;
10. $\mathcal{F} := \bigcup_{X\in\mathcal{X}_k : |X|\geq p,\ |V-X|\geq k-p} \{F_1 \cup F_2 \mid F_1 \in \text{ALL\_CUTS}(G[X], p),\ F_2 \in \text{ALL\_CUTS}(G[V-X], k-p)\}$;
11. Return $\mathcal{F} := \mathcal{F} - \{F \in \mathcal{F} \mid cost(F) > opt(G,k)\}$ /* $\min_{F\in\mathcal{F}} cost(F) = opt(G,k)$ */
12. **end** /* if */
13. **end** /* if */
14. **end**. /* if */

We have the same recursive formula on the number of instances generated during the execution of ALL_CUTS, where the difference from the analysis for the runtime of MULTIWAY is some of initial terms in $M(k)$. By checking a solution to the formula up to 1.3×10^6 by a computer program, we see that the total number of instances is $O(n^{4k/(1-1.71/\sqrt{k})-20})$. Since each subset $X \in \mathcal{X}_k$ is generated in $O(nF(n,m)) = O(n^4)$ time, we establish the next result.

THEOREM 7.2. *For a graph $G=(V,E)$ with n vertices m edges and an integer $k \in [3,n]$,* ALL_CUTS(G,k) *finds all minimum k-way cuts in $O(n^{4k/(1-1.71/\sqrt{k})-19}F(n,m))$ time, where $F(n,m)$ denotes the time complexity for computing a maximum flow in a graph with n vertices and m edges.*

8. Concluding remarks. In this paper, we have shown that, for general k, all minimum k-way cuts can be computed in $O(n^{4k/(1-1.71/\sqrt{k})-16})$ time. This is the first deterministic algorithm with time complexity whose exponent is $O(k)$ for a general graph. The new time bound improves the previous time bound $O(n^{k^2/2-3k/2+4}F(n,m))$ for deterministic algorithms but is still higher than the bound $O(n^{2(k-1)}\log^3 n)$ of a Monte Carlo algorithm due to Karger and Stein [23]. So, it is left open to derive a better upper bound on the number of subsets X with $f(X) < opt(G,k)/2$. The key property for our algorithm is Theorem 5.1. A similar property is found in the article by Goldschmidt and Hochbaum [8], but we have a simpler proof for this property under a less constrained setting, which allows us to apply Lemma 3.1 to generate

instances with nearly halved k. However, based on Theorem 5.1, we can find *all* minimum k-way cuts, requiring a high time complexity. Another future work is to find a more restricted characterization for a family $\mathcal{X}$ of subsets X from which we can construct *at least one* minimum k-way cut.

Acknowledgments. The authors would like to thank the referees for their thorough reading of this article, which led to many useful suggestions for improving presentation, and the editor for the effort of coordinating.

REFERENCES

[1] C. J. Alpert and A. B. Kahng, *Recent directions in netlist partitioning: A survey*, Integration, The VLSI Journal, 19 (1995), pp. 1–81.
[2] D. Bertsimas, C. P. Teo, and R. Vohra, *Analysis of LP relaxations for multiway and multicut problems*, Networks, 34 (1999), pp. 102–114.
[3] C. J. Colbourn, *The Combinatorics of Network Reliability*, Oxford University Press, Oxford, UK, 1987.
[4] W. H. Cunningham, *Optimal attack and reinforcement of a network*, J. ACM, 32 (1985), pp. 549–561.
[5] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis, *The complexity of multiterminal cuts*, SIAM J. Comput., 23 (1994), pp. 864–894.
[6] W. E. Donath, *Logic partitioning*, in Physical Design Automation of VLSI Systems, B. T. Preas and M. J. Lorenzetti, eds., Benjamin Cummings, Menlo Park, CA, 1988, pp. 65–86.
[7] A. V. Goldberg and R. E. Tarjan, *A new approach to the maximum-flow problem*, J. ACM, 35 (1998), pp. 921–940.
[8] O. Goldschmidt and D. S. Hochbaum, *Polynomial algorithm for the k-cut problem for fixed k*, Math. Oper. Res., 19 (1994), pp. 24–37.
[9] D. Gusfield, *Connectivity and edge-disjoint spanning trees*, Inform. Process. Lett., 16 (1983), pp. 87–89.
[10] H. W. Hamacher, J.-C. Picard, and M. Queyranne, *Ranking the cuts and cut-sets of a network*, in Algebraic and Combinatorial Methods in Operations Research, North–Holland Math. Stud. 95, North–Holland, Amsterdam, 1984, pp. 183–200.
[11] H. W. Hamacher, J.-C. Picard, and M. Queyranne, *On finding the K best cuts in a network*, Oper. Res. Lett., 2 (1984), pp. 303–305.
[12] J. Hao and J. Orlin, *A faster algorithm for finding the minimum cut in a directed graph*, J. Algorithms, 17 (1994), pp. 424–446.
[13] D. Hartvigsen, *Minimum path basis*, J. Algorithms, 15 (1993), pp. 125–142.
[14] D. Hartvigsen, *The planar multiterminal cut problem*, Discrete Appl. Math., 85 (1998), pp. 203–222.
[15] X. He, *An improved algorithm for the planar 3-cut problem*, J. Algorithms, 12 (1991), pp. 23–37.
[16] S. Kapoor, *On minimum 3-cuts and approximating k-cuts using cut trees*, in Integer Programming and Combinatorial Optimization, Lecture Notes in Comput. Sci. 1084, Springer-Verlag, Berlin, 1996, pp. 132–146.
[17] Y. Kamidoi, S. Wakabayashi, and N. Yoshida, *Faster algorithms for finding a minimum k-way cut in a weighted graph*, in Proceedings of the IEEE International Symposium on Circuits and Systems, IEEE Circuits and Systems Society, Piscataway, NJ, 1997, pp. 1009–1012.
[18] Y. Kamidoi, S. Wakabayashi, and N. Yoshida, *A divide-and-conquer approach to the minimum k-way cut problem*, Algorithmica, 32 (2002), pp. 262–276.
[19] D. R. Karger, *Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm*, in Proceedings of the ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1993, pp. 21–30.
[20] D. R. Karger, *Minimum cuts in near-linear time*, in Proceedings of the 28th ACM Symposium on Theory of Computing, ACM, New York, 1996, pp. 56–63.
[21] D. R. Karger, P. Klein, C. Stein, M. Thorup, and N. Young, *Rounding algorithms for a geometric embedding of minimum multiway cut*, in Proceedings of the 31st ACM Symposium on Theory of Computing, ACM, New York, 1999, pp. 668–678.
[22] D. R. Karger and C. Stein, *An $\tilde{O}(n^2)$ algorithm for minimum cuts*, in Proceedings of the 25th ACM Symposium on Theory of Computing, ACM, New York, 1993, pp. 757–765.

[23] D. R. Karger and C. Stein, *A new approach to the minimum cut problems*, J. ACM, 43 (1996), pp. 601–640.
[24] A. V. Karzanov, *Determining the maximal flow in a network by the method of preflows*, Soviet Math. Dokl., 15 (1974), pp. 434–437.
[25] C. H. Lee, M. Kim, and C. I. Park, *An efficient k-way graph partitioning algorithm for task allocation in parallel computing systems*, in Proceedings of the IEEE International Conference on Computer-Aided Design, IEEE Computer Society, Los Alamitos, CA, 1990, pp. 748–751.
[26] T. Lengaur, *Combinatorial Algorithms for Integrated Circuit Layout*, Wiley, New York, 1990.
[27] M. S. Levine, *Faster randomized algorithms for computing minimum* $\{3, 4, 5, 6\}$*-way cuts*, in Proceedings of the ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2000, pp. 735–742.
[28] H. Nagamochi and T. Ibaraki, *Computing the edge-connectivity in multigraphs and capacitated graphs*, SIAM J. Discrete Math., 5 (1992), pp. 54–66.
[29] H. Nagamochi and T. Ibaraki, *A fast algorithm for computing minimum* 3*-way and* 4*-way cuts*, in Integer Programming and Combinatorial Optimization, Lecture Notes in Comput. Sci. 1610, Springer-Verlag, 1999, pp. 377–390.
[30] H. Nagamochi and T. Ibaraki, *A fast algorithm for computing minimum* 3*-way and* 4*-way cuts*, Math. Program., 88 (2000), pp. 507–520.
[31] H. Nagamochi, S. Katayama, and T. Ibaraki, *A faster algorithm for computing minimum* 5*-way and* 6*-way cuts in graphs*, J. Comb. Optim., 4 (2000), pp. 151–169.
[32] J. Naor and Y. Rabani, *Tree packing and approximating k-cuts*, in Proceedings of the ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2001, pp. 26–27.
[33] H. Saran and V. V. Vazirani, *Finding k cuts within twice the optimal*, SIAM J. Comput., 24 (1995), pp. 101–108.
[34] H. S. Stone, *Multiprocessor scheduling with the aid of network flow algorithms*, IEEE Trans. Software Engrg., 3 (1977), pp. 85–93.
[35] P. Tittmann, *Partitions and network reliability*, Discrete Appl. Math., 95 (1999) pp. 445–453.
[36] V. Vazirani and M. Yannakakis, *Suboptimal cuts: Their enumeration, weight, and number*, in Algebraic and Logic Programming, Lecture Notes in Comput. Sci. 632, Springer-Verlag, Berlin, 1992, pp. 366–377.
[37] W. C. Yeh, *A simple algorithm for the planar multiway put problem*, J. Algorithms, 39 (2001), pp. 68–77.
[38] L. Zhao, H. Nagamochi, and T. Ibaraki, *Approximating the minimum k-way cut in a graph via minimum* 3*-way cuts*, J. Comb. Optim., 5 (2001), pp. 397–410.
[39] L. Zhao, H. Nagamochi, and T. Ibaraki, *On generalized greedy splitting algorithms for multiway partition problems*, Discrete Appl. Math., 143 (2004), pp. 130–143.
[40] L. Zhao, H. Nagamochi, and T. Ibaraki, *Greedy splitting algorithms for approximating multiway partition problems*, Math. Program., 102 (2005), pp. 167–183.

SIAM J. COMPUT.
Vol. 36, No. 5, pp. 1342–1359

ONLINE CONFLICT-FREE COLORING FOR INTERVALS*

KE CHEN†, AMOS FIAT‡, HAIM KAPLAN‡, MEITAL LEVY‡, JIŘÍ MATOUŠEK§, ELCHANAN MOSSEL¶, JÁNOS PACH‖, MICHA SHARIR#, SHAKHAR SMORODINSKY††, ULI WAGNER‡‡, AND EMO WELZL††

Abstract. We consider an online version of the conflict-free coloring of a set of points on the line, where each newly inserted point must be assigned a color upon insertion, and at all times the coloring has to be *conflict-free*, in the sense that in every interval I there is a color that appears exactly once in I. We present deterministic and randomized algorithms for achieving this goal, and analyze their performance, that is, the maximum number of colors that they need to use, as a function of the number n of inserted points. We first show that a natural and simple (deterministic) approach may perform rather poorly, requiring $\Omega(\sqrt{n})$ colors in the worst case. We then derive two efficient variants of this simple algorithm. The first is deterministic and uses $O(\log^2 n)$ colors, and the second is randomized and uses $O(\log n)$ colors with high probability. We also show that the $O(\log^2 n)$ bound on the number of colors used by our deterministic algorithm is tight on the worst case. We also analyze the performance of the simplest proposed algorithm when the points are inserted in a random order and present an incomplete analysis that indicates that, with high probability, it uses only $O(\log n)$ colors. Finally, we show that in the extension of this problem to two dimensions, where the relevant ranges are disks, n colors may be required in the worst case.

Key words. conflict-free coloring, online algorithms, randomized algorithms, branching processes

AMS subject classifications. 05C15, 52C45, 68Q25, 68W20, 68W40

DOI. 10.1137/S0097539704446682

1. Introduction.

Let P be a set of n points in $\mathbb{R}^d$ and $\mathcal{R}$ a set of subsets of $\mathbb{R}^d$, called *ranges* (e.g., the set of all disks in the plane). A coloring of P is called

*Received by the editors December 14, 2004; accepted for publication (in revised form) June 23, 2006; published electronically December 26, 2006. Part of the work on the paper was carried out at MSRI, Berkeley, when several of the authors visited this institute during the fall of 2003. This paper combines and extends results from [5, 11].

http://www.siam.org/journals/sicomp/36-5/44668.html

†Department of Computer Science, University of Illinois, 201 N. Goodwin Ave., Urbana, IL 61801 (kechen@uiuc.edu). Work by this author was partially supported by NSF award CCR-0132901.

‡School of Computer Science, Tel Aviv University, Tel Aviv, Israel (fiat@post.tau.ac.il, haimk@post.tau.ac.il, levymeit@post.tau.ac.il). Work by the third author was partially supported by German Israeli Foundation (GIF) grant 2051-1156-6/2002.

§Department of Applied Mathematics and Institute for Theoretical Computer Science (ITI), Charles University, Prague, Czech Republic (matousek@kam.mff.cuni.cz).

¶Department of Statistics, University of California at Berkeley, Berkeley, CA 94720 (mossel@stat.berkeley.edu). Work by this author was supported by a Miller Fellowship in Statistics and Computer Science, University of California at Berkeley.

‖Courant Institute of Mathematical Sciences, New York University, New York, NY 10012 (pach@cims.nyu.edu).

#School of Computer Science, Tel Aviv University, Tel Aviv, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012 (michas@post.tau.ac.il). Work by this author was supported by a grant from the U.S.–Israeli Binational Science Foundation, by a grant from the Israel Science Fund (for a Center of Excellence in Geometric Computing), by NSF grants CCR-97-32101 and CCR-00-98246, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University. Part of the work was carried out during a visit to Charles University, which was supported by COMBSTRU.

††Institute for Theoretical Computer Science, ETH Zürich, Zürich, Switzerland (sshakhar@inf.ethz.ch, emo@inf.ethz.ch).

‡‡Department of Applied Mathematics, Charles University, Prague, Czech Republic (uli@kam.mff.cuni.cz). Work by this author was supported by COMBSTRU.

conflict-free (CF) with respect to $\mathcal{R}$ if for each $r \in \mathcal{R}$ with $P \cap r \neq \emptyset$, there is at least one color that appears exactly once in r.

We consider the following dynamic scenario of CF coloring of points on the line, with respect to interval ranges. We maintain a finite set $P \subset \mathbb{R}$. Initially, P is empty, and we repeatedly insert points into P, one point at a time. We denote by $P(t)$ the set P after the tth point has been inserted. Each time we insert a point p, we need to assign a color $c(p)$ to it, which is a positive integer. Once the color has been assigned to p, it cannot be changed in the future. The coloring should remain CF at all times. That is, as in the static case, for any interval I that contains points of $P(t)$, there is a color that appears exactly once in I.

The static version of CF coloring has been studied recently in several papers [9, 10, 12, 14, 15] in considerably more general settings, involving point sets in higher dimensions and ranges that are disks, balls, axis-parallel boxes, or more general ranges that satisfy certain geometric conditions. The study of this problem is motivated by the problem of frequency-assignment in cellular networks. Specifically, cellular networks are heterogeneous networks with two different types of nodes: *base stations* (that act as servers) and *clients*. The base stations are interconnected by an external fixed backbone network. Clients are connected only to base stations; connections between clients and base stations are implemented by radio links. Fixed frequencies are assigned to base stations to enable links to clients. Clients, on the other hand, continuously scan frequencies in search of a base station with good reception. The fundamental problem of frequency-assignment in cellular networks is to assign frequencies to base stations so that every client, located within the receiving range of at least one station, can be served by some base station, in the sense that the client is located within the range of the station and no other station within its reception range has the same frequency (such a station would be in "conflict" with the given station due to mutual interference). The goal is to minimize the number of assigned frequencies ("colors") since the frequency spectrum is limited and costly.

Suppose we are given a set of n base stations, also referred to as *antennae*. Assume, for simplicity, that the area covered by a single antenna is given as a disk in the plane. Namely, the location of each antenna (base station) and its radius of transmission are fixed and given (the transmission radii of the antennae are not necessarily equal). Even et al. [10] have shown that one can find an assignment of frequencies to the antennae with a total of at most $O(\log n)$ frequencies such that each antenna is assigned one of the frequencies and the resulting assignment is free of conflicts, in the preceding sense. Furthermore, it has been shown that this bound is worst-case optimal [10, 13, 14]. When the given antennae all have the same radius of transmission (say, unit radius), the problem is easily seen to be equivalent to that of coloring n points in the plane such that for any unit radius disk that contains more than one of the given points, at least one of the colors in that disk is unique. This is the scenario whose online version is studied in this paper. We do not address the dual version, in which the goal is to color n given ranges so that, for each point p that lies in their union, there is a color that appears exactly once among the ranges that contain p. See [10, 12, 14] for many variants of both (static) versions of the problem.

To capture a dynamic scenario where antennae can be added to the network, we introduce and study an online version of the CF coloring problem, as described above. As we show in this paper, the online version of the problem is considerably harder, even in the one-dimensional case, where the static version is trivial and fully understood. We begin by proposing a natural, simple, and obvious coloring algorithm (which we call the UniMax greedy algorithm), but show that in the worst case it

has poor performance. Specifically, the UniMax greedy algorithm may require $\Omega(\sqrt{n})$ colors in the worst case. We still do not have any nontrivial (i.e., sublinear) upper bound on the performance of the algorithm.

The UniMax greedy algorithm is indeed greedy in nature, but there are several different greedy approaches, and we briefly discuss another greedy alternative, about which almost nothing is known.

We next remedy the situation by presenting two more efficient algorithms. We describe a 2-stage deterministic variant of the UniMax greedy algorithm and show that the maximum number of colors that it uses is $\Theta(\log^2 n)$. We also describe a randomized version of the UniMax greedy algorithm which uses, with high probability,[1] only $O(\log n)$ colors.

The best known general lower bound for this problem is $\Omega(\log n)$, which holds also for the static case (see [10, 13, 14]), so there still remains a gap between the upper and lower bounds in the deterministic case.

Our randomized algorithm works against an oblivious adversary. That is, we assume that the sequence of points is fixed by the adversary before starting to feed them one by one to the online algorithm. The adversary cannot choose its next point based on the actions or the random choices that the online algorithm has made so far. The coloring that our algorithm produces is CF no matter which random choices it makes. For further discussion on different kinds of adversaries in online computations, see [4] for the general case and [2] for the specific case of online CF coloring.

Next, we return to the UniMax greedy algorithm, which can be inefficient in the worst case, and analyze its performance when the points are inserted in a *random order*. We reduce the problem to a certain stationary stochastic process, and present partial analysis of its performance, as well as a fairly reasonable set of conjectures, strongly supported by simulations, that indicate that the expected number of colors that the simple algorithm uses in this case is only $O(\log n)$.

Finally, we consider the extension of the online version to point sets in the plane. Unfortunately, we show that, in the simple case where the ranges that are required to be CF are disks (or arbitrary radii), n colors may be needed in the worst case. Nevertheless, (much) better solutions might still exist for random distributions of the points, for other ranges, or for relaxed versions of the problem, in which each range has a color that appears in it at least once and at most k times for some constant k [14]. A recent follow-up study by Chen, Kaplan, and Sharir [6] (see also [5]) gives randomized online CF coloring algorithms for points in the plane, with respect to half-planes, unit disks, or nearly equal axis-parallel rectangles. The algorithms use $O(\log n)$ colors with high probability. An even more recent result of Bar-Noy, Cheilaris, and Smorodinsky [3] provides a general randomized online CF coloring algorithm, which achieves the same performance as [6] in the special cases just mentioned.

There are many open problems that our study raises: Obtain, if possible, an improved algorithm-independent deterministic lower bound for online CF coloring for intervals; get a better understanding of the problem behavior in the plane and in higher dimensions; design and analyze other strategies, and so on (see additional problems posed throughout the paper). We note that CF coloring is closely related to the problem of *vertex ranking* in graphs (see, e.g., [8]). Some of our algorithms, which maintain the property that the *maximum* color in any interval is unique, actually per-

[1] This means that the probability of failure is at most $1/p(n)$, where $p(n)$ is polynomial in n, whose degree can be made arbitrarily large by adjusting the constants of proportionality in the performance bound.

form online vertex ranking in *paths*. Extending our analysis to online vertex ranking in other kinds of graphs (trees, for example) raises yet another set of interesting open problems.

2. The UniMax greedy coloring algorithm. Instead of the usual CF property, we wish to maintain the following stronger *unique maximum invariant* (in which we assume that the colors are positive integers):

> *At any given step t and for any interval I that contains points of $P(t)$, there is only one element of $P(t) \cap I$ that attains the* maximum *color in that set.*

This invariant implies that the coloring of $P(t)$ is CF at any time t. It is indeed a stronger condition: CF coloring requires only that for each interval there exists a color (not necessarily the maximum) that appears there only once.

We employ the following simple-minded algorithm for inserting a point p into the current set $P(t)$. In a nutshell, the rule is simply to assign to p the *smallest* possible color that maintains the invariant. This rule is implemented as follows. We say that the newly inserted point p *sees* a point x if all the colors of the points between p and x (exclusive) are smaller than $c(x)$. In this case we also say that p sees the color $c(x)$. We then give p the smallest color that it does not see. (Note that a color can be seen from p either to the left or to the right, but not in both directions; see below.) We refer to this algorithm as the *Unique Maximum Greedy* algorithm, or the UniMax greedy algorithm, for short.

Below is an illustration of the coloring rule of the UniMax greedy algorithm. The left column gives the colors (integers in the range $1, 2, \ldots, 6$) assigned to the points in the current set P and the location of the next point to be inserted (indicated by a period). The right column gives the colors "seen" by the new point. The colors seen to the left precede the $\cdot$, and those seen to the right succeed the $\cdot$.

$1\cdot$	$[1\cdot]$
$1 \cdot 2$	$[1 \cdot 2]$
$1 \cdot 32$	$[1 \cdot 3]$
$12 \cdot 32$	$[2 \cdot 3]$
$121 \cdot 32$	$[21 \cdot 3]$
$121 \cdot 432$	$[21 \cdot 4]$
$121 \cdot 3432$	$[21 \cdot 34]$
$1215 \cdot 3432$	$[5 \cdot 34]$
$1215 \cdot 13432$	$[5 \cdot 134]$
$12152 \cdot 13432$	$[52 \cdot 134]$
$121526 \cdot 13432$	$[6 \cdot 134]$

Correctness. The correctness of the algorithm is established by induction on the insertion order. First, note that no color can be seen twice from p: This is obvious for two points that lie both to the left or both to the right of p. If p sees the same color at a point u to its left and at a point v to its right, then the interval $[u, v]$, before p is inserted, does not have a unique maximum color; thus this case is impossible, too. Next, if p is assigned color c, any interval that contains p still has a unique maximum color: This follows by induction when the maximum color is greater than c. If the maximum color is c, then it cannot be shared by another point u in the interval, because then p would have seen the nearest such point and thus would not be assigned color c. It is also easy to see that the algorithm assigns to each newly inserted point the smallest possible color that maintains the invariant of a unique

maximum color in each interval. This makes the algorithm *greedy* with respect to the unique maximum condition.

Special insertion orders. Denote by $C(P(t))$ the sequence of colors assigned to the points of $P(t)$, in left-to-right order along the line. Let $c_{\max}(P(t))$ denote the maximum color in $C(P(t))$.

The *complete binary tree sequence* S_k of order k is defined recursively as $S_1 = (1)$ and $S_k = S_{k-1} \| (k) \| S_{k-1}$, for $k > 1$, where $\|$ denotes concatenation. Clearly, $|S_k| = 2^k - 1$.

For each pair of integers $a < b$, denote by $C_0(a, b)$ the following special sequence. Let k be the integer satisfying $2^{k-1} \le b < 2^k$. Then $C_0(a, b)$ is the subsequence of S_k from the ath place to the bth place (inclusive). For example, $C_0(5, 12)$ is the subsequence $(1, 2, 1, 4, 1, 2, 1, 3)$ of $(1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1)$.

LEMMA 2.1. (a) *If each point is inserted into P to the right of all preceding points, then $C(P(t)) = C_0(1, t)$.*

(b) *If each point is inserted into P to the left of all preceding points, then $C(P(t)) = C_0(2^k - t, 2^k - 1)$, where k satisfies $2^{k-1} \le t < 2^k$.*

(c) *If each point is inserted into P either from the left or from the right, then $C(P(t))$ is some subsequence of the form $C_0(a, b)$, where $b \le |P(t)|$.*

Proof. The proof is easy and therefore omitted. □

2.1. Lower bound for the UniMax greedy algorithm.

THEOREM 2.2. *The UniMax greedy algorithm may require $\Omega(\sqrt{n})$ colors in the worst case for a set of n points.*

Proof. For each integer k, define the sequence

$$C_k = (1, 2, 1, 3, 2, 1, \ldots, k-1, k-2, \ldots, 1, k, k-1, \ldots, 1).$$

Note that C_k is the concatenation of k sequences $D_1 \| D_2 \| \cdots \| D_k$, where $D_j = (j, j-1, \ldots, 2, 1)$. Put $n_k = k(k+1)/2$. We prove the following property, from which the assertion of the theorem is an immediate corollary.

$(*)$ There exists an insertion order of n_k points for which the color sequence produced by the UniMax greedy algorithm is C_k.

The proof proceeds by induction on k. We note that the claim easily holds for $k = 1, 2$. Suppose that there is an insertion sequence S_k for which the UniMax greedy algorithm produces the color sequence C_k. We insert the next point between D_{k-1} and D_k and observe that it is assigned color $k+1$. We then insert a point between D_{k-2} and D_{k-1}, which is assigned color k. Proceeding in this manner from right to left, we insert k points between consecutive subsequences D_{j-1}, D_j. The color sequence now becomes

$$D_2 \| D_3 \| D_4 \| \cdots \| D_k \| D_{k+1}.$$

To complete the step, we insert one additional point to the left of the whole sequence, which gets the color 1, thereby producing the color sequence C_{k+1}. This completes the proof of $(*)$ and thus of the theorem. □

Open problem. Obtain an upper bound for the maximum number of colors that the algorithm uses for n inserted points. We conjecture that the bound is close to the $\Omega(\sqrt{n})$ lower bound. At the moment, we do not have any sublinear upper bound.

2.2. Related algorithms.

The First-Fit algorithm—another greedy strategy. The UniMax greedy algorithm is greedy for maintaining the unique maximum invariant, namely, that in each interval

the *maximum color* appears exactly once. Perhaps it is more natural to consider a greedy approach in which we want only to enforce the standard CF property. That is, we want to assign to each newly inserted point the *smallest* color for which the CF property continues to hold. There are cases where this *First-Fit* greedy algorithm uses fewer colors than the UniMax greedy algorithm: Consider an insertion of five points in the order (1 3 2 4 5). The UniMax greedy algorithm produces the color sequence (1 3 2 1 4), whereas the First-Fit algorithm produces the coloring (1 3 2 1 2).

Very recently, after the original submission of this paper, Bar-Noy, Cheilaris, and Smorodinsky [2] have shown that in the worst case the First-Fit algorithm uses about $n/2$ colors. More precisely, there are sequences with $2i + 3$ elements that force the algorithm to use $i + 3$ colors, and this bound is tight.

CF coloring for unit intervals. Consider the special case where we want the CF property to hold only for *unit intervals*. In this case, $O(\log n)$ colors suffice: Partition the line into the unit intervals $J_i = [i, i + 1)$ for $i \in \mathbb{Z}$. Color the intervals J_i with even i as white, and those with odd i as black. Note that any unit interval meets only one white and one black interval. We color the points in each J_i independently, using the same set of "light colors" for each white interval and the same set of "dark colors" for each black interval. For each J_i, we color the points that it contains using the UniMax greedy algorithm, except that new points inserted into J_i between two previously inserted points get a special color, color 0. It is easily checked that the resulting coloring is CF with respect to unit intervals. Since we effectively insert points into any J_i only to the left or to the right of the previously inserted points, Lemma 2.1(c) implies that the algorithm uses only $O(\log n)$ (light and dark) colors. We remark that this algorithm satisfies the unique maximum color property for unit-length intervals.

We note that, in contrast to the static case (which can always be solved with $O(1)$ colors), $\Omega(\log n)$ colors may be needed in the worst case. Indeed, consider a left-to-right insertion of n points into a sufficiently small interval. Each contiguous subsequence σ of the points will be a suffix of the whole sequence at the time the rightmost element of σ is inserted. Since such a suffix can be cut off the current set by a unit interval, it must have a unique color. Hence, at the end of insertion, *every* subsequence must have a unique color, which implies (see [10, 14]) that $\Omega(\log n)$ colors are needed.

3. An efficient deterministic algorithm. In this section we modify the UniMax greedy algorithm into a deterministic 2-stage coloring scheme and show that it uses only $O(\log^2 n)$ colors. We refer to this algorithm as the *leveled UniMax greedy algorithm*.

Let x be the point which we currently insert. We assign a color to x in two steps. First we assign x to a *level*, denoted by $\ell(x)$. Once x is assigned to level $\ell(x)$ we give it an actual color among the set of colors dedicated to $\ell(x)$. We maintain the invariant that each color is used by at most one level. Formally, the colors that we use are pairs $(\ell(x), c(x)) \in \mathbb{Z}^2$, where $\ell(x)$ is the level of x and $c(x)$ is its integer color within that level.

Modifying the definition from the UniMax greedy algorithm, we say that point x *sees* point y (or that point y is *visible* to x) if and only if for every point z between x and y, $\ell(z) < \ell(y)$. When x is inserted, we set $\ell(x)$ to be the smallest level ℓ such that either to the left of x or to the right of x (or in both directions) there is no point y visible to x at level ℓ.

To give x a color, we now consider only the points of level $\ell(x)$ that x can see.

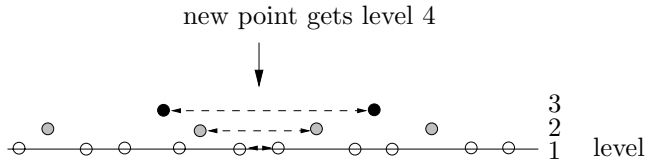


FIG. 3.1. *Illustrating the* 2*-stage deterministic algorithm. An insertion order that realizes the depicted assignment of levels to points is to first insert all level-*1 *points from left to right, then insert the level-*2 *points from left to right, and then the level-*3 *points.*

That is, we discard every point y such that $\ell(y) \neq \ell(x)$, and every point y such that $\ell(y) = \ell(x)$ and there is a point z between x and y such that $\ell(z) > \ell(y)$. We apply the UniMax greedy algorithm so as to color x with respect to the sequence P_x of the remaining points, using the colors of level $\ell(x)$ only. That is, we give x the color $(\ell(x), c(x))$, where $c(x)$ is the smallest color that ensures that the coloring of P_x maintains the unique maximum color condition. This completes the description of the algorithm. See Figure 3.1 for an illustration.

We begin the analysis of the algorithm by making a few observations on its performance.

(a) Suppose that a point x is inserted and is assigned to level $i > 1$. Since x was not assigned to any level $j < i$, it must see a point ℓ_j at level j that lies to its left, and another such point r_j that lies to its right. Let $E_j(x)$ denote the interval $[\ell_j, r_j]$. Note that, by definition, these intervals are *nested*, that is, $E_j(x) \subset E_k(x)$ for $j < k < i$. See Figure 3.1.

(b) We define a *run* at level i to be a maximal sequence of points $x_1 < x_2 < \cdots < x_k$ at level i, such that all points between x_1 and x_k that are distinct from $x_2, x_3, \ldots, x_{k-1}$ are assigned to levels smaller than i. Whenever a new point x is assigned to level i and is inserted into a run of that level, it is always inserted either to the left or to the right of all points in the run. Moreover, the actual color that x gets is determined solely from the colors of the points already in the run. See Figure 3.1.

(c) The runs keep evolving as new points are inserted. A run may either grow when a new point of the same level is inserted at its left or right ends (note that other points at smaller levels may separate the new point from the former end of the run) or split into two runs when a point of a higher level is inserted somewhere between its ends.

(d) As in observation (a), the points at level i define *intervals*, called *i-intervals*. Any such interval E is a contiguous subsequence $[x, y]$ of P, so that x and y are both at level i and all the points between x and y have smaller levels. E is formed when the second of its endpoints, say x, is inserted. We say that x *closes* the interval E and refer to it as a *closing point*. Note that, by construction, x cannot close another interval.

(e) Continuing observation (a), when x is inserted, it *destroys* the intervals $E_j(x)$, for $j < i$, into which it is inserted, and only these intervals. That is, each of these intervals now contains a point with a level greater than that of its endpoints, so it is no longer a valid interval. We charge x to the set of the closing endpoints of all these intervals. Clearly, none of these points will ever be charged again by another insertion (since it is the closing endpoint of only one interval, which is now destroyed). We maintain a forest F, whose nodes are all the points of P. The leaves of F are all the points at level 1. When a new point x is inserted, we make it a new root of F, and the parent of all the closing points that it charges. Since these points have smaller

levels than x, and since none of these points becomes a child of another parent, it follows that F is indeed a forest.

Note that the nonclosing points can only be roots of trees of F. Note also that a node at level i has exactly $i-1$ children, exactly one at each level $j < i$. Hence, each tree of F is a *binomial tree* (see [7]); if its root has level i, then it has 2^i nodes.

This implies that if m is the maximal level assigned after n points have been inserted, then we must have $2^m \le n$, or $m \le \log n$. That is, the algorithm uses at most $\log n$ levels.

We next prove that our algorithm uses only $O(\log n)$ colors at each level. We recall the way runs evolve: They grow by adding points at their right or left ends, and split into prefix and suffix subruns, when a point with a larger level is inserted in their middle.

LEMMA 3.1. *At any time during the insertion process, the colors assigned to the points in a run form a sequence of the form $C_0(a,b)$ (as defined in section 2). Moreover, when the jth smallest color of level i is given to a point x, the run to which x is appended has at least $2^{j-2}+1$ elements (including x).*

Proof. The proof proceeds by induction through the sequence of insertion steps and is based on the following observation. Let σ be a contiguous subsequence of the complete binary tree sequence S_{k-1}, and let x be a point added, say, to the left of σ. If we assign to x color $c(x)$, using the UniMax greedy algorithm, then $(c(x))\|\sigma$ is a contiguous subsequence of either S_{k-1} or S_k. The latter happens only if σ contains $S_{k-2}\|(k-1)$ as a prefix. Symmetric properties hold when x is inserted to the right of σ. We omit the straightforward proof of this observation. □

As a consequence, we obtain the following result.

THEOREM 3.2. (a) *The algorithm uses at most $(2+\log n)\log n$ colors.*

(b) *At any time, the coloring is CF.*

(c) *In the worst case the algorithm may be forced to use $\Omega(\log^2 n)$ colors after n points are inserted.*

Proof. (a) We have already argued that the number of levels is at most $\log n$. Within a level i, the kth smallest color is assigned when a run contains at least 2^{k-2} points. Hence $2^{k-2} \le n$, or $k \le 2+\log n$, and (a) follows.

To show (b), consider an arbitrary interval I. Let ℓ be the highest level of a point in I. Let $\sigma = (y_1, y_2, \ldots, y_j)$ be the sequence of the points in I of level ℓ. Since ℓ is the highest level in I, σ is a contiguous subsequence of some run, and, by Lemma 3.1, the sequence of the colors of its points is also of the form $C_0(a', b')$. Hence, there is a point $y_i \in \sigma$ which is uniquely colored among $y_1, y_2, \ldots, y_j$ by a color of level ℓ.

To show (c), we construct a sequence P so as to force its coloring to proceed level by level. We first insert 2^{k-1} points from left to right, thereby making them all be assigned to level 1 and colored with k different colors of that level. Let P_1 denote the set of these points. We next insert a second batch of 2^{k-2} points from left to right. The first point is inserted between the first and second points of P_1, the second point between the third and fourth points of P_1, and so on, where the jth new point is inserted between the $(2j-1)$th and $(2j)$th points of P_1. By construction, all points in the second batch are assigned to level 2, and they are colored with $k-1$ different colors of that level. Let P_2 denote the set of all points inserted so far. P_2 is the concatenation of 2^{k-2} triples, where the levels in each triple are $(1,2,1)$. We now insert a third batch of 2^{k-3} points from left to right. The first point is inserted between the first and second triples of P_2, the second point between the third and fourth triples of P_2, and so on, where the jth new point is inserted between the $(2j-1)$th and $(2j)$th triples of P_2. By construction, all points in the third batch are

assigned to level 3, and they are colored with $k-2$ different colors of that level.

The construction is continued in this manner. Just before inserting the ith batch of 2^{k-i} points, we have a set P_{i-1} of $2^{k-1}+\cdots+2^{k-i+1}$ points, which is the concatenation of 2^{k-i+1} tuples, where the sequences of levels in each of these tuples are all identical and equal to the "complete binary tree sequence" $C_0(1, 2^{i-1}-1)$, as defined in section 2 (whose elements now encode levels rather than colors). The points of the ith batch are inserted from left to right, where the jth point is inserted between the $(2j-1)$th and $(2j)$th tuples of P_{i-1}. By construction, all points in the ith batch are assigned to level i and are colored with $k-i+1$ different colors of that level. Proceeding in this manner, we end the construction by inserting the $(k-1)$th batch, which consists of a single point that is assigned to level k. Altogether we have inserted $n = 2^k - 1$ points and forced the algorithm to use $k+(k-1)+\cdots+1 = k(k+1)/2 = \Omega(\log^2 n)$ different colors. □

Remark. One can modify the algorithm so that the set of colors that it uses can be identified with (a subset of a prefix of) the integers, and so that it maintains the property of the UniMax greedy algorithm: At any time t and for any interval I, there is a unique point in I with maximum color. The modified algorithm also uses $O(\log^2 n)$ colors.

Specifically, we proceed as follows. Suppose first that n is known in advance. Order the pairs $(k,i) \in \{1,\ldots,\log n\}\times\{1,\ldots,2+\log n\}$ lexicographically, i.e., $(k,i) < (k',i')$ if $k<k'$ or $(k=k'$ and $i<i')$. Let $f(k,i)$ be the rank of the pair (k,i) in this lexicographic order. Then the set of numbers $f(k(p),i(p))$, where $p \in P$ is assigned level $k(p)$ and the $i(p)$th color within that level, is (a subset of) a prefix of the integers, and the unique maximum color property is satisfied.

If n is not known in advance, we apply the same strategy as the one discussed at the end of the preceding section. That is, when the number of inserted points reaches one of the values 2^{2^i} for $i \geq 0$, we start coloring new points with a completely new set of colors, which are mapped lexicographically onto integer values that are larger than the largest integer color used so far.

4. An efficient randomized algorithm. We next modify the UniMax greedy deterministic algorithm into the following randomized algorithm, which we call the *randomized UniMax greedy algorithm.* The randomized UniMax greedy algorithm does not partition the points into levels but assigns a color directly, using the following randomized variant of the UniMax greedy strategy.

Let p be the next point inserted. Recall that p *sees* a point x (alternatively, the color $c(x)$) if all the colors of points between p and x (exclusive) have color smaller than $c(x)$. We say that p is *eligible* for color m if p does not see m. To give p a color, we scan all colors in increasing order. For each color i, if p is not eligible for color i, we continue to color $i+1$. Otherwise, if p is eligible for color i, we set $c(p)=i$ with probability $1/2$ and continue to color $i+1$ with probability $1/2$.

By the same reasoning as for the UniMax greedy algorithm, the coloring produced by the randomized UniMax greedy algorithm is CF at any stage. We next show that it uses $O(\log n)$ colors with high probability.

LEMMA 4.1. *If the algorithm reaches color i when processing a point p, then p gets the color i with probability at least $1/8$. More formally, let C_i (resp., $C_{\geq i}$) be the random variable which is equal to the set of points of color i (resp., of color $\geq i$). Then*

$$\mathbf{Pr}\Big\{p \in C_i \mid p \in C_{\geq i}\Big\} \geq \frac{1}{8}.$$

Proof. Assume first that p is neither the leftmost nor the rightmost point at the time of its insertion. Let Q be the set of points inserted before p. Fix a point $p_\ell \in Q$ to the left of p, and a point $p_r \in Q$ to the right of p. Consider all random choices (which we refer to as "executions") of the randomized UniMax algorithm, in which p_ℓ, p, and p_r end up as three consecutive points in $C_{\geq i}$.

In at least $1/2$ of these executions p_ℓ gets a color greater than i (either because it is ineligible for color i or because the coin toss has moved it to color $i+1$); similarly, in at least $1/2$ of these executions p_r gets a color greater than i. Since the coin tosses of p_r are independent of those of p_ℓ, both p_ℓ and p_r get color greater than i in at least $1/4$ of these executions. In these cases, p is eligible for color i and with probability $1/2$ does get that color. Hence, in at least $1/8$ of the above executions p gets color i. Since this is true for every choice of p_ℓ and p_r, the lemma follows.

If p is the leftmost or rightmost point, then it is eligible for color i (assuming it has reached $C_{\geq i}$) with probability $1/2$, and if p is the first inserted point, then it is eligible for color i with probability 1. Hence, the preceding argument implies the lemma in this case too. □

THEOREM 4.2. *The randomized UniMax greedy algorithm uses $O(\log n)$ colors with high probability.*

Proof. Using the same notation as above, Lemma 4.1 implies that

$$\mathbf{E}\left(|C_{\geq i+1}|\right) \leq \frac{7}{8}\mathbf{E}\left(|C_{\geq i}|\right).$$

Since $|C_{\geq 1}| = n$, we have for $i \geq 1$

$$\mathbf{E}\left(|C_{\geq i+1}|\right) \leq \left(\frac{7}{8}\right)^i n.$$

For $i = c\log_{8/7} n$, we get that $\mathbf{E}\left(|C_{\geq i+1}|\right) \leq 1/n^{c-1}$. Hence, by Markov's inequality,

$$\mathbf{Pr}\left\{|C_{\geq i+1}| \geq 1\right\} \leq 1/n^{c-1},$$

from which the lemma follows. □

Remark. We leave it as an open problem to determine whether a nonoblivious adversary can cause the algorithm to use more than $\Theta(\log n)$ colors.

5. Random insertion order. In this section we consider the special case where the points are inserted in a *random* order, and where we color them by the UniMax greedy algorithm of section 2. We have simulated the execution of the UniMax greedy algorithm under such an insertion order. The results of the simulation strongly suggest the following conjecture.

CONJECTURE 5.1. *For each integer $k \geq 1$, the expected frequency of the color k in $C(P(t))$, as generated by the UniMax greedy algorithm, converges to $\frac{1}{3}\left(\frac{2}{3}\right)^{k-1}$ as $t \to \infty$.*

Assuming Conjecture 5.1, the following is an easy consequence.

COROLLARY 5.2. *If each point is inserted into P at a random place, the expected value of $c_{\max}(P(t))$, under the UniMax greedy algorithm, is $O(\log t)$. This also holds with high probability if the constant of proportionality is chosen sufficiently large.*

Proof. Let $P(n)$ be a set of n points inserted in a random order. Let X_k be a random variable counting the number of points in $P(n)$ that were colored with k

by the UniMax greedy algorithm. Let I_k be the indicator variable for the color k to appear at all.

We are interested in the number of colors used, that is, $Y := \sum_k I_k$.

Assume that $\mathbf{E}(X_k) = \frac{1}{3}(\frac{2}{3})^{k-1}n$. Then, using Markov's inequality, $\mathbf{E}(I_k) = \mathbf{Pr}\{I_k = 1\} = \mathbf{Pr}\{X_k \geq 1\} \leq \mathbf{E}(X_k)$. Hence,

$$\begin{aligned}\mathbf{E}(Y) = \mathbf{E}\left(\sum_{1\leq k} I_k\right) &= \mathbf{E}\left(\sum_{1\leq k<1+\log_{3/2} n} I_k\right) + \mathbf{E}\left(\sum_{k\geq 1+\log_{3/2} n} I_k\right)\\ &\leq 1 + \log_{3/2} n + \sum_{k\geq 1+\log_{3/2} n} \frac{1}{3}\left(\frac{2}{3}\right)^k n\\ &\leq 1 + \log_{3/2} n + \sum_{i\geq 0} \frac{1}{3}\left(\frac{2}{3}\right)^i\\ &= \log_{3/2} n + 2.\end{aligned}$$

Arguing as in the proof of Theorem 4.2, we also have

$$\begin{aligned}\mathbf{Pr}\Big\{\text{more than } c\log_{3/2} n \text{ colors are used}\Big\} &= \mathbf{Pr}\Big\{I_{\lceil c\log_{3/2} n\rceil} = 1\Big\}\\ &\leq \frac{1}{3}\left(\frac{2}{3}\right)^{\lceil c\log_{3/2} n\rceil - 1} n \leq \frac{1}{2n^{c-1}}.\end{aligned}$$ □

At this stage, we do not have a complete proof of Conjecture 5.1. We do have some partial results that we now present. In particular, they show that Conjecture 5.1 holds for $k = 1, 2, 3$. Completing the proof is one of the major open problems raised in this paper.

LEMMA 5.3. *The expected number of points assigned the color* 1*, after a random insertion of* t *points, is* $\frac{t+1}{3}$ *for* $t \geq 2$.

Proof. Denote by X_i the random variable whose value is the number of 1's after the insertion of the first i points. Then $X_{i+1} = X_i + Y_i$, where Y_i is an indicator variable, equal to 1 if the $(i+1)$st point p_{i+1} is colored by 1, and to 0 otherwise. Note that p_{i+1} is colored by 1 if and only if it is inserted at a place that is not adjacent to any point colored 1. Each of the current X_i 1-colored points has two adjacent insertion places, and all these places are distinct, because $P(i)$ does not contain two adjacent points colored 1. Hence, out of the $i+1$ available insertion places, $i+1-2X_i$ will cause p_{i+1} to be colored 1. Taking expectations, we obtain

$$\begin{aligned}&\mathbf{E}(X_{i+1}) = \mathbf{E}(X_i) + \mathbf{E}(Y_i) = \mathbf{E}(X_i) + \mathbf{E}(\mathbf{E}(Y_i \mid X_i))\\ &= \mathbf{E}(X_i) + \mathbf{E}\left(\frac{i+1-2X_i}{i+1}\right) = \mathbf{E}(X_i) + \frac{i+1-2\mathbf{E}(X_i)}{i+1},\end{aligned}$$

or $\mathbf{E}(X_{i+1}) = \frac{i-1}{i+1}\mathbf{E}(X_i)+1$, for $i \geq 2$. The solution of this recurrence, with the initial value $\mathbf{E}(X_2) = 1$, is easily seen to be $\mathbf{E}(X_t) = \frac{t+1}{3}$ for $t \geq 2$. □

Analysis for $k \geq 2$. We next present a framework for estimating the expected number of points that are assigned the color k for $k \geq 2$. We apply this framework to get a complete solution for $k = 2, 3$. We fix k, and define a *k-state* to be any valid contiguous sequence of colors in $\{1, \ldots, k\}$ that may show up in $C(P(t))$, delimited on both sides by $*$, which designates a color greater than k. The validity of a state

means that it satisfies the unique maximum color invariant: Any contiguous nonempty subsequence of s has a unique largest element. We refer to the portion of a state that excludes the $*$'s as its *core*.

Denote by S_k the set of all k-states. For example, the set S_2 consists of the following states:

(5.1)
$$s_1 = \langle **\rangle, \quad s_2 = \langle *1* \rangle, \quad s_3 = \langle *2* \rangle, \quad s_4 = \langle *12* \rangle, \quad s_5 = \langle *21* \rangle, \quad s_6 = \langle *121* \rangle.$$

For example, the sequence $C(P(t)) = (1\ 2\ 1\ 3\ 2\ 4\ 2\ 1\ 3\ 5\ 1\ 2\ 3)$ is decomposed into the following sequence of 2-states:

$$\big(\langle *121* \rangle, \quad \langle *2* \rangle, \quad \langle *21* \rangle, \quad \langle ** \rangle, \quad \langle *12* \rangle, \quad \langle ** \rangle\big).$$

We denote by S_k^+ the subset of S_k consisting of those k-states that contain the color k (necessarily at a unique location), and by S_k^- the subset of those states that do not contain k. We refer to states in S_k^+ (resp., S_k^-) as *major* k-states (resp., *minor* k-states). The *size* $|s|$ of a k-state s is the length of its core plus 1; it designates the number of places in s at which a new point can be inserted. For example, for 2-states we have $S_2^- = \{s_1, s_2\}$, $S_2^+ = \{s_3, s_4, s_5, s_6\}$. Also, we have $|s_1| = 1$, $|s_2| = |s_3| = 2$, $|s_4| = |s_5| = 3$, and $|s_6| = 4$.

Let $s \in S_k^+$. It has the form $(*ukv*)$, where u and v can be regarded as the cores of two respective $(k-1)$-states, s_L and s_R. We refer to s_L and s_R as the *left wing* and the *right wing* of s, respectively. We have $|s| = |s_L| + |s_R|$. Care should be exercised in the treatment of s_L and s_R. Specifically, we will consider the actual sequence of colors $C(P(t))$ as a concatenation of states, which depends on the choice of k. We denote by $C^{(k)}(t)$ the (unique) partition of $C(P(t))$ into the concatenation of k-states, and refer to it as the *k-scenario*. Then, for a major state $s \in S_k^+$, its left and right wings are not counted as separate states in the k-scenario but as states in the $(k-1)$-scenario.

We need one more notion. When we insert a new point into a k-state s, there are two possible outcomes: (i) The point gets a color smaller than or equal to k, in which case s is transformed to another, *single* state in S_k. (ii) The point gets a color greater than k, in which case s is split into two new k-states. Note that, for case (ii) to occur, s must be a *major* state (if s were minor, we could have assigned the color k to the new point). Moreover, in this case one of the two new states, s', must be a major state, and the other, s'', must be minor. We refer to this case by saying that *s spawns s''* and is *transformed* into s'. (Note that not every insertion into a major state necessarily causes a spawning.)

It is easy to show that the size $|S_k|$ of S_k satisfies $|S_{k+1}| = |S_k| + |S_k|^2$; thus $|S_k|$ is doubly exponential in k. We have $|S_1| = 2$, $|S_2| = 6$, $|S_3| = 42$, and $|S_4| = 1806$.

Let k be fixed. For states $s, r \in S_k$, we denote by a_{sr} the expected change in the number of states r that are generated by an insertion of a new point, conditioned on having chosen an insertion place at a state s (within $C^{(k)}$). For example, for $k = 2$ we have (see (5.1) for the notation)

$$a_{s_4 s_1} = a_{s_4 s_2} = a_{s_4 s_3} = a_{s_4 s_6} = \frac{1}{3} \quad \text{and} \quad a_{s_4 s_4} = -\frac{2}{3}$$

(in two of the three possible insertion places, s_4 is destroyed by the insertion, and in the third insertion it survives, so the net expected increase in the number of s_4-states

is $0 \cdot \frac{1}{3} + (-1) \cdot \frac{2}{3} = -\frac{2}{3}$). Put $w_{sr} = |s|a_{sr}$, and let W denote the resulting matrix (w_{sr}).

We first provide some intuitive and informal derivation of the equations that we will rigorously derive shortly. Let $M_s^{(t)}$ denote the random variable equal to the number of k-states s in $C(P(t))$. Define the *frequency* of state s at time t to be $X_s^{(t)} = M_s^{(t)}/(t+1)$. Note that $|s|X_s^{(t)}$ is the frequency of the insertion places that belong to occurrences of s in $C(P(t))$. In particular, $\sum_{s \in S_k} |s|X_s^{(t)} = 1$ for each t. We also have

$$(t+2)\mathbf{E}(X_r^{(t+1)}) = (t+1)\mathbf{E}(X_r^{(t)}) + \sum_{s \in S_k} |s|a_{sr}\mathbf{E}(X_s^{(t)}). \tag{5.2}$$

Indeed, $|s|X_s^{(t)}$ is the probability that the next insertion place belongs to an occurrence of state s in $C(P(t))$, and a_{sr} is the corresponding conditional expected change in the number of occurrences of state r. Since $M_r^{(t)} = (t+1)X_r^{(t)}$ (resp., $M_r^{(t+1)} = (t+2)X_r^{(t+1)}$) is the number of occurrences of state r at time t (resp., $t+1$), the equality follows.

Letting $t \to \infty$, applying an informal limit process to (5.2), and denoting the limit of $\mathbf{E}(X_s^{(t)})$ as X_s for $s \in S_k$, we arrive at the equations

$$X_r = \sum_{s \in S_k} |s|a_{sr}X_s = \sum_{s \in S_k} w_{sr}X_s.$$

We now proceed to justify this process rigorously.

Existence of limiting frequencies. The random insertion order defines in a natural way a *multitype branching process* (see [1]). We briefly review the ingredients of the theory of branching processes that we need to apply. A (discrete) branching process of this kind manipulates objects (referred to as "particles") that can have a finite number m of types. Each type i is associated with weights $(\xi_{i,J})$, where J is a multiset of types. The weight $\xi_{i,J}$ should be thought of as the relative frequency at which a particle of type i gives birth to the multiset J (for each type j that appears μ times in J, the particle generates μ new particles of type j). Each particle giving birth dies immediately after doing so. Set $\xi_i = \sum_J \xi_{i,J}$. The process may then be formally defined as follows. Let $S(t)$ be the population at time t. Choose $x \in S(t)$ with probability $\xi_{i(x)}/\sum_{y \in S(t)} \xi_{i(y)}$, where $i(u)$ is the type of particle u. Then x gives birth to the multiset J with probability $\xi_{i(x),J}/\xi_{i(x)}$ and then dies.

In our case, the different particle types correspond to different state types in S_k. A state s of length ℓ has total weight ℓ. If some insertions into s produce the single state s' (without spawning), then $\xi_{s,\{s'\}} = j$, where j is the number of places at which this occurs. If some j insertions produce two states s', s'' (by spawning), then $\xi_{s,\{s',s''\}} = j$. The entries of our transition matrix W are then defined as $w_{sr} = \sum_{r \in J} \xi_{s,J}$ for $r \neq s$, and $w_{ss} = \left(\sum_{r \in J} \xi_{s,J}\right) - |s|$. See pp. 200–202 in [1] for a similar construction of a transition matrix for general multitype processes (where the matrix is called the *infinitesimal generator* of a corresponding semigroup of mean matrices).

A standard trick in the theory of branching processes is to embed discrete branching processes of the kind described above into continuous-time branching processes, in which particles give birth in continuous time. More specifically, $S(t)$ evolves in continuous time. For any fixed time t, we associate, with each $x \in S(t)$ and each multiset J, an exponential random variable with rate $w_{i(x),J}$. We then take the one

with the smallest actual value—suppose this is the variable $w_{i(x'),J'}$ and that it has the value h. Now the population at time $t+h$ is obtained from the population at time t by killing x' and replacing it by J'. We now obtain a new population $S(t+h)$ and a new collection of exponential random variables, and the process continues.

If we extract from the continuous branching process only those times at which new particles are born, we obtain exactly the same discrete process that we started with (see [1] for details). In the terminology of the theory of branching processes, the discrete and continuous processes are the same, up to a time change. The reason for this roundabout reasoning is that the theory of the continuous-time branching process is better developed and provides machinery for proving the existence of limit frequencies and for analyzing their properties. In particular, the limiting frequencies for the new continuous process (whose existence is established next) are identical to those of the original discrete process.

It is easy to see that the (continuous) branching process just defined is *supercritical* and satisfies the $Z \log Z$ moment condition (see, e.g., [1] for background and details). It therefore follows (see, e.g., Theorem 2, p. 206, in [1]) that the limiting frequencies exist almost surely. We let X_s denote the expected limit relative frequency of state s in $C(P(t))$ when $t \to \infty$, where the nonlimit frequencies are as defined above.

In addition, the just cited Theorem 2, p. 206, in [1] asserts that the limiting distribution $X = (X_s)_{s \in S_k}$ is given by the eigenvector of W^T corresponding to the largest eigenvalue. In our case, this does indeed coincide with our informal derivation and means that X satisfies the linear system

$$(W^T X)_r = \sum_{s \in S_k} w_{sr} X_s = X_r, \qquad r \in S_k. \tag{5.3}$$

For example, for $k=2$, the transition weights a_{sr} between the six states listed in (5.1) are given in the following matrix A, where $A_{ij} = a_{s_i s_j}$ (the fourth row of A has already been discussed):

$$A = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & -1 & \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & -\frac{2}{3} & 0 & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & -\frac{2}{3} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{2} & 0 & \frac{1}{4} & \frac{1}{4} & -\frac{1}{2} \end{pmatrix},$$

and

$$W = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 1 & 1 & 0 \\ 0 & 0 & -2 & 1 & 1 & 0 \\ 1 & 1 & 1 & -2 & 0 & 1 \\ 1 & 1 & 1 & 0 & -2 & 1 \\ 2 & 2 & 0 & 1 & 1 & -2 \end{pmatrix}.$$

The system of equations for the limit distribution is $W^T X = X$. To normalize X, we extend it by the equation

$$\sum_i |s_i| X_i = X_1 + 2X_2 + 2X_3 + 3X_4 + 3X_5 + 4X_6 = 1,$$

which expresses the fact that the sum of lengths of the 2-states that compose $C(P(t))$ is equal to $|C(P(t))|$ (see above for a similar equation for the nonlimit frequencies $X_s^{(t)}$). The solution of the extended system is

$$X = \left(\frac{1}{9}, \frac{1}{9}, \frac{2}{45}, \frac{1}{15}, \frac{1}{15}, \frac{2}{45}\right).$$

In particular, the expected limit frequency of color 1 is $X_2 + X_4 + X_5 + 2X_6 = \frac{1}{3}$ (in accordance with Lemma 5.3), and the expected limit frequency of color 2 is $X_3 + X_4 + X_5 + X_6 = \frac{2}{9}$. We have thus verified Conjecture 5.1 for $k = 2$.

LEMMA 5.4. *The limit frequency of color* 2 *is* 2/9.

Analysis of 3*-states.* The same machinery can be applied to the 42 states in S_3. The solution of (5.3) for $k = 3$ is presented in Table 5.1.

By adding up the frequencies of all major 3-states (those that contain the color 3), we verify Conjecture 5.1 for $k = 3$.

LEMMA 5.5. *The limit frequency of color* 3 *is* 4/27.

Open problem. Find closed-form expressions for the state frequencies for $k = 3$ (using the data in Table 5.1) and for $k > 3$. This may lead to a simple inductive proof of Conjecture 5.1.

Further analysis of k*-states.* The system (5.3) becomes considerably harder to solve explicitly for larger values of k, so we look for simpler relationships. Put

$$N_k = \sum_{s \in S_k^+} X_s, \qquad Z_k = \sum_{s \in S_k^-} X_s.$$

Note that N_k is the expected frequency of color k. Recall that Conjecture 5.1 says that $N_k = \frac{1}{3}\left(\frac{2}{3}\right)^{k-1}$.

LEMMA 5.6. *For each* $k \geq 2$ *we have* $2N_k + Z_k = N_{k-1} + Z_{k-1}$.

Proof. Let s be a state in S_k^+, and let s_L (resp., s_R) denote the state obtained by taking the portion of s to the left (resp., right) of (the unique) k and appending $*$ at the right (resp., left). If we repeat this splitting process to each state of S_k^+ in $C(P(t))$ and add to the output all states in S_k^- (which we leave intact), we obtain the set of all states of S_{k-1} that appear in $C(P(t))$. The sum of the frequencies of these states is clearly $N_{k-1} + Z_{k-1}$. On the other hand, by our construction, this sum is $2N_k + Z_k$; thus the lemma follows. □

The following conjecture is *equivalent* to Conjecture 5.1.

CONJECTURE 5.7. $N_k = Z_k$ *for each* $k \geq 1$.

We verify the conjecture for $k = 1$, where $N_1 = Z_1 = \frac{1}{3}$; for $k = 2$, where $N_2 = Z_2 = \frac{2}{9}$; and for $k = 3$, where $N_3 = Z_3 = \frac{4}{27}$ (see Table 5.1).

Assuming that Conjecture 5.7 holds and combining it with Lemma 5.6, we obtain $3N_k = 2N_{k-1}$, for $k \geq 2$, and $N_1 = \frac{1}{3}$. Hence

$$N_k = \frac{1}{3}\left(\frac{2}{3}\right)^{k-1}.$$

The converse direction is established in a similar manner: Conjecture 5.1 and Lemma 5.6 imply

$$Z_k = Z_{k-1} + N_{k-1} - 2N_k = Z_{k-1} - \frac{1}{9}\left(\frac{2}{3}\right)^{k-2},$$

for $k \geq 2$, and $Z_1 = \frac{1}{3}$. The solution of this recurrence is $Z_k = \frac{1}{3}\left(\frac{2}{3}\right)^{k-1} = N_k$, thus showing that the two conjectures are indeed equivalent.

TABLE 5.1

The frequencies of 3-states. The second column is the numerator of the frequency under the common denominator $47628000 = 2^5 3^5 5^3 7^2$.

State	Frequency	Numerator	As a fraction	With factored denominator
0	0.03704	1764000	1/27	$1/3^3$
1	0.03704	1764000	1/27	$1/3^3$
12	0.02222	1058400	1/45	$1/3^2 5^1$
21	0.02222	1058400	1/45	$1/3^2 5^1$
2	0.01481	705600	2/135	$2/3^3 5^1$
121	0.01481	705600	2/135	$2/3^3 5^1$
13	0.00800	381024	1/125	$1/5^3$
31	0.00800	381024	1/125	$1/5^3$
12321	0.00388	184800	11/2835	$11/3^4 5^1 7^1$
3	0.00948	451584	32/3375	$32/3^3 5^3$
131	0.00652	310464	22/3375	$22/3^3 5^3$
123	0.00366	174440	89/24300	$89/2^2 3^5 5^2$
321	0.00366	174440	89/24300	$89/2^2 3^5 5^2$
1231	0.00737	350840	179/24300	$179/2^2 3^5 5^2$
1321	0.00737	350840	179/24300	$179/2^2 3^5 5^2$
32	0.00167	79576	203/121500	$203/2^2 3^5 5^3$
23	0.00167	79576	203/121500	$203/2^2 3^5 5^3$
132	0.00686	326536	833/121500	$833/2^2 3^5 5^3$
231	0.00686	326536	833/121500	$833/2^2 3^5 5^3$
213121	0.00156	74466	197/126000	$197/2^4 3^2 5^3 7^1$
121312	0.00156	74466	197/126000	$197/2^4 3^2 5^3 7^1$
2321	0.00233	111160	397/170100	$397/2^2 3^5 5^2 7^1$
1232	0.00233	111160	397/170100	$397/2^2 3^5 5^2 7^1$
232	0.00093	44464	397/425250	$397/2^1 3^5 5^3 7^1$
121321	0.00191	90755	2593/1360800	$2593/2^5 3^5 5^2 7^1$
123121	0.00191	90755	2593/1360800	$2593/2^5 3^5 5^2 7^1$
12312	0.00307	146405	4183/1360800	$4183/2^5 3^5 5^2 7^1$
21321	0.00307	146405	4183/1360800	$4183/2^5 3^5 5^2 7^1$
12131	0.00344	163928	20491/5953500	$20491/2^2 3^5 5^3 7^2$
13121	0.00344	163928	20491/5953500	$20491/2^2 3^5 5^3 7^2$
1312	0.00485	231208	28901/5953500	$28901/2^2 3^5 5^3 7^2$
2131	0.00485	231208	28901/5953500	$28901/2^2 3^5 5^3 7^2$
1213	0.00588	279848	34981/5953500	$34981/2^2 3^5 5^3 7^2$
3121	0.00588	279848	34981/5953500	$34981/2^2 3^5 5^3 7^2$
213	0.00835	397528	49691/5953500	$49691/2^2 3^5 5^3 7^2$
312	0.00835	397528	49691/5953500	$49691/2^2 3^5 5^3 7^2$
2132	0.00198	94467	31489/15876000	$31489/2^5 3^4 5^3 7^2$
2312	0.00198	94467	31489/15876000	$31489/2^5 3^4 5^3 7^2$
21312	0.00240	114326	57163/23814000	$57163/2^4 3^5 5^3 7^2$
1213121	0.00099	47206	23603/23814000	$23603/2^4 3^5 5^3 7^2$
12132	0.00104	49397	49397/47628000	$49397/2^5 3^5 5^3 7^2$
23121	0.00104	49397	49397/47628000	$49397/2^5 3^5 5^3 7^2$

6. Lower bound for online CF coloring in the plane. We finally show that online CF coloring of points in the plane, with respect to disks (of arbitrary radii), may require n colors in the worst case and is therefore quite impractical. (Nevertheless, as mentioned in the introduction, the problem can be solved with many fewer colors for other kinds of ranges; see [5, 6].)

THEOREM 6.1. *There exists a sequence P of n points in the plane, so that when these points are inserted according to their order in P, any online CF coloring scheme with respect to disks has to use n different colors.*

Proof. We construct a sequence $P = (p_1, p_2, \ldots, p_n)$ with the following property:

(*) *For every $t = 2, 3, \ldots, n$, the edges of the Delaunay triangulation of the set $\{p_1, p_2, \ldots, p_t\}$ include all the edges $\{p_i, p_t\}$, $i =$*

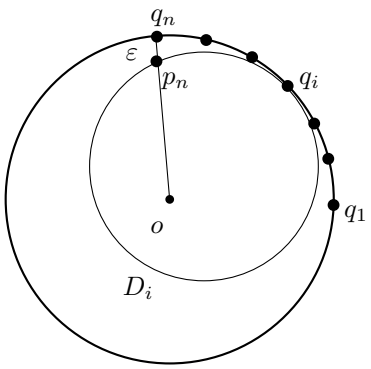


FIG. 6.1. *The construction that requires n colors for the planar case with disk ranges.*

$1, 2, \ldots, t-1$.

We prove the following stronger statement by induction on n:

For every n, every choice of distinct points $q_1, q_2, \ldots, q_n$ on the unit circle $\mathbb{S}^1$, and every $\varepsilon > 0$, there exists a sequence $(p_1, p_2, \ldots, p_n)$ with the property (*) such that $\|p_i - q_i\| \leq \varepsilon$ and p_i lies on the radius oq_i for every i.

For the induction step, given $q_1, \ldots, q_n$ and $\varepsilon < \frac{1}{2}$, let p_n be obtained by moving q_n by ε towards the center o of $\mathbb{S}^1$. We note that the Delaunay graph of $\{q_1, q_2, \ldots, q_{n-1}, p_n\}$ contains all edges $\{q_i, p_n\}$, $i = 1, 2, \ldots, n$. Indeed, there is a circle γ_i tangent to $\mathbb{S}^1$ from the inside at q_i and passing through p_n, and the closed disk D_i bounded by γ_i contains q_i, p_n, and no other q_j. See Figure 6.1. Let $\delta_i > 0$ denote the minimum distance from any q_j, $j \neq i$, to D_i.

We apply the induction hypothesis with $q_1, \ldots, q_{n-1}$ and with $\varepsilon^* < \min\{\varepsilon, \delta_1, \ldots, \delta_{n-1}\}$, obtaining a sequence $(p_1, \ldots, p_{n-1})$. We can now verify that, by construction, for every $i = 1, 2, \ldots, n-1$, the disk D_i contains p_i and p_n but no other p_j. □

7. Conclusion. The paper still leaves many open problems, some of which have been listed earlier. Here are several concluding open problems.

(a) Theorem 6.1 and the initial encouraging results of Chen, Kaplan, and Sharir [6] and of Bar-Noy, Cheilaris, and Smorodinsky [3], as reviewed in the introduction, raise many interesting open problems, such as the following: (i) Obtain *deterministic* algorithms with good performance for the cases studied in [6], viz. where the ranges are half-planes, congruent disks, and nearly equal axis-parallel rectangles. (ii) Improve further the performance of the algorithms of [6]. (iii) Find solutions with good performance for other ranges, such as arbitrary axis-parallel rectangles. (iv) Extend the results to $d \geq 3$ dimensions.

(b) It is likely that the bound in Theorem 6.1 improves significantly if the points are chosen from some random distribution, extending our conjectured bounds of section 5.2 to two (and higher?) dimensions.

(c) Can one obtain better upper bounds for online *k-CF coloring* ($k \geq 2$) of points in the plane with respect to disks? Namely, online color the points so that, at any given time t and for any disk D, there is at least one color that is assigned to at least one but at most k points of $P(t) \cap D$. For $k = 1$, this is the CF coloring problem, where we have just shown a lower bound of n, but perhaps this can be improved when $k \geq 2$. See [14] for results concerning k-CF coloring in the static case.

(d) Finally, can one obtain an efficient randomized algorithm, for the online CF coloring problem on the line, that works against an adaptive adversary (that is, an adversary that observes the actions of the randomized online algorithm and decides where to insert the next point based on these actions)?

REFERENCES

[1] K. B. Athreya and P. E. Ney, *Branching Processes*, Grundlehren Math. Wiss. 196, Springer-Verlag, New York, 1972.

[2] A. Bar-Noy, P. Cheilaris, and S. Smorodinsky, *Conflict-free coloring for intervals: From offline to online*, in Proceedings of the 18th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2006), 2006, pp. 128–137.

[3] A. Bar-Noy, P. Cheilaris, and S. Smorodinsky, *Online Conflict-Free Colorings for Hypergraphs*, manuscript, 2006.

[4] A. Borodin and R. El Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, New York, 1998.

[5] K. Chen, *How to play a coloring game against a color-blind adversary*, in Proceedings of the 22nd Annual ACM Symposium on Computational Geometry, 2006, pp. 44–51.

[6] K. Chen, H. Kaplan, and M. Sharir, *Online CF Coloring for Halfplanes, Congruent Disks, and Axis-Parallel Rectangles*, manuscript, 2006.

[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

[8] J. S. Deogun, T. Kloks, D. Kratsch, and H. Müller, *On vertex ranking for permutation and other graphs*, in STACS 94, Lecture Notes in Comput. Sci. 775, P. Enjalbert, E. W. Mayr, and K. W. Wagner, eds., Springer-Verlag, Berlin, 1994, pp. 747–758.

[9] K. Elbassioni and N. Mustafa, *Conflict-free colorings for rectangle ranges*, in Proceedings of the 23rd International Symposium on Theoretical Aspects of Computer Science (STACS 2006), pp. 254–263.

[10] G. Even, Z. Lotker, D. Ron, and S. Smorodinsky, *Conflict-free colorings of simple geometric regions with applications to frequency assignment in cellular networks*, SIAM J. Comput., 33 (2003), pp. 94–136.

[11] A. Fiat, M. Levy, J. Matoušek, E. Mossel, J. Pach, M. Sharir, S. Smorodinsky, U. Wagner, and E. Welzl, *Online conflict-free coloring for intervals*, in Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2005, pp. 545–554.

[12] S. Har-Peled and S. Smorodinsky, *On conflict-free coloring of points and simple regions in the plane*, Discrete Comput. Geom., 34 (2005), pp. 47–70.

[13] J. Pach and G. Tóth, *Conflict-free colorings*, in Discrete and Computational Geometry—The Goodman–Pollack Festschrift, B. Aronov, S. Basu, J. Pach, and M. Sharir, eds., Springer-Verlag, Heidelberg, 2003, pp. 665–671.

[14] S. Smorodinsky, *Combinatorial Problems in Computational Geometry*, Ph.D. dissertation, School of Computer Science, Tel-Aviv University, Tel-Aviv, 2003.

[15] S. Smorodinsky, *On the chromatic number of some geometric hypergraphs*, in Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), Miami, 2006, pp. 316–323.

SIAM J. COMPUT.
Vol. 36, No. 5, pp. 1360–1375

IMPROVED COMBINATORIAL GROUP TESTING ALGORITHMS FOR REAL-WORLD PROBLEM SIZES*

DAVID EPPSTEIN†, MICHAEL T. GOODRICH†, AND DANIEL S. HIRSCHBERG†

Abstract. We study practically efficient methods for performing combinatorial group testing. We present efficient nonadaptive and two-stage combinatorial group testing algorithms, which identify the at most d items out of a given set of n items that are defective, using fewer tests for all practical set sizes. For example, our two-stage algorithm matches the information-theoretic lower bound for the number of tests in a combinatorial group testing regimen.

Key words. combinatorial group testing, Chinese remaindering, Bloom filters

AMS subject classification. 68Q25

DOI. 10.1137/050631847

1. Introduction.

The problem of combinatorial group testing dates back to World War II, for the problem of determining which in a group of n blood samples contain the syphilis antigen (hence, are contaminated). Formally, in combinatorial group testing, we are given a set of n items, at most d of which are defective (or contaminated), and we are interested in identifying exactly which of the n items are defective. In addition, items can be "sampled" and these samples can be "mixed" together, and so tests for contamination can be applied to arbitrary subsets of these items. The result of a test may be positive, indicating that at least one of the items of that subset is defective, or negative, indicating that all items in that subset are good. Example applications that fit this framework include the following:

- *Screening blood samples for diseases.* In this application, items are blood samples and tests are disease detections done on mixtures taken from selected samples.
- *Screening vaccines for contamination.* In this case, items are vaccines and tests are cultures done on mixtures of samples taken from selected vaccines.
- *Clone libraries for a DNA sequence.* Here, the items are DNA subsequences (called *clones*) and tests are done on pools of clones to determine which clones contain a particular DNA sequence (called a *probe*) [10].
- *Data forensics.* In this case, items are documents and the tests are applications of one-way hash functions with known expected values applied to selected collections of documents. The differences from the expected values are then used to identify which, if any, of the documents have been altered.

The primary goal of a testing algorithm is to identify all defective items using as few tests as possible. That is, we wish to minimize the following function:

- $t(n, d)$: the number of tests needed to identify up to d defectives among n items.

This minimization may be subject to possibly additional constraints, as well. For example, we may wish to identify all the defective items in a single (*nonadaptive*)

*Received by the editors May 18, 2005; accepted for publication (in revised form) June 7, 2006; published electronically January 12, 2007. This work was supported in part by NSF grants CCR-0312760, CCR-0311720, CCR-0225642, and CCR-0098068. The results of this paper were announced in preliminary form in [9].

http://www.siam.org/journals/sicomp/36-5/63184.html

†Department of Computer Science, University of California, Irvine, Irvine, CA 92697-3425 (eppstein@ics.uci.edu, goodrich@ics.uci.edu, dan@ics.uci.edu).

round of testing, we may wish to do this in two (*partially-adaptive*) rounds, or we may wish to perform the tests sequentially one after the other in a *fully adaptive* fashion.

In this paper we are interested in efficient solutions to combinatorial group testing problems for realistic problem sizes, which could be applied to solve the motivating examples given above. That is, we wish for solutions that minimize $t(n, d)$ for practical values of n and d as well as asymptotically. Because of the inherent delays that are built into fully adaptive, sequential solutions, we are interested only in solutions that can be completed in one or two rounds. Moreover, we desire solutions that are efficient not only in terms of the total number of tests performed but also for the following measures:

- $A(n, t)$: the *analysis* time needed to determine which items are defective.
- $S(n, d)$: the *sampling* rate—the maximum number of tests any item may be included in.

An analysis algorithm is said to be *efficient* if $A(n, t)$ is $O(tn)$, where n is the number of items and t is the number of tests conducted. It is *time-optimal* if $A(n, t)$ is $O(t)$. Likewise, we desire efficient sampling rates for our algorithms; that is, we desire that $S(n, d)$ be $O(t(n, d)/d)$. Moreover, we are interested in this paper in solutions that improve previous results, either asymptotically or by constant factors, for realistic problem sizes. We do not define such "realistic" problem sizes formally, but we may wish to consider as unrealistic a problem that is larger than the total memory capacity (in bytes) of all CDs and DVDs in the world ($< 10^{25}$), the number of atomic particles in the earth ($< 10^{50}$), or the number of atomic particles in the universe ($< 10^{80}$).

Viewing testing regimens as matrices. A single round in a combinatorial group testing algorithm consists of a test regimen and an analysis algorithm (which, in a nonadaptive (one-stage) algorithm, must identify all the defectives). The test regimen can be modeled by a $t \times n$ Boolean matrix, M. Each of the n columns of M corresponds to one of the n items. Each of the t rows of M represents a test of items whose corresponding column has a 1-entry in that row. All tests are conducted before the results of any test are made available. The analysis algorithm uses the results of the t tests to determine which of the n items are defective.

As described by Du and Hwang [6, p. 133], the matrix M is *d-disjunct* if the Boolean sum of any d columns does not contain any other column. In the analysis of a *d-disjunct* testing algorithm, items included in a test with negative outcome can be identified as pure. Using a d-disjunct matrix enables the conclusion that if there are d or fewer items that cannot be identified as pure in this manner, then all those items must be defective and there are no other defective items. If more than d items remain, then at least $d + 1$ of them are defective. Thus, using a d-disjunct matrix enables an efficient analysis algorithm, with $A(n, t)$ being $O(tn)$.

M is *d-separable* (*$\overline{d}$-separable*) if the Boolean sums of d (up to d) columns are all distinct. The $\overline{d}$-separable property implies that each selection of up to d defective items induces a different set of tests with positive outcomes. Thus, it is possible to identify which are the up to d defective items by checking, for each possible selection, whether its induced positive test set is exactly the obtained positive outcomes. However, it might not be possible to detect that there are more than d defective items. This analysis algorithm takes time $\Theta(n^d)$ or requires a large table mapping t-subsets to d-subsets.

Generally, $\overline{d}$-separable matrices can be constructed with fewer rows than can d-disjunct matrices having the same number of columns. Although the analysis algorithm described above for d-separable matrices is not efficient, some $\overline{d}$-separable matrices that are not d-disjunct have an efficient analysis algorithm.

Previous related work. Combinatorial group testing is a rich research area with many applications to many other areas, including communications, cryptography, and networking [3]. For an excellent discussion of this topic, the reader is referred to the book by Du and Hwang [6]. For general d, Du and Hwang [6, p. 149] describe a slight modification of the analysis of a construction due to Hwang and Sós [11] that results in a $t \times n$ d-disjunct matrix, with $n \geq (2/3)3^{t/16d^2}$, and so $t \leq 16d^2(1 + \log_3 2 + (\log_3 2)\lg n)$. For two-stage testing, De Bonis, Gasieniec, and Vaccaro [5] provide a scheme that achieves a number of tests within a factor of $7.54(1 + o(1))$ of the information-theoretic lower bound of $d\log(n/d)$. For $d = 2$, Kautz and Singleton [12] construct a 2-disjunct matrix with $t = 3^{q+1}$ and $n = 3^{2^q}$, for any positive integer q. Macula and Reuter [13] describe a $\overline{2}$-separable matrix and a time-optimal analysis algorithm with $t = (q^2 + 3q)/2$ and $n = 2^q - 1$, for any positive integer q. For $d = 3$, Du and Hwang [6, p. 159] describe the construction of a $\overline{3}$-separable matrix (but do not describe the analysis algorithm) with $t = 4\binom{3q}{2} = 18q^2 - 6q$ and $n = 2^q - 1$, for any positive integer q.

Our results. In this paper, we consider problems of identifying defectives using nonadaptive or two-stage protocols with efficient analysis algorithms. We present several such algorithms that require fewer tests than do previous algorithms for practical-sized sets, although we omit the proofs of some supporting lemmas in this paper, due to space constraints. Our general case algorithm, which is based on a method we call the Chinese remainder sieve, improves the construction of Hwang and Sós [11] for all values of d for real-world problem instances as well as for $d \geq n^{1/5}$ and $n \geq e^{10}$. Our two-stage algorithm achieves a bound for $t(n, d)$ that is within a factor of $4(1+o(1))$ of the information-theoretic lower bound. This bound improves the bound achieved by De Bonis, Gasieniec, and Vaccaro [5] by almost a factor of 2. Likewise, our algorithm for $d = 2$ improves on the number of tests required for all real-world problem sizes and is time-optimal (that is, with $A(n, t) \in O(t)$). Our algorithm for $d = 3$ is the first known time-optimal testing algorithm for that d-value. Moreover, our algorithms all have efficient sampling rates.

2. The Chinese remainder sieve. In this section, we present a solution to the problem for determining which items are defective when we know that there are at most $d < n$ defectives. Using a simple number-theoretic method, which we call the *Chinese remainder sieve* method, we describe the construction of a d-disjunct matrix with $t = O(d^2 \log^2 n/(\log d + \log\log n))$. As we will show, our bound is superior to that of the method of Hwang and Sós [11] for all realistic instances of the combinatorial group testing problem.

Suppose we are given n items, numbered $0, 1, \ldots, n-1$, such that at most $d < n$ are defective. Let $\{p_1^{e_1}, p_2^{e_2}, \ldots, p_k^{e_k}\}$ be a sequence of powers of distinct primes, multiplying to at least n^d. That is, $\prod_j p_j^{e_j} \geq n^d$. We construct a $t \times n$ matrix M as the vertical concatenation of k submatrices, $M_1, M_2, \ldots, M_k$. Each submatrix M_j is a $t_j \times n$ testing matrix, where $t_j = p_j^{e_j}$; hence, $t = \sum_{j=1}^{k} p_j^{e_j}$. We form each row of M_j by associating it with a nonnegative value x less than $p_j^{e_j}$. Specifically, for each x, $0 \leq x < p_j^{e_j}$, we form a test in M_j consisting of the item indices (in the range $0, 1, \ldots, n-1$) that equal $x \pmod{p_j^{e_j}}$. For example, if $x = 2$ and $p_j^{e_j} = 3^2$, then the row for x in M_j has a 1 only in columns 2, 11, 20, and so on.

The following lemma shows that the test matrix M is d-disjunct.

LEMMA 1. *If there are at most d defective items, and all tests in M are positive for i, then i is defective.*

Proof. If all k tests for i (one for each prime power $p_j^{e_j}$) are positive, then there exists at least one defective item. With each positive test that includes i (that is, it has a 1 in column i), let $p_j^{e_j}$ be the modulus used for this test, and associate with j a defective index i_j that was included in that test (choosing i_j arbitrarily in case test j includes multiple defective indices). For any defective index i', let

$$P_{i'} = \prod_{j \text{ s.t. } i_j = i'} p_j^{e_j}.$$

That is, $P_{i'}$ is the product of all the prime powers such that i' caused a positive test that included i for that prime power. Since there are k tests that are positive for i, each $p_j^{e_j}$ appears in exactly one of these products, $P_{i'}$. So $\prod P_{i'} = \prod p_j^{e_j} \geq n^d$. Moreover, there are at most d products, $P_{i'}$. Therefore, $\max_{i'} P_{i'} \geq (n^d)^{1/d} = n$; hence, there exists at least one defective index i' for which $P_{i'} \geq n$. By construction, i' is congruent to the same values to which i is congruent, modulo each of the prime powers in $P_{i'}$. By the Chinese remainder theorem, the solution to these common congruences is unique modulo the least common multiple of these prime powers, which is $P_{i'}$ itself. Therefore, i is equal to i' modulo a number that is at least n, and so $i = i'$; hence, i is defective. □

The important role of the Chinese remainder theorem in the proof of the above lemma gives rise to our name for this construction—the Chinese remainder sieve.

Analysis. As mentioned above, the total number of tests, $t(n, d)$, constructed in the Chinese remainder sieve is $\sum_{j=1}^{k} p_j^{e_j}$, where $\prod p_j^{e_j} \geq n^d$. If we let each $e_j = 1$, we can simplify our analysis to note that $t(n, d) = \sum_{j=1}^{k} p_j$, where p_j denotes the jth prime number and k is chosen so that $\prod_{j=1}^{k} p_j \geq n^d$. To produce a closed-form upper bound for $t(n, d)$, we make use of the prime counting function, $\pi(x)$, which is the number of primes less than or equal to x. We also use the well-known *Chebyshev function*, $\theta(x) = \sum_{j=1}^{\pi(x)} \ln p_j$. In addition, we make use of the following (less well-known) prime summation function, $\sigma(x) = \sum_{j=1}^{\pi(x)} p_j$. Using these functions, we bound the number of tests in the Chinese remainder sieve method as $t(n, d) \leq \sigma(x)$, where x is chosen so that $\theta(x) \geq d \ln n$, since $\ln \prod_{p_j \leq x} p_j = \theta(x)$. For the Chebyshev function, it can be shown [1] that $\theta(x) \geq x/2$ for $x > 4$ and that $\theta(x) \sim x$ for large x. So if we let $x = \lceil 2d \ln n \rceil$, then $\theta(x) \geq d \ln n$. Thus, we can bound the number of tests in our method as $t(n, d) \leq \sigma(\lceil 2d \ln n \rceil)$. To further bound $t(n, d)$, we use the following lemma, which may be of mild independent interest.

LEMMA 2. *For integer $x \geq 2$,*

$$\sigma(x) < \frac{x^2}{2 \ln x} \left(1 + \frac{1.2762}{\ln x}\right).$$

Proof. Let $n = \pi(x)$. Dusart [7, 8] shows that, for $n \geq 799$,

$$\frac{1}{n} \sum_{j=1}^{n} p_j < \frac{1}{2} p_n;$$

that is, the average of the first n primes is half the value of the nth prime. Thus,

$$\sigma(x) = \sum_{j=1}^{\pi(x)} p_j < \frac{\pi(x)}{2} p_n \leq \frac{\pi(x)}{2} x$$

for integer $x \geq 6131$ (the 799th prime). Dusart [7, 8] also shows that

$$\pi(x) < \frac{x}{\ln x}\left(1 + \frac{1.2762}{\ln x}\right)$$

for $x \geq 2$. Therefore, for integer $x \geq 6131$,

$$\sigma(x) < \frac{x^2}{\ln x}\left(1 + \frac{1.2762}{\ln x}\right).$$

In addition, we have verified by an exhaustive computer search that this inequality also holds for all integers $2 \leq x < 6131$. This completes the proof. □

Thus, we can characterize the Chinese remainder sieve method as follows.

THEOREM 1. *Given a set of n items, at most d of which are defective, the Chinese remainder sieve method can identify the defective items using a number of tests*

$$t(n,d) < \frac{\lceil 2d \ln n \rceil^2}{2 \ln \lceil 2d \ln n \rceil}\left(1 + \frac{1.2762}{\ln \lceil 2d \ln n \rceil}\right).$$

The sample rate can be bounded by

$$S(n,d) < \frac{\lceil 2d \ln n \rceil}{2 \ln \lceil 2d \ln n \rceil}\left(1 + \frac{1.2762}{\ln \lceil 2d \ln n \rceil}\right),$$

and the analysis time, $A(n,t)$, is $O(nt(n,d))$.

By calculating the exact numbers of tests required by the Chinese remainder sieve method for particular parameter values and comparing these numbers to the claimed bounds for Hwang and Sós [11], we see that our algorithm is an improvement when

- $d = 2$ and $n \leq 10^{57}$
- $d = 3$ and $n \leq 10^{66}$
- $d = 4$ and $n \leq 10^{70}$
- $d = 5$ and $n \leq 10^{74}$
- $d = 6$ and $n \leq 10^{77}$
- $d \geq 7$ and $n \leq 10^{80}$.

Of course, these are the most likely cases for any expected actual instance of the combinatorial group testing problem. In addition, our analysis shows that our method is superior to the claimed bounds of Hwang and Sós [11] for $d \geq n^{1/5}$ and $n \geq e^{10}$. Less precisely, we can say that $t(n,d)$ is $O(d^2 \log^2 n/(\log d + \log\log n))$, that $S(n,d)$ is $O(d \log n/(\log d + \log\log n)$, and that $A(n,t)$ is $O(tn)$, which is $O(d^2 n \log^2 n/(\log d + \log\log n))$.

Heuristic improvements. Although it will not reduce the asymptotic complexity of t, we can reduce the number of tests by starting with a sequence of primes up to some upper bound x and efficiently constructing a set of good prime powers from this sequence. We can allow some powers, e_j, to be zero (meaning that we do not use this prime), while giving others values greater than one. The objective is to choose carefully the values e_j in order to minimize the number of tests while maintaining the property that $\prod p_j^{e_j} \geq n^d$. This typically yields a savings of between five and ten percent.

An example implementation in Python 2.3 is shown in the appendix in Figures A.1 and A.2. This implementation starts with the $e_j = 1$ solution to determine an initial suitable sequence of primes, p_j, to use. It then does a backtracking search to find the optimal set of e_j for these p_j, subject to the constraint that each $p_j^{e_j}$ is not greater than the largest prime in the original solution (with each $e_j = 1$). Since the number of e_j powers is sublogarithmic, and most of them must be 0 or 1, this backtracking search takes time sublinear in n for fixed d.

TABLE 2.1
Comparing $t(n)$ for $d = 5$ and $d = 10$.

$(d = 5)$	100	10^4	10^6	10^8	10^{10}	10^{20}	10^{30}
Our bktrk	131	378	738	1176	1709	5737	11782
Our genl	160	440	791	1264	1851	6081	12339
HS	2329	4006	5683	7359	9036	17420	25803
$(d = 10)$	100	10^4	10^6	10^8	10^{10}	10^{20}	10^{30}
Our bktrk	378	1176	2350	3896	5737	19681	41020
Our genl	440	1264	2584	4227	6081	20546	42468
HS	9316	16023	22730	29437	36144	69678	103213

Comparison of the number of tests required. Table 2.1 lists the number of tests required by the Hwang–Sós (HS) algorithm, our general algorithm (using the initial set of primes p_j having exponents $e_j = 1$), and our improved backtrack algorithm, for some values of n. As can be seen, for moderate values of n our algorithms require a small fraction of the number of tests required by the HS algorithm. However, asymptotically for fixed d, the HS algorithm requires fewer tests.

3. A two-stage rake-and-winnow protocol. In this section, we present a randomized construction for two-stage group testing. This two-stage method uses a number of tests within a constant factor of the information-theoretic lower bound. It improves previous upper bounds [5] by almost a factor of 2. In addition, it has an efficient sampling rate, with $S(n, d)$ being only $O(\log(n/d))$. All the constant factors "hiding" behind the big-ohs in these bounds are small.

Preliminaries. One of the important tools we use in our analysis is the following lemma for bounding the tail of a certain distribution. It is a form of Chernoff bound [14].

LEMMA 3. *Let X be the sum of n independent indicator random variables, such that $X = \sum_{i=1}^{n} X_i$, where each $X_i = 1$ with probability p_i, for $i = 1, 2, \ldots, n$. If $E[X] = \sum_{i=1}^{n} p_i \le \hat{\mu} < 1$, then, for any integer $k > 0$,*

$$\Pr(X \ge k) \le \left(\frac{e\hat{\mu}}{k}\right)^k.$$

Proof. Let $\mu = E[X]$ be the actual expected value of X. Then, by a well-known Chernoff bound [14], for any $\delta > 0$,

$$\Pr[X \ge (1+\delta)\mu] \le \left[\frac{e^\delta}{(1+\delta)^{1+\delta}}\right]^\mu.$$

(The bound in [14] is for strict inequality, but the same bound holds for nonstrict inequality.) We are interested in the case when $(1+\delta)\mu = k$, that is, when $1+\delta = k/\mu$. Observing that $\delta < 1 + \delta$, we can therefore deduce that

$$\Pr(X \ge k) \le \left[\frac{e^{k/\mu}}{(k/\mu)^{k/\mu}}\right]^\mu = \frac{e^k}{(k/\mu)^k} = \left(\frac{e\mu}{k}\right)^k.$$

Finally, noting that $\mu \le \hat{\mu}$,

$$\Pr(X \ge k) \le \left(\frac{e\hat{\mu}}{k}\right)^k. \qquad \square$$

In addition to this lemma, we also use the following.

LEMMA 4. *If $d < n$, then*

$$\binom{n}{d} < \left(\frac{en}{d}\right)^d.$$

Proof.

$$\begin{aligned}\binom{n}{d} &= \frac{n!}{(n-d)!\,d!} \\ &= \frac{n(n-1)(n-2)\cdots(n-d+1)}{d!} \\ &< \frac{n^d}{d!}.\end{aligned}$$

By Stirling's approximation [4],

$$d! = \sqrt{2\pi n}\left(\frac{d}{e}\right)^d\left(1+\theta\left(\frac{1}{n}\right)\right).$$

Thus, $d! > (d/e)^d$. Therefore,

$$\frac{n^d}{d!} < \frac{n^d}{(d/e)^d} = \left(\frac{en}{d}\right)^d. \qquad \square$$

Identifying defective items in two stages. As with our Chinese remainder sieve method, our randomized combinatorial group testing construction is based on the use of a Boolean matrix M where columns correspond to items and rows correspond to tests, so that if $M[i,j] = 1$, then item j is included in test j. Let C denote the set of columns of M. Given a set D of d columns in M, and a specific column $j \in C - D$, we say that j is *distinguishable* from D if there is a row i of M such that $M[i,j] = 1$ but i contains a 0 in each of the columns in D. Such a property is useful in the context of group testing, for the set D could correspond to the defective items, and if a column j is distinguishable from the set D, then there would be a test in our regimen that would determine that the item corresponding to column j is not defective.

An alternate and equivalent definition [6, p. 165] for a matrix M to be *d-disjunct* is if, for any d-sized subset D of C, each column in $C - D$ is distinguishable from D. Such a matrix determines a powerful group testing regimen, but, unfortunately, building such a matrix requires M to have $\Omega(d^2 \log n / \log d)$ rows, by a result of Ruszinkó [15] (see also [6, p. 139]). The best known constructions have $\Theta(d^2 \log(n/d))$ rows [6], which is a factor of d greater than information-theoretic lower bound, which is $\Omega(d \log(n/d))$.

Instead of trying to use a matrix M to determine all the defectives immediately, we will settle for a weaker property for M, which nevertheless is still powerful enough to define a good group testing regimen. We say that M is *(d,k)-resolvable* if, for any d-sized subset D of C, there are fewer than k columns in $C - D$ that are not distinguishable from D. Such a matrix defines a powerful group testing regimen, for defining tests according to the rows of a d-resolvable matrix allows us to restrict the set of defective items to a group D' of smaller than $d + k$ size. Given this set, we can then perform an additional round of individual tests on all the items in D'. This two-stage approach is sometimes called the trivial two-stage algorithm; we refer to this two-stage algorithm as the *rake-and-winnow* approach.

Thus, a (d, k)-resolvable matrix determines a powerful group testing regimen. Of course, a matrix is d-disjunct iff it is $(d, 1)$-resolvable. Unfortunately, as mentioned above, constructing a $(d, 1)$-resolvable matrix requires that the number of rows (which correspond to tests) be significantly greater than the information-theoretical lower bound. Nevertheless, if we are willing to use a (d, k)-resolvable matrix, for a reasonably small value of k, we can come within a constant factor of the information-theoretical lower bound.

Our construction of a (d, k)-resolvable matrix is based on a simple, randomized *sample-injection* strategy, which itself is based on the approach popularized by the Bloom filter [2]. This novel approach also allows us to provide a strong worst-case bound for the sample rate, $S(n, d)$, of our method. Given a parameter t, which is a multiple of d that will be set in the analysis, we construct a $2t \times n$ matrix M in a columnwise fashion. For each column j of M, we choose t/d rows at random and set the values of these entries to 1. The other entries in column j are set to 0. In other words, we "inject" the sample j into each of the t/d random tests we pick for the corresponding column (since rows of M correspond to tests and the columns correspond to samples). Note, then, that for any set of d defective samples, there are at most t tests that will have positive outcomes and, therefore, at least t tests that will have negative outcomes. The columns that correspond to samples that are distinguishable from the defectives ones can be immediately identified. The remaining issue, then, is to determine the value of t needed so that, for a given value of k, M is a (d, k)-resolvable matrix with high probability.

Let D be a fixed set of d defectives samples. For each (column) item i in $C - D$, let X_i denote the indicator random variable that is 1 if i is falsely identified as a positive sample by M (that is, i is not included in the set of (negative) items distinguished from those in D), and is 0 otherwise. Observe that the X_i's are independent, since X_i depends only on whether the choice of rows we picked for column i collide with the at most t rows of M that we picked for the columns corresponding to items in D. Furthermore, this observation implies that any X_i is 1 (a false positive) with probability at most $2^{-t/d}$. Therefore, the expected value of X, $E[X]$, is at most $\hat{\mu} = n/2^{t/d}$. This fact allows us to apply Lemma 3 to bound the probability that M does not satisfy the (d, k)-resolvable property for this particular choice, D, of d defective samples. In particular,

$$\Pr(X \geq k) \leq \left(\frac{e\hat{\mu}}{k}\right)^k = \frac{\left(\frac{en}{k}\right)^k}{2^{(t/d)k}}.$$

Note that this bound immediately implies that if $k = 1$ and $t \geq d(e+1)\log n$, then M will be completely $(d, 1)$-resolvable with high probability $(1 - 1/n)$ for any particular set of defective items, D.

We are interested, however, in a bound implying that for *any* subset D of d defectives (of which there are $\binom{n}{d} < (en/d)^d$, by Lemma 4), our matrix M is (d, k)-resolvable with high probability, that is, probability at least $1 - 1/n$. That is, we are interested in the value of t such that the above probability bound is $(en/d)^{-d}/n$. From the above probability bound, therefore, we are interested in a value of t such that

$$\frac{2^{(t/d)k}}{\left(\frac{en}{k}\right)^k} \geq \left(\frac{en}{d}\right)^d n.$$

That is, we would like

$$2^{(t/d)k} \geq \left(\frac{en}{d}\right)^d \left(\frac{en}{k}\right)^k n.$$

This bound will hold whenever

$$t \geq (d^2/k)\log(en/d) + d\log(en/k) + (d/k)\log n.$$

Thus, we have the following.

THEOREM 2. *If* $t \geq (d^2/k)\log(en/d) + d\log(en/k) + (d/k)\log n$, *then a* $2t \times n$ *random matrix* M *constructed by sample-injection is* (d,k)*-resolvable with high probability, that is, with probability at least* $1 - 1/n$.

Taking $k = 1$, therefore, we have an alternative method for constructing a d-disjunct matrix M with high probability.

COROLLARY 1. *If* $t \geq d^2\log(en/d) + d\log en + d\log n$, *then a* $2t \times n$ *random matrix* M *constructed by sample-injection is* d*-disjunct with high probability.*

That is, we can construct a one-round group test based on sample-injection that uses $O(d^2\log(n/d))$ tests.

As mentioned above, a productive way of using the sample-injection construction is to build a (d,k)-resolvable matrix M for a reasonably small value of k. We can then use this matrix as the first round in a two-round rake-and-winnow testing strategy, where the second round simply involves our individual testing of the at most $d + k$ samples left as potential positive samples from the first round.

COROLLARY 2. *If* $t \geq 2d\log(en/d) + \log n$, *then the* $2t \times n$ *random matrix* M *constructed by sample-injection is* (d,d)*-resolvable with high probability.*

This corollary implies that we can construct a rake-and-winnow algorithm where the first stage involves performing $O(d\log(n/d))$ tests, which is within a (small) constant factor of the information theoretic lower bound, and the second round involves individually testing at most $2d$ samples.

4. Improved bounds for small *d* values. In this section, we consider efficient algorithms for the special cases when $d = 2$ and $d = 3$. We present time-optimal algorithms for these cases, that is, with $A(n,t)$ being $O(t)$. Our algorithm for $d = 3$ is the first known such algorithm.

Finding up to two defectives. Consider the problem of determining which items are defective when we know that there are at most two defectives. We describe a $\overline{2}$-separable matrix and a time-optimal analysis algorithm with $t = (q^2 + 5q)/2$ and $n = 3^q$, for any positive integer q.

Let the number of items be $n = 3^q$, and let the item indices be expressed in radix 3. Index $X = X_{q-1}\cdots X_0$, where each digit $X_p \in \{0,1,2\}$.

Hereafter, X ranges over the item index numbers $\{0,\ldots,n-1\}$, p ranges over the radix positions $\{0,\ldots,q-1\}$, and v ranges over the digit values $\{0,1,2\}$.

For our construction, matrix M is partitioned into submatrices B and C. Matrix B is the submatrix of M consisting of its first $3q$ rows. Row $\langle p,v\rangle$ of B is associated with radix position p and value v. $B[\langle p,v\rangle, X] = 1$ iff $X_p = v$.

Matrix C is the submatrix of M consisting of its last $\binom{q}{2}$ rows. Row $\langle p,p'\rangle$ of C is associated with distinct radix positions p and p', where $p < p'$. $C[\langle p,p'\rangle, X] = 1$ iff $X_p = X_{p'}$.

Let $test_B(p,v)$ be the result (1 for positive, 0 for negative) of the test of items having a 1-entry in row $\langle p,v\rangle$ in B. Similarly, let $test_C(p,p')$ be the result of testing row $\langle p,p'\rangle$ in C. Let $test1(p)$ be the number of different values held by defectives in radix position p. $test1(p)$ can be computed by $test_B(p,0) + test_B(p,1) + test_B(p,2)$.

The analysis algorithm is shown in the appendix in Figure A.3.

It is easy to determine how many defective items are present. There are no defective items when $test1(0) = 0$. There is only one defective item when $test1(p) = 1$

for all p, since if there were two defective items, then there must be at least one position p in which their indices differ and $test1(p)$ would then have value 2. The one defective item has index $D = D_{q-1} \cdots D_0$, where digit D_p is the value v for which $test_B(p, v) = 1$.

Otherwise, there must be two defective items, $D = D_{q-1} \cdots D_0$ and $E = E_{q-1} \cdots E_0$. We iteratively determine the values of the digits of indices D and E.

For radix positions in which defective items exist for only one value of that digit, both D and E must have that value for that digit. For each other radix position, two distinct values for that digit occur in the defective items.

The first radix position in which D and E differ is recorded in the variable p^* and the value of that digit in D (respectively, E) is recorded in v_1^* (respectively, v_2^*).

For any subsequent position p in which D and E differ, the digit values of the defectives in that position are v_a and v_b, which are two distinct values from $\{0, 1, 2\}$, as are v_1^* and v_2^*, and therefore there must be at least one value in common between $\{v_a, v_b\}$ and $\{v_1^*, v_2^*\}$.

Let a common value be v_a and, without loss of generality, let $v_a = v_1^*$.

LEMMA 5. *The digit assignment for position p is $D_p = v_a$ and $E_p = v_b$ iff $test_C(p^*, p) = 1$.*

Proof. We consider the two possibilities of which defective item has v_a as its digit in position p.

Case 1. $D_p = v_a$.

We see that $D_p = v_a = v_1^*$. Accordingly, a defective (D) would be among the items tested in $test_C(p^*, p)$. Therefore, $test_C(p^*, p) = 1$.

Case 2. $E_p = v_a$.

We see that $D_p \neq v_1^*$, because $D_p \neq E_p = v_a = v_1^*$, and also that $E_p \neq v_2^*$, because $E_p = v_a = v_1^* \neq v_2^*$. Accordingly, neither of the defective items would be among the items tested in $test_C(p^*, p)$. Therefore, $test_C(p^*, p) = 0$. □

We have determined the values of defectives D and E for all positions—those where they are the same and those where they differ. For each position, only a constant amount of work is required to determine the assignment of digit values. Therefore, we have proven the following theorem.

THEOREM 3. *A $\overline{2}$-separable matrix that has a time-optimal analysis algorithm can be constructed with $t = (q^2 + 5q)/2$ and $n = 3^q$, for any positive integer q.*

Comparison of the number of tests required for the $d = 2$ method. A $\overline{2}$-separable or a 2-disjunct $t \times n$ matrix enables determination of up to two defective items from among n *or fewer* items using t tests. An algorithm is more competitive at or just below one of its breakpoints, values of n for which increasing n by one significantly increases t. The Macula–Reuter (MR) algorithm has breakpoints at one under all powers of 2, our (d=2) algorithm at all powers of 3, and the Kautz–Singleton (KS) algorithm at only certain powers of 3. Our general-d algorithms do not have significant breakpoints.

Table 4.1 lists the number of tests required by these algorithms for some small values of n. For all $n \leq 3^{63}$, our $d = 2$ algorithm uses the smallest number of tests. For higher values of $n \leq 3^{130}$, the KS and our $d = 2$ and general (Chinese remainder sieve) algorithms alternate being dominant. The alternations are illustrated in Table 4.2. For all $n \geq 3^{131}$, the HS algorithm uses the fewest tests.

Finding up to three defectives. Consider the problem of determining which items are defective when we know that there are at most three defectives. We describe a $\overline{3}$-separable matrix and a time-optimal analysis algorithm with $t = 2q^2 - 2q$ and $n = 2^q$, for any positive integer q.

TABLE 4.1
$t(n)$ for small n ($d = 2$).

$(d = 2)$	15	100	10^3	10^4	10^5	10^6	10^8	10^{10}	10^{20}	10^{30}
Our $d = 2$	12	25	42	63	88	117	187	273	987	2142
Our bktrk	19	36	60	89	131	168	268	378	1176	2350
Our genl	28	41	77	100	160	197	281	440	1264	2584
MR	14	35	65	119	170	230	405	629	2345	5150
KS	27	81	81	243	243	243	729	729	2187	2187
HS		373	507	641	775	909	1177	1446	2787	4129

TABLE 4.2
$t(n)$ for large n ($d = 2$).

$(d = 2)$	3^{63}	3^{64}	3^{104}	3^{112}	3^{128}	3^{130}	3^{256}
Our $d = 2$	**2142**	2208	**5668**	6552	8512	8775	33408
Our bktrk	2366	2424	5687	**6454**	8184	**8394**	28311
Our genl	2584	2584	6081	6870	8582	8893	29296
KS	2187	**2187**	6561	6561	**6561**	19683	19683
HS	4136	4200	6760	7272	8296	8424	**16488**

Let the number of items be $n = 2^q$, and let the item indices be expressed in radix 2. Index $X = X_{q-1} \cdots X_0$, where each digit $X_p \in \{0, 1\}$.

Hereafter, X ranges over the item index numbers $\{0, \ldots, n-1\}$, p ranges over the radix positions $\{0, \ldots, q-1\}$, and v ranges over the digit values $\{0, 1\}$.

Matrix M has $2q^2 - 2q$ rows. Row $\langle p, p', v, v' \rangle$ of M is associated with distinct radix positions p and p', where $p < p'$, and with values v and v', each of which is in {0,1}. $M[\langle p, p', v, v' \rangle, X] = 1$ iff $X_p = v$ and $X_{p'} = v'$.

Let $test_M(p, p', v, v')$ be the result (1 for positive, 0 for negative) of testing items having a 1-entry in row $\langle p, p', v, v' \rangle$ in M. For $p' > p$, define $test_M(p', p, v', v) = test_M(p, p', v, v')$.

The following three functions can be computed in terms of $test_M$.

- $test_B(p, v)$ has value 1 (0) if there are (not) any defectives having value v in radix position p, i.e., $test_B(0, v) = 0$ if $test_M(0, 1, v, 0) + test_M(0, 1, v, 1) = 0$, and 1 otherwise. For $p > 0$, $test_B(p, v) = 0$ if

$$test_M(p, 0, v, 0) + test_M(p, 0, v, 1) = 0,$$

  and 1 otherwise.
- $test1(p)$ is the number of different binary values held by defectives in radix position p. Thus, $test1(p) = test_B(p, 0) + test_B(p, 1)$.
- $test2(p, p')$ is the number of different ordered pairs of binary values held by defectives in the designated ordered pair of radix positions. Therefore,

$$test2(p, p') = test_M(p, p', 0, 0) + test_M(p, p', 0, 1) + test_M(p, p', 1, 0) + test_M(p, p', 1, 1).$$

The analysis algorithm is shown in the appendix in Figure A.4.

We determine the number of defective items and the value of their digits. There are no defective items when $test1(0) = 0$. Moreover, at each radix position p in which $test1(p) = 1$, all defective items have the same value of that digit. If all defectives agree on all digit values, then there is only one defective. Otherwise there are at least two defectives, and we need to consider how to assign digit values for only the set of

positions P in which there is at least one defective having each of the two possible binary digit values.

LEMMA 6. *There are only two defectives iff, for* $p, p' \in P, test2(p, p') = 2$.

Proof. A defective item can contribute at most one new combination of values in positions p, p', and so $test2(p, p') \leq$ the number of defectives. Accordingly, if there are fewer than two defectives, then $test2(p, p') < 2$.

If there are exactly two defectives, then $test2(p, p') \leq 2$. Since $p \in P$, both binary values appear among defectives, and so $test2(p, p') \geq 2$, and therefore $test2(p, p') = 2$.

Consider the case in which there are three defectives. In any position p_1 in which both binary values appear at that digit among the set of defectives, one of the defectives (say, D) has one binary value (say, v_1) and the other two defectives (E, F) have the other binary value $(\overline{v}_1)$. Since E and F are distinct, they must differ in value at some other position p_2. Therefore, there will be three different ordered pairs of binary values held by defectives in positions p_1 and p_2, and so $test2(p_1, p_2) = 3$. □

Accordingly, if there is no pair of positions for which $test2$ has value 3, we can conclude that there are only two defectives. Otherwise, there are positions p_1, p_2 for which $test2(p_1, p_2) = 3$, and one of the four combinations of two binary values will not appear. Let that missing combination be v_1, v_2. Thus, while position p_1 uniquely identifies one defective, say D, as the only defective having value v_1 at that position, position p_2 uniquely identifies one of the other defectives, say E, as having value v_2.

LEMMA 7. *If the position* p^* *uniquely identifies the defective* X *to have value* v^*, *then the value of the defective* X *at any other position* p *will be that value* v *such that* $test_M(p^*, p, v^*, v) = 1$.

Proof. If position p^* uniquely identifies defective X as having value v^*, then $X_{p^*} = v^*$ and, for any other defective Y, $Y_{p^*} \neq v^*$.

Let $v = X_p$ for any $p \neq p^*$. Then $test_M(p^*, p, v^*, v) = 1$, since X is a defective that has the required values at the required positions to be included in this test.

Also, $test_M(p^*, p, v^*, \overline{v}) = 0$, because none of the defectives is included in this test. Defective X is not included, because $X_p \neq \overline{v}$. Any other defective, $Y \neq X$, is not included, because $Y_{p^*} \neq v^*$. □

Since we have positions that uniquely identify D and E, we can determine the values of all their other digits, and the only remaining problem is to determine the values of the digits of defective F.

Since position p_1 uniquely identifies D, we know that $F_{p_1} = \overline{v}_1$. For any other position p, after determining that $E_p = v$, we note that if $test_M(p_1, p, \overline{v}_1, \overline{v}) = 1$, then there must be at least one defective, X, for which $X_{p_1} = \overline{v}_1$ and $X_p = \overline{v}$. Defective D is ruled out, since $D_{p_1} = v_1$, and defective E is ruled out, since $E_p = v$. Therefore, it must be that $F_p = \overline{v}$. Otherwise, if that $test_M = 0$, then $F_p = v$, since $F_p = \overline{v}$ would have caused $test_M = 1$.

We have determined the values of defectives D, E, and F for all positions. For each position, only a constant amount of work is required to determine the assignment of digit values. Therefore, we have proven the following theorem.

THEOREM 4. *A* $\overline{3}$*-separable matrix that has a time-optimal analysis algorithm can be constructed with* $t = 2q^2 - 2q$ *and* $n = 2^q$, *for any positive integer* q.

Comparison of the number of tests required for the $d = 3$ *method.* The general d algorithm due to Hwang and Sós [11] requires fewer tests than does the Du–Hwang (DH) algorithm for $d = 3$ suggested in [6]. For $n < 10^{10}$, our $(d = 3)$ algorithm requires even fewer tests and our general (Chinese remainder sieve) algorithm the fewest. However, asymptotically the algorithm of Hwang and Sós uses the fewest

TABLE 4.3
Comparing $t(n)$ for $d = 3$.

$(d = 3)$	100	10^4	10^6	10^8	10^{10}	10^{20}	10^{30}
Our bktrk	60	168	321	513	738	2350	4777
Our genl	77	197	381	568	791	2584	5117
Our $d = 3$	84	364	760	1404	2244	8844	19800
HS	838	1442	2046	2649	3253	6271	9289
DH	840	3444	7080	12960	20604	80400	179400

tests. We note that, unlike these other efficient algorithms, our $(d = 3)$ algorithm is time-optimal. Table 4.3 lists the number of tests required by these algorithms for some small values of n.

Appendix A. Pseudocode listings.

```
def eratosthenes():
    """Generate the sequence of prime numbers via the sieve of Eratosthenes."""
    D = {} # map composite integers to primes witnessing their compositeness
    q = 2  # first integer to test for primality
    while True:
        if q not in D:
            yield q          # not marked composite, must be prime
            D[q*q] = [q]     # first multiple of q not already marked
        else:
            for p in D[q]: # move each witness to its next multiple
                D.setdefault(p+q,[]).append(p)
            del D[q]         # no longer need D[q], free memory
        q += 1

def search(primes,maxpow,target):
    """
    Backtracking search for exponents of prime powers, each at most maxpow,
    so that the product of the powers is at least target and the sum of the
    nonunit powers is minimized. Returns the pair [sum,list of exponents].
    """
    if target <= 1:          # all unit powers will work?
        return [0,[0]*len(primes)]
    elif not primes or maxpow**len(primes) < target:
        return None          # no primes supplied, no solution exists
    primes = list(primes)    # list all but the last prime for recursive calls
    p = primes.pop()
    best = None              # no solution found yet
    i = 0
    while p**i <= maxpow: # loop through possible exponents of p
        s = search(primes,maxpow,(target + p**i - 1)//p**i)
        if s is not None:
            s[0] += i and p**i
            s[1].append(i)
            best = min(best,s) or s
        i += 1
    return best
```

FIG. A.1. *Subroutines for construction based on prime factorization.*

```
def prime_cgt(n,d):
    """Find a CGT for n and d and output a description of it to stdout."""

    # collect primes until their total product is large enough
    primes = []
    product = 1
    for p in eratosthenes():
        primes.append(p)
        product *= p
        if product > n**d:
            break

    # now find good collection of powers of those primes...
    result = search(primes,primes[-1],n**d)
    powers = result[1]

    # output results
    print "n =",n,"d =",d,":",
    for i in range(len(primes)):
        if powers[i] == 1:
            print primes[i],
        elif powers[i] > 1:
            print str(primes[i]) + "^" + str(powers[i]),
    print "total tests:", sum([primes[i]**powers[i] for i in range(len(primes))
                                    if powers[i]])

if __name__ == "__main__":
    for d in range(2,6):
        for x in range(6,16):
            prime_cgt(1<<x,d)
        print
```

FIG. A.2. *Construct tests based on prime factorization.*

```
    if test1(0) = 0 then return there are no defective items
    p* ← −1
    for p ← 0 to q − 1 do
        if test1(p) = 1 then
            D_p ← E_p ← the value v such that test_B(p, v) = 1
        else        // test1(p) has value 2
            Let v_1, v_2 be the two values of v such that test_B(p, v) = 1
            if p* < 0 then
                p* ← p
                v_1* ← D_p ← v_1
                v_2* ← E_p ← v_2
            else
                if test_C(p*, p) = 1 and ( v_1* = v_1 or v_2* = v_2 ) then
                    D_p ← v_1
                    E_p ← v_2
                else
                    D_p ← v_2
                    E_p ← v_1
    if p* < 0 then
        return there is one defective item D
    else
        return there are two defective items D and E
```

FIG. A.3. *Analysis algorithm for up to two defectives.*

```
if test1(0) = 0 then return there are no defective items
P ← ∅
for p ← 0 to q − 1 do
    if test1(p) = 1 then
        D_p ← E_p ← F_p ← the value v such that test_B(p, v) = 1
    else P ← P ∪ {p}
if P = ∅ then return there is one defective item D
if test2(p_1, p_2) = 2 for all p_1, p_2 ∈ P then
    p* ← −1
    for p ∈ P do
        if p* < 0 then
            p* ← p
            v* ← D_p ← 0
        else if test_M(p*, p, v*, 0) = 1 then
                D_p ← 0
            else D_p ← 1
        E_p ← 1 − D_p
    return there are two defective items D, E
else
    Let p_1, p_2 be positions such that test2(p_1, p_2) = 3
    Let v_1, v_2 be values such that test_M(p_1, p_2, v_1, v_2) = 0
    D_{p_1} ← v_1
    F_{p_1} ← E_{p_1} ← 1 − v_1
    E_{p_2} ← v_2
    F_{p_2} ← D_{p_2} ← 1 − v_2
    for p ∈ P − {p_1, p_2} do
        if test_M(p_1, p, v_1, 0) = 1 then
            D_p ← 0
        else D_p ← 1
        if test_M(p_2, p, v_2, 0) = 1 then
            E_p ← 0
        else E_p ← 1
        v ← E_p
        if test_M(p_1, p, 1 − v_1, 1 − v) = 1 then
            F_p ← 1 − v
        else F_p ← v
return there are three defective items D, E, and F
```

FIG. A.4. *Analysis algorithm for up to three defectives.*

Appendix B. We would like to thank George Lueker and Dennis Shasha for several helpful discussions related to the topics of this paper.

REFERENCES

[1] E. BACH AND J. SHALLIT, *Algorithmic Number Theory, Vol.* 1: *Efficient Algorithms*, MIT Press, Cambridge, MA, 1996.

[2] B. H. BLOOM, *Space/time trade-offs in hash coding with allowable errors*, Comm. ACM, 13 (1970), pp. 422–426.

[3] C. J. COLBOURN, J. H. DINITZ, AND D. R. STINSON, *Applications of combinatorial designs to communications, cryptography, and networking*, in Surveys in Combinatorics, 1993, London Math. Soc. Lecture Note Ser. 187, J. D. Lamb and D. A. Preece, eds., Cambridge University Press, Cambridge, UK, 1999, pp. 37–100..

[4] T. H. CORMEN, C. E. LEISERSON, AND R. L. RIVEST, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

[5] A. DE BONIS, L. GASIENIEC, AND U. VACCARO, *Generalized framework for selectors with applications in optimal group testing*, in Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP'03), Springer, Berlin, 2003, pp. 81–96.

[6] D.-Z. Du and F. K. Hwang, *Combinatorial Group Testing and Its Applications*, 2nd ed., World Scientific, River Edge, NJ, 2000.

[7] P. Dusart, *Encadrements Effectifs des Fonctions de Chebyshev (Sharper Bounds for ψ, θ, π, p_k)*, Rapport 1998-06, Laboratoire d'Arithmétique, de Calcul formel et d'Optimisation, Limoges, Cedex, France, http://www.unilim.fr/laco/rapports/1998/R1998_06.pdf (1998).

[8] P. Dusart, *The kth prime is greater than $k(\ln k + \ln\ln k - 1)$ for $k \geq 2$*, Math. Comp., 68 (1999), pp. 411–415.

[9] D. Eppstein, M. T. Goodrich, and D. S. Hirschberg, *Improved combinatorial group testing for real-world problem sizes*, in Algorithms and Data Structures, 9th International Workshop, WADS 2005, Lecture Notes Comput. Sci. 3608, Springer, Berlin, 2005, pp. 86–98.

[10] M. Farach, S. Kannan, E. Knill, and S. Muthukrishnan, *Group testing problems with sequences in experimental molecular biology*, in SEQUENCES, IEEE Press, Washington, DC, 1997, pp. 357–367.

[11] F. K. Hwang and V. T. Sós, *Non-adaptive hypergeometric group testing*, Studia Sci. Math. Hungar., 22 (1987), pp. 257–263.

[12] W. H. Kautz and R. C. Singleton, *Nonrandom binary superimposed codes*, IEEE Trans. Information Theory, 10 (1964), pp. 363–377.

[13] A. J. Macula and G. R. Reuter, *Simplified searching for two defects*, J. Statist. Plann. Inference, 66 (1998), pp. 77–82.

[14] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, New York, 1995.

[15] M. Ruszinkó, *On the upper bound of the size of the r-cover-free families*, J. Combin. Theory Ser. A, 66 (1994), pp. 302–310.

SIAM J. COMPUT.
Vol. 36, No. 5, pp. 1376–1386

THE HARDNESS OF METRIC LABELING*

JULIA CHUZHOY† AND JOSEPH (SEFFI) NAOR‡

Abstract. The metric labeling problem is an elegant and powerful mathematical model capturing a wide range of classification problems. The input to the problem consists of a set L of labels and a weighted graph $G = (V, E)$. Additionally, a metric distance function on the labels is defined, and for each label and each vertex, an assignment cost is given. The goal is to find a minimum-cost assignment of the vertices to the labels. The cost of the solution consists of two parts: the assignment costs of the vertices and the separation costs of the edges (where each edge pays its weight times the distance between the two labels to which its endpoints are assigned). Due to the simple structure and the variety of applications, the problem and its special cases (with various distance functions on the labels) have recently received much attention. Metric labeling is known to have a logarithmic approximation, and it has been an open question for some time whether a constant approximation exists. We refute this possibility and prove that no constant factor approximation algorithm exists for metric labeling unless P=NP. Moreover, we prove that the problem is $\Omega((\log |V|)^{1/2-\delta})$-hard to approximate for any constant $\delta : 0 < \delta < 1/2$, unless NP has quasi-polynomial time algorithms.

Key words. metric labeling, hardness of approximation, classification

AMS subject classifications. 68Q15, 68Q17, 68Q25, 68W25

DOI. 10.1137/06065430X

1. Introduction.

The metric labeling problem, first formulated by Kleinberg and Tardos [18], captures a broad range of classification problems that arise in computer vision and related fields. In such classification problems, the goal is to assign labels to some given set of objects in a way consistent with observed data or some other form of prior knowledge. Formally, the input to the metric labeling problem consists of an n-vertex undirected graph $G(V, E)$, with weights w on edges, and a set L of labels with metric distance function $d : L \times L \to R$ associated with them. Additionally, for each vertex $v \in V$ and label $\ell \in L$, an assignment cost $c(v, \ell)$ is specified. The problem output is an assignment $f : V \to L$ of the vertices to the labels. Intuitively, the vertices are the objects we would like to classify, and the assignment function f provides such a classification. The prior knowledge is modeled by the means of the vertex assignment costs $c(v, \ell)$, which can be used to express an estimate of how likely it is that ℓ is the correct label for vertex v, and by the edge weights, which define pairwise relations between the objects. The weights of the edges express a prior estimate on how likely it is that the endpoints of the given edge should be assigned to close or identical labels.

Given a solution $f : V \to L$ to the metric labeling problem, its cost $Q(f)$ consists of two components.

Vertex labeling cost: For each $v \in V$, this cost is $c(v, f(v))$.

Edge separation cost: For each edge $e = (u, v)$ the cost is $w(u, v) \cdot d(f(u), f(v))$.

*Received by the editors March 14, 2006; accepted for publication (in revised form) September 11, 2006; published electronically January 12, 2007.

http://www.siam.org/journals/sicomp/36-5/65430.html

†School of Mathematics, IAS, 1 Einstein Drive, Princeton, NJ 08540 (cjulia@csail.mit.edu). This work was done while this author was a graduate student at the Computer Science Department at the Technion.

‡Computer Science Department, Technion, Haifa 32000, Israel (naor@cs.technion.ac.il). This author's research was supported in part by US-Israel BSF grant 2002276 and by EU contract IST-1999-14084 (APPOL II).

Thus,

$$Q(f) = \sum_{u \in V} c(u, f(u)) + \sum_{(u,v) \in E} w(u,v) \cdot d(f(u), f(v)),$$

and the goal is to find a labeling $f : V \to L$ minimizing $Q(f)$.

Metric labeling has rich connections to some well-known problems in combinatorial optimization. A special case of metric labeling is the 0-*extension* problem, studied by Karzanov [16, 17]. In this problem, there are no assignment costs. However, the graph contains a set $\{t_1, \ldots, t_k\}$ of terminals, where the label of terminal t_i is fixed in advance to i, while the nonterminals are free to be assigned to any of the labels. As in the metric labeling problem, a metric is defined on the set of labels. The cost of an assignment consists only of the *edge separation cost*. The 0-extension problem generalizes the well-studied *multiway cut* problem [9, 6, 14], which is defined exactly like 0-extension except that the metric on the label set is the uniform metric.

The first approximation algorithm for the metric labeling problem was shown by Kleinberg and Tardos [18]. This algorithm uses the probabilistic tree embedding technique [4, 5], and its approximation factor is $O(\log k \log\log k)$, where k denotes the number of labels in L. This bound was recently improved to $O(\log k)$ [11] and it is currently the best known approximation factor for the metric labeling problem. Some special cases of metric labeling, where the metric on the terminals belongs to some restricted class of metrics, were shown to have better approximation factors [18, 13, 8].

Chekuri et al. [8] proposed a natural linear programming formulation for the general metric labeling problem. A solution to this linear program is an embedding of the graph in a k-dimensional simplex, where the distance between points in the simplex is defined by a special metric, the *earth mover's distance metric* (EMD), and not by the (standard) ℓ_1 metric. It was also shown in [8] that the integrality gap of this formulation is at most the distortion of a probabilistic tree embedding of the given metric d, i.e., $O(\log k)$ [11]. Archer et al. [1] presented an approximation algorithm which is based on rounding the EMD solution to the linear program of [8] and achieves an $O(\log |V|)$ approximation factor.

Călinescu, Karloff, and Rabani [7] considered approximation algorithms for the 0-extension problem via a metric relaxation, originally studied by Karzanov [16], and obtained an $O(\log k)$-approximation algorithm for general metrics. This result was improved to $O(\log k / \log\log k)$ by Fakcharoenphol et al. [10].

Our results. A question that has intrigued many researchers since the appearance of [18] is whether there exists a constant factor approximation algorithm for the metric labeling problem. We answer this question in the negative and prove an $\Omega((\log n)^{1/2-\delta})$-hardness of approximation for any constant $\delta : 0 < \delta < 1/2$, assuming NP $\not\subseteq$ DTIME$(n^{\mathrm{poly}(\log n)})$. We also prove that there is no constant factor approximation algorithm for metric labeling unless P = NP. For the sake of simplicity, we focus on a problem called *restricted metric labeling*, which was shown by Chekuri et al. [8] to be equivalent to metric labeling. In the restricted metric labeling problem, the assignment costs of the vertices are restricted to be either 0 or ∞, or equivalently each vertex $v \in V$ has a list of labels, $L(v)$, to which it is allowed to be assigned. The solution cost then consists only of the edge separation cost.

Following our work, Karloff et al. [15] showed that the 0-extension problem is $\Omega((\log n)^{1/4-\epsilon})$-hard to approximate for any constant ϵ, unless NP has quasi-polynomial time algorithms. Their proof builds on ideas presented in this work.

Organization. We start in section 2 with some preliminaries, and we present in section 3 a simple $(3-\delta)$-hardness proof (for any constant $0 < \delta < 1$) for the restricted metric labeling problem. This proof provides the intuition and motivation for the new ideas needed to obtain the stronger hardness bounds shown in section 4.

2. Preliminaries. We prove our hardness results for the restricted metric labeling problem, defined as follows. The input consists of an undirected graph $G(V, E)$ with weights w on edges, and a set L of labels with distance function $d : L \times L \to R$. Additionally, for each vertex $v \in V$, there is a subset $L(v) \subseteq L$ of labels to which v can be assigned. The goal is to find an assignment $f : V \to L$ such that for each $v \in V$, $f(v) \in L(v)$. The solution cost is the sum, over all edges $e = (u, v)$, of the edge separation cost $w(e)d(f(u), f(v))$. We notice that our hardness results work even for the uniform weight function, i.e., for each edge $e \in E$, $w(e) = 1$.

We perform our reduction from the gap version of Max 3SAT(5). The input to the problem is a conjunctive normal form (CNF) formula ϕ with n variables and $\frac{5n}{3}$ clauses. Each clause consists of three literals, and each variable participates in five clauses, appearing in each clause at most once.

Let $\epsilon : 0 < \epsilon < 1$ be a constant, and let ϕ be an instance of Max 3SAT(5). Then ϕ is called a *Yes-instance* if there is an assignment that satisfies all the clauses, and it is called a *No-instance* (with respect to ϵ) if any assignment satisfies at most a fraction $(1 - \epsilon)$ of the clauses. Following is one of the several equivalent statements of the probabilistically checkable proofs (PCP) theorem [2, 3].

THEOREM 2.1. *There is a constant ϵ, $0 < \epsilon < 1$, such that it is NP-hard to distinguish between Yes-instances and No-instances of the Max 3SAT(5) problem.*

In our reduction, we start from a 3SAT(5) formula ϕ and produce an instance of the restricted metric labeling problem. Our first step is describing and analyzing a (standard) two-prover protocol for the 3SAT(5) problem. This protocol will help us translate 3SAT(5) instances into instances of restricted metric labeling, and analyze the reduction.

The one-round two-prover protocol for 3SAT(5) is defined as follows. Given a 3SAT(5) formula ϕ on n variables, we have the following:

- The verifier randomly chooses a clause C from the formula ϕ and one of the variables x belonging to C. Variable x is called the *distinguished variable*.
- Prover 1 receives clause C and is expected to return an assignment to all the variables appearing in the clause. Prover 2 receives variable x and is expected to return an assignment to x.
- After receiving the answers of the provers, the verifier checks that the answer of prover 1 defines a satisfying assignment for clause C and that the assignments of prover 1 and prover 2 to variable x are identical.

The following well-known theorem easily follows from Theorem 2.1.

THEOREM 2.2. *If ϕ is a Yes-instance, then there is a strategy of the provers such that the verifier always accepts. If ϕ is a No-instance, then for any strategy of the provers, the acceptance probability is at most $(1 - \frac{\epsilon}{3})$.*

3. A simple $(3 - \delta)$-hardness. In this section we present a simple $(3 - \delta)$-hardness for the restricted metric labeling problem (for any constant $0 < \delta < 1$), and also provide some intuition as to the new ideas needed to improve this bound.

We start by amplifying the soundness of the two-prover protocol presented above by means of parallel repetitions, a usual practice in PCP reductions. The number of repetitions is a sufficiently large constant l. The new protocol proceeds as follows:

- The verifier chooses, randomly and independently, l clauses $C_1, \ldots, C_l$ from the input formula ϕ. For each i, $1 \leq i \leq l$, the verifier chooses, randomly and independently, one variable x_i belonging to C_i.
- Prover 1 receives clauses $C_1, \ldots, C_l$ and is expected to return an assignment to all the variables appearing in the clauses, such that all clauses are satisfied. Prover 2 receives variables $x_1, \ldots, x_l$ and is expected to return an assignment to these variables.
- After receiving the answers of the provers, the verifier checks that the answer of prover 1 defines satisfying assignments for clauses $C_1, \ldots, C_l$ and that the assignments of prover 1 and prover 2 to variables $x_1, \ldots, x_l$ are identical.

The following theorem follows from the well-known Raz parallel repetition theorem [19], which bounds the error probability of the above protocol.

THEOREM 3.1. *There is a constant $\alpha > 0$ such that if ϕ is a Yes-instance, there is a strategy of the provers for which the verifier always accepts, and if ϕ is a No-instance, then for any strategy of the provers, the acceptance probability is at most $2^{-\alpha l}$.*

Let Q_1 denote the set of all the possible queries to prover 1 (i.e., each query $q \in Q_1$ is an l-tuple of clauses). Given a query $q_1 \in Q_1$, let $A(q_1)$ denote the set of all the assignments to the variables that appear in the clauses of q_1 that satisfy these clauses. Similarly, Q_2 denotes the set of all the possible queries to prover 2 (each query is an l-tuple of variables), and given $q_2 \in Q_2$, $A(q_2)$ is the set of all the possible answers of prover 2 to query q_2.

We assume that, at the beginning of the protocol, the verifier chooses a random string r, which determines the choice of the clauses and the variables sent to the provers. Let R denote the set of all the possible random strings. Given a random string $r \in R$, let $q_1(r), q_2(r)$ be the queries sent to prover 1 and prover 2, respectively, when the verifier chooses r.

The set of labels is defined as follows. For every possible query of each one of the two provers, and for every possible answer to this query, there is a label, i.e.,

$$L = \{\ell(q, A) \mid q \in Q_1 \cup Q_2, A \in A(q)\}.$$

In order to define the metric distance function on the labels, we construct a label graph G_L. The vertices of this graph are the labels, and the metric distance between the labels is defined to be the length of the shortest path in this graph. We now define the edges of graph G_L. Consider some random string r of the verifier, and let $q_1 = q_1(r)$, $q_2 = q_2(r)$. Let $A_1 \in A(q_1), A_2 \in A(q_2)$ be any pair of consistent answers to these queries. Then there is an edge of length 1 between $\ell(q_1, A_1)$ and $\ell(q_2, A_2)$ in G_L. Note that since each edge connects a label belonging to prover 1 and a label belonging to prover 2, the graph is bipartite. Therefore, for any random string r, if $q_1 = q_1(r)$ and $q_2 = q_2(r)$, and if $A_1 \in A(q_1)$, $A_2 \in A(q_2)$ are inconsistent answers to these queries, then the distance between labels $\ell(q_1, A_1)$ and $\ell(q_2, A_2)$ in graph G_L is at least 3.

We now proceed to define the input graph. For every query $q \in Q_1 \cup Q_2$, there is a vertex $v(q)$. This vertex can be assigned only to those labels that correspond to query q, i.e.,

$$V = \{v(q) \mid q \in Q_1 \cup Q_2\},$$
$$L(v(q)) = \{\ell(q, A) \mid A \in A(q)\}.$$

The edge set is defined as follows. For each random string r of the verifier, there is an edge connecting $v(q_1(r))$ and $v(q_2(r))$. All edges have unit weight.

Yes-instance. If ϕ is a Yes-instance, then there is a strategy of the provers such that their answers are always accepted by the verifier. This strategy defines the assignments of the vertices to the labels, namely, vertex $v(q)$ for $q \in Q_1 \cup Q_2$ is assigned to label $\ell(q, A)$, where $A \in A(q)$ is the answer of the corresponding prover to query q under the above strategy. Consider some random string r of the verifier and the corresponding queries $q_1 = q_1(r), q_2 = q_2(r)$. Let $A_1 \in A(q_1)$, $A_2 \in A(q_2)$ be the answers of the provers according to the above strategy. Note that vertices $v(q_1)$, $v(q_2)$ are assigned to labels $\ell(q_1, A_1)$, $\ell(q_2, A_2)$ and that the answers A_1 and A_2 of the provers are consistent. Therefore, there is an edge in the label graph between the labels $\ell(q_1, A_1)$ and $\ell(q_2, A_2)$, and thus the distance between the two labels (and the cost incurred by the edge connecting $v(q_1)$ and $v(q_2)$) is 1. The total cost of the solution is therefore $|R|$, where R is the set of all the random strings of the verifier.

No-instance. Consider any solution to the problem. Note that the assignments of the vertices to the labels define a strategy of the provers. (The assignment of vertex $v(q)$, $q \in Q_1 \cup Q_2$, to label $\ell(q, A)$, $A \in A(q)$, implies that the answer of the corresponding prover to query q is A.) Let $R' \subseteq R$ be the set of random strings of the verifier for which the answers of the two provers are inconsistent. From Theorem 3.1, $|R'| \geq (1 - 2^{-\alpha l})|R|$. Consider a random string $r \in R'$ and let $q_1 = q_1(r), q_2 = q_2(r)$. Let $\ell(q_1, A_1)$, $\ell(q_2, A_2)$ be the labels to which the vertices $v(q_1)$, $v(q_2)$ are assigned. As the answers A_1, A_2 of the provers are inconsistent, the distance between the two labels (and hence the separation cost paid by the edge between $v(q_1)$ and $v(q_2)$) is at least 3. Therefore, the total cost of the solution is at least $3(1 - 2^{-\alpha l})|R|) = 3(1 - \delta)|R|$, where δ is an arbitrarily small constant.

It follows that the gap between the costs of Yes- and No-instances is $3(1-\delta)$, and since the size of the construction is polynomial in n, we have that restricted metric labeling is $3(1 - \delta)$-hard to approximate for any constant δ, unless P=NP.

It is not hard to see that the analysis is tight. Consider some random string r and the corresponding queries $q_1 = q_1(r)$, $q_2 = q_2(r)$. Let A_1, A_2 be a pair of inconsistent answers to queries q_1, q_2. We show that there is a path of length 3 in the graph G_L between the pair of labels $\ell(q_1, A_1)$, $\ell(q_2, A_2)$. We denote $q_1 = (C_{i_1}, \ldots, C_{i_l})$ and $q_2 = (x_{i_1}, \ldots, x_{i_l})$, and recall that for each $j : 1 \leq j \leq l$, x_{i_j} is one of the variables of clause C_{i_j}. Let x'_{i_j} and x''_{i_j} denote the other two variables. The path of length 3 connecting the two labels starts at label $\ell(q_1, A_1)$. The second label on this path is $\ell(q'_2, A'_2)$, where $q'_2 = (x'_{i_1}, \ldots, x'_{i_l})$ and A'_2 contains assignments to $(x'_{i_1}, \ldots, x'_{i_l})$ identical to those in A_1. The third label is $\ell(q_1, A'_1)$ (we define A'_1 below), and the final fourth label is $\ell(q_2, A_2)$. In order to define A'_1, fix some $j : 1 \leq j \leq l$, and consider the jth entry of q_1, a clause whose variables are x_{i_j}, x'_{i_j}, and x''_{i_j}. We need to specify the assignments to these variables in A'_1. The assignment to x_{i_j} is defined to be the same assignment that appears in A_2, the assignment to x'_{i_j} is the same as in A'_2, and the assignment to x''_{i_j} is set in such a way that clause C_{i_j} is satisfied.

Thus, even though the two answers A_1 and A_2 of the provers might be inconsistent in many coordinates, there is still a short path between the two labels. In order to improve the hardness bound, it would be helpful (and enough) to ensure that if two answers are inconsistent in almost all the coordinates, then the length of the shortest path between the two corresponding labels is $\Omega(l)$. This is the intuition behind the construction and the k-prover protocol described in the next section.

4. The main hardness result. In this section we prove an $\Omega((\log n)^{1/2-\delta})$-hardness of restricted metric labeling, for any constant $\delta : 0 < \delta < \frac{1}{2}$. We start by

defining a new k-prover protocol for 3SAT(5). The protocol is then used in a way which is similar to the construction in section 3 to obtain the stronger hardness result.

4.1. A new k-prover protocol. We define a new k-prover protocol, where the k provers are denoted by $P_1, \ldots, P_k$, and k will be later set to poly$(\log n)$. This protocol is based on the basic one-round two-prover protocol, and it proceeds as follows.

- The verifier sends one query to each prover. Each one of the queries has $\binom{k}{2}$ entries, which are determined in the following way. For each pair (i, j) of provers, where $1 \leq i < j \leq k$, the verifier chooses, uniformly, independently at random, a clause C_{ij} and a distinguished variable x_{ij} belonging to this clause. The entry (i, j) in the queries of the provers is then defined as follows. In the query sent to prover P_i, this entry contains clause C_{ij}. In the query sent to prover P_j, this entry contains variable x_{ij}. For each other prover P_a, where $a \neq i, j$, the entry (i, j) of its query contains both clause C_{ij} and variable x_{ij}.
  Thus, in general, for any prover P_h, $1 \leq h \leq k$, coordinate (y, z) of its query (where $1 \leq y < z \leq k$), is defined as follows:
  - if $h = y$, then the entry contains C_{yz}.
  - if $h = z$, then the entry contains x_{yz}.
  - if $h \neq y, z$, then the entry contains both C_{yz} and x_{yz}.
- Each one of the provers responds with an assignment to all the variables appearing in its query, both as parts of clauses and as distinguished variables.
- After receiving the answers of the provers, the verifier checks, for each coordinate (i, j), $1 \leq i < j \leq k$, that the answers of all the provers are consistent, i.e., all the provers P_a, $a \neq j$, return an identical assignment to the variables of C_{ij}, and the assignment of prover P_j to variable x_{ij} matches the assignments of all the other provers.

We note that our k-prover system departs from standard protocols in several ways. First, we do not use the parallel repetitions theorem here, as there is no need to amplify the soundness of the protocol. Observe also that for each prover P_a, for each coordinate $(i, j) : i, j \neq a$, the prover receives both the clause C_{ij} and the distinguished variable x_{ij}. It may appear that some of the information the prover receives is redundant. Indeed, in k-prover systems (e.g., [12]), the provers usually receive either the clause or the distinguished variable, but not both. However, this sending of redundant information to the provers is essential for our reduction. Intuitively, it will ensure that if, for some random string r, the answers of the k provers are inconsistent in many coordinates, then the distances between the corresponding labels are long.

We assume again that all the random choices of the verifier are made at the beginning of the protocol, by choosing a random string r out of the set R of all the possible random strings of the desired length. Given a random string $r \in R$, for each i, $1 \leq i \leq k$, let $q_i(r)$ be the query sent to prover P_i when the verifier chooses the random string r, and let Q_i be the set of all the possible queries of prover i. For each $i : 1 \leq i \leq k$, for each $q_i \in Q_i$, let $A(q_i)$ denote the set of all the possible answers of prover P_i to query q_i, which satisfy all the clauses appearing in the query.

DEFINITION 4.1. *Consider a pair of provers P_i and P_j, $1 \leq i < j \leq k$, and let $q_i \in Q_i$, $q_j \in Q_j$ be a pair of queries such that for some random string $r \in R$, $q_i = q_i(r), q_j = q_j(r)$. Let A_i and A_j denote the respective answers of the provers to the queries. We say that the answers are* weakly consistent *if the assignments to C_{ij} and x_{ij} in A_i and A_j, respectively, are consistent. The answers are called* strongly consistent *if they are also consistent in every other coordinate, i.e., for each (a, b), $1 \leq a < b \leq k$, where $(a, b) \neq (i, j)$, the following hold:*

- *If both entries $q_i(a,b)$ and $q_j(a,b)$ contain clause C_{ab} and variable x_{ab}, then the assignments to the variables of clause C_{ab} in A_i and A_j are identical.*
- *If one of the entries $q_i(a,b)$ and $q_j(a,b)$ contains clause C_{ab} and the other contains clause C_{ab} and variable x_{ab}, then the assignments to the variables of the clause C_{ab} in A_i and A_j are identical.*
- *If one of the entries $q_i(a,b)$ and $q_j(a,b)$ contains variable x_{ab} and the other contains clause C_{ab} and variable x_{ab}, then the assignments to the variables of clause C_{ab} and variable $x_{a,b}$ in A_i and A_j are consistent.*

THEOREM 4.2. *If ϕ is a Yes-instance, then there is a strategy of the k provers such that the verifier always accepts. If ϕ is a No-instance, then for any strategy of the provers, for every pair of provers P_i and P_j, $1 \le i < j \le k$, the probability that their answers are weakly consistent is at most $(1 - \frac{\epsilon}{3})$.*

Proof. For the Yes-instance, the theorem follows immediately. We now prove that the theorem holds for the No-Instance. Assume otherwise. Let P_i and P_j be a pair of provers such that the probability that their answers are weakly consistent is more than $(1 - \frac{\epsilon}{3})$. We partition the set of random strings R into classes such that within each class the random strings are identical except for the clause C_{ij} and the distinguished variable x_{ij}. Each such class (together with the corresponding queries and answers to the queries) can be viewed as a two-prover protocol (while we ignore all the coordinates of the queries and the answers except for (i,j)). As the probability of obtaining a pair of weakly consistent answers is more than $(1 - \frac{\epsilon}{3})$, at least for one of the classes, the probability that the verifier accepts is greater than $(1 - \frac{\epsilon}{3})$. This defines a strategy for the two-prover protocol in which the acceptance probability of the verifier is greater than $(1 - \frac{\epsilon}{3})$, contradicting Theorem 2.2. □

4.2. The graph and the label set. In this section we construct an instance of the restricted metric labeling problem from an input 3SAT(5) formula ϕ. Our construction is based on the k-prover system described above.

The set of labels L consists of two subsets:

Query labels: For each prover P_i, $1 \le i \le k$, for each query $q \in Q_i$, and for each answer $A \in A(q)$ to the query q, there is a label $\ell(P_i, q, A)$.

Constraint labels: Consider a random string r of the verifier. Let $A_1, \ldots, A_k$, be any collection of possible answers of the provers to the queries $q_1(r), \ldots, q_k(r)$; i.e., for each $1 \le i \le k$, $A_i \in A(q_i(r))$. Moreover, assume that these answers are accepted by the verifier, (i.e., $A_1, \ldots, A_k$ are strongly consistent). Then, there is a label $\ell(r, A_1, A_2, \ldots, A_k)$.

We now define a graph $G_L(L, E')$ on the label set. The metric on the label set is implied by the shortest path distance function in the graph. The vertices of G_L are the labels, and the edges are defined as follows. Consider a constraint label $\ell = \ell(r, A_1, A_2, \ldots, A_k)$, Then, for each i, $1 \le i \le k$, there is an edge of length $\frac{1}{2}$ between ℓ and $\ell(P_i, q_i(r), A_i)$ (see Figure 4.1).

Thus, the graph is a collection of stars, while some stars share some of their leaves.

We now proceed to define graph $G(V, E)$. The vertex set V is the union of two vertex sets: a set of *query vertices*, denoted by V_1, and a set of *constraint vertices*, denoted by V_2.

Query vertices: For each prover P_i, $1 \le i \le k$, and for each query $q \in Q_i$, there is a vertex $v(P_i, q)$. Thus,

$$V_1 = \{v(P_i, q) \mid 1 \le i \le k \text{ and } q \in Q_i\}.$$

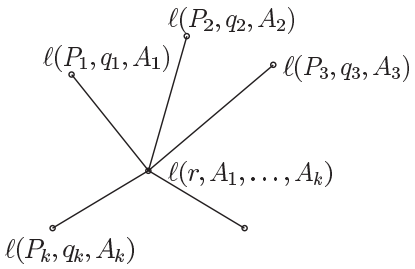


FIG. 4.1. *Edges in the graph of labels incident to* $\ell(r, A_1, \ldots, A_k)$.

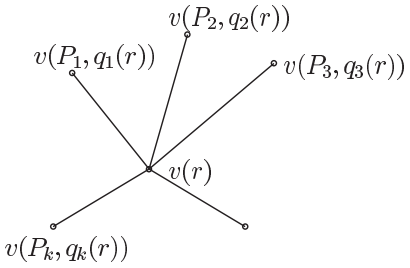


FIG. 4.2. *Edges incident to* $v(r)$.

Vertex $v(P_i, q)$ can be assigned only to the labels corresponding to (P_i, q_i), i.e.,

$$L(v(P_i, q)) = \{\ell(P_i, q, A) \mid A \in A(q)\}.$$

Note that assigning $v(P_i, q)$ to a label in $L(v(P_i, q))$ defines an answer of prover P_i to query q.

Constraint vertices: For each random string r, there is a vertex $v(r)$, i.e.,

$$V_2 = \{v(r) \mid r \in R\}.$$

Vertex $v(r)$ can be assigned only to labels corresponding to r; i.e., $L(v(r))$ consists of labels $\ell(r, A_i, \ldots, A_k)$ such that $\forall i$, $A_i \in A(q_i(r))$ and $(A_1, \ldots, A_k)$ are strongly consistent.

The edges of the graph are as follows. Every constraint vertex $v(r)$ is connected to every assignment vertex $v(P_i, q_i(r))$ by a unit-weight edge (see Figure 4.2).

The graph is therefore a collection of stars that can have common leaves.

4.3. Hardness of approximation proof.

4.3.1. Yes-instances. Assume that the input 3SAT(5) formula ϕ is a Yes-instance. Consider a strategy of the provers for which the acceptance probability of the verifier is 1. For every prover P_i, $1 \le i \le k$, for every query $q \in Q_i$, let $f(q) \in A(q)$ be the answer of prover P_i to query q under this strategy. Note that for each random string r, $f(q_1(r)), \ldots, f(q_k(r))$ are strongly consistent. We define the following labeling of the graph G (see Figure 4.3):

- For each random string $r \in R$, vertex $v(r)$ is assigned to label $\ell(r, f(q_1(r)), \ldots, f(q_k(r)))$.

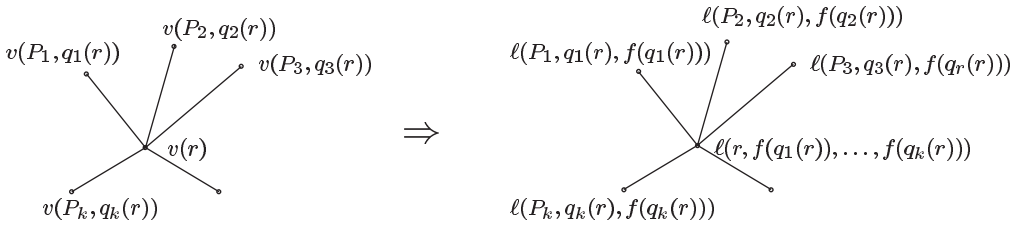


FIG. 4.3. *Yes-instance: the embedding of edges incident to $v(r)$.*

- For each $i : 1 \le i \le k$, $q \in Q_i$, vertex $v(P_i, q)$ is assigned to label $\ell(P_i, q, f(q))$.

Consider an edge in the graph G between $v(r)$ and $v(P_i, q_i(r))$, $r \in R$, $1 \le i \le k$. Vertex $v(r)$ is assigned to label $\ell(r, f(q_1(r)), \ldots, f(q_k(r)))$, and vertex $v(P_i, q_i(r))$ is assigned to label $\ell((P_i, q_i(r), f(q_i(r)))$. Thus, the separation cost of the edge is $\frac{1}{2}$, since the distance between the two labels is $\frac{1}{2}$. Hence, the total cost of the solution is $\frac{1}{2} \cdot k \cdot |R|$.

4.3.2. No-instances. Assume that the input 3SAT(5) formula ϕ is a No-instance. We prove that the cost of any solution to the metric labeling instance is at least $\binom{k}{2} \cdot \frac{\epsilon}{3} \cdot |R|$, and thus the gap between the Yes- and the No-instances is $\Omega(k)$. Observe that the assignment of the query vertices to query labels defines a strategy of the provers. We concentrate on this strategy and define the set $T \subseteq R \times [k] \times [k]$.

DEFINITION 4.3. *For $r \in R$, $1 \le i < j \le k$, $(r, i, j) \in T$ if and only if the answers of provers P_i and P_j to queries $q_i(r)$ and $q_j(r)$, respectively, are not weakly consistent (under the above strategy).*

The following proposition is a direct consequence of Theorem 4.2.

PROPOSITION 4.4. $|T| \ge \binom{k}{2} \cdot \frac{\epsilon}{3} \cdot |R|$.

Consider an edge $e \in E$, and assume that the endpoints of the edge are assigned to labels ℓ_1 and ℓ_2. We denote by $\mathcal{P}_e$ the shortest path between the labels ℓ_1 and ℓ_2 in the graph of labels G_L. Note that the length of $\mathcal{P}_e$ is exactly the cost paid by edge e, and the solution cost is $\sum_{e \in E} |\mathcal{P}_e|$. We define the set $T' \subseteq R \times [k] \times [k]$ as follows. Consider a random string $r \in R$ and a pair of provers P_i and P_j, $1 \le i, j \le k$, $i \ne j$. Let e be the edge between $v(r)$ and $v(P_i, q_i(r))$. Then, $(r, i, j) \in T'$ if and only if the path $\mathcal{P}_e$ contains a label belonging to prover P_j (i.e., a label of the form $\ell(P_j, q, A)$, for some $q \in Q_j, A \in A(q)$). Observe that the cost of the solution is at least $|T'|$.

LEMMA 4.5. *For $r \in R$, suppose $(r, i, j) \in T$, where $1 \le i < j \le k$. Then, either $(r, i, j) \in T'$ or $(r, j, i) \in T'$.*

Proof. Suppose that vertex $v(r)$ is assigned to label $\ell(r, A_1, \ldots, A_k)$, and suppose vertices $v(P_i, q_i(r))$ and $v(P_j, q_j(r))$ are assigned to labels $\ell(P_i, q_i(r), A'_i)$ and $\ell(P_j, q_j(r), A'_j)$, respectively. As $(r, i, j) \in T$, the answers A'_i and A'_j of provers P_i and P_j cannot be weakly consistent. However, the answers A_i and A_j are strongly consistent. Therefore, either the (i, j) coordinates in A_i and A'_i differ (recall that this coordinate contains an assignment to a clause C_{ij}), or the (i, j) coordinates in A_j and A'_j differ (this coordinate contains an assignment to a distinguished variable x_{ij}). Assume that the former is true (the other case is handled similarly).

Let e be the edge between $v(r)$ and $v(P_i, q_i(r))$. It is enough to show that the path $\mathcal{P}_e$ contains a label corresponding to prover P_j. Suppose that this is not the case. Let $\ell(P_a, q_a, A)$ and $\ell(P_b, q_b, A')$ be two consecutive query labels on the path. As the two labels are at distance 1, there must be an $r' \in R$ such that $q_a = q_a(r')$ and $q_b = q_b(r')$ and the answers A and A' are strongly consistent. As $a, b \ne j$, the (i, j)

coordinate in q_a and in q_b must contain some clause, and the two clauses are identical. Moreover, coordinate (i, j) of A and A' must contain an identical assignment to the variables of this clause. Therefore, if path $\mathcal{P}_e$ starts at $\ell(P_i, q_i(r), A'_i)$ and does not pass through any label belonging to prover P_j, then for every query label $\ell(P_s, q_s, A)$ appearing on the path, coordinate (i, j) of q_s contains the same clause as that of $q_i(r)$, and coordinates (i, j) in A and A'_i are identical. This is also true for the last query label on the path, denoted by $\ell(P_d, q_d, A_d)$. But this label is connected by an edge to label $\ell(r, A_1, \ldots, A_k)$, and therefore coordinates (i, j) of A_d and A_i must be identical, which is impossible. □

It follows from the lemma that $|T'| \geq |T|$, yielding that the solution cost is at least $\binom{k}{2} \cdot \frac{\epsilon}{3} \cdot |R|$.

4.3.3. The hardness factor. The gap between the cost of the Yes-instance and the No-instance solutions is $\Omega(k)$. The size of the construction is dominated by the number of labels. For each i, $1 \leq i \leq k$, $|Q_i| \leq (5n)^{k^2}$, and for each $q \in Q_i$, $|A(q)| \leq 7^{k^2}$, and therefore the number of query labels is at most $k(5n)^{k^2} \cdot 7^{k^2}$. The size of R is at most $(5n)^{k^2}$ and for each $r \in R$ the number of k-tuples of consistent answers is at most 7^{k^2}. Hence, the number of constraint labels is bounded by $(5n)^{k^2} \cdot 7^{k^2}$. The construction size is therefore $N = n^{O(k^2)}$. If k is a constant, then it is polynomial in n. Choosing $k = \text{poly}(\log n)$, we get that $k = (\log N)^{\frac{1}{2}-\delta}$ for arbitrarily small constant $\delta > 0$.

Thus, we have proved the following result.

THEOREM 4.6. *There is no efficient constant factor approximation algorithm for the metric labeling problem, unless P=NP. Moreover, for any constant* $0 < \delta < 1/2$, *there is no* $\Omega((\log N)^{\frac{1}{2}-\delta})$*-approximation algorithm for the problem, unless NP* $\subseteq$ *DTIME*$(n^{poly(\log n)})$.

Acknowledgment. The authors would like to thank Sanjeev Khanna for helpful comments on the presentation of the paper.

REFERENCES

[1] A. ARCHER J. FAKCHAROENPHOL, C. HARRELSON, R. KRAUTHGAMER, K. TALWAR, AND É. TARDOS, *Approximate classification via earthmover metrics*, in Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA, 2004, SIAM, Philadelphia, 2004, pp. 1072–1080.

[2] S. ARORA, C. LUND, R. MOTWANI, M. SUDAN, AND M. SZEGEDY, *Proof verification and the hardness of approximation problems*, J. ACM, 45 (1998), pp. 501–555.

[3] S. ARORA AND S. SAFRA, *Probabilistic checking of proofs: A new characterization of* NP, J. ACM, 45 (1998), pp. 70–122.

[4] Y. BARTAL, *Probabilistic approximation of metric spaces and its algorithmic applications*, in Proceedings of the 37th IEEE Symposium on Foundations of Computer Science, Burlington, VT, 1996, IEEE Press, Piscataway, NJ, 1996, pp. 184–193.

[5] Y. BARTAL, *On approximating arbitrary metrics by tree metrics*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, Dallas, TX, 1998, ACM, New York, 1998, pp. 161–168.

[6] G. CĂLINESCU, H. KARLOFF, AND Y. RABANI, *An improved approximation algorithm for multiway cut*, J. Comput. System Sci., 60 (2000), pp. 564–574.

[7] G. CĂLINESCU, H. KARLOFF, AND Y. RABANI, *Approximation algorithms for the* 0*-extension problem*, SIAM J. Comput., 34 (2005), pp. 358–372.

[8] C. CHEKURI, S. KHANNA, J. NAOR, AND L. ZOSIN, *A linear programming formulation and approximation algorithms for the metric labeling problem*, SIAM J. Discrete Math., 18 (2005), pp. 608–625.

[9] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis, *The complexity of multiterminal cuts*, SIAM J. Comput., 23 (1994), pp. 864–894.
[10] J. Fakcharoenphol, C. Harrelson, S. Rao, and K. Talwar, *An improved approximation algorithm for the* 0*-extension problem*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, 2003, SIAM, Philadelphia, 2003, pp. 257–265.
[11] J. Fakcharoenphol, S. Rao, and K. Talwar, *A tight bound on approximating arbitrary metrics by tree metrics*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, San Diego, CA, 2003, ACM, New York, 2003, pp. 448–455.
[12] U. Feige, *A threshold of ln n for approximating set cover*, J. ACM, 45 (1998), pp. 634–652.
[13] A. Gupta and E. Tardos, *A constant factor approximation algorithm for a class of classification problems*, in Proceedings of the ACM Symposium on the Theory of Computing, Portland, OR, 2000, ACM, New York, 2000, pp. 652–658.
[14] D. R. Karger, P. N. Klein, C. Stein, M. Thorup, and N. E. Young, *Rounding algorithms for a geometric embedding of minimum multiway cut*, Math. Oper. Res., 29 (2004), pp. 436–461.
[15] H. Karloff, S. Khot, A. Mehta, and Y. Rabani, *On earthmover distance, metric labeling, and* 0*-extension*, in Proceedings of the 38th Annual ACM symposium on Theory of Computing, Seattle, WA, 2006, ACM, New York, 2006, pp. 547–556.
[16] A. Karzanov, *Minimum* 0*-extension of graph metrics*, European J. Combin., 19 (1998), pp. 71–101.
[17] A. Karzanov, *A combinatorial algorithm for the minimum* $(2, r)$*-metric problem and some generalizations*, Combinatorica, 18 (1999), pp. 549–569.
[18] J. Kleinberg and E. Tardos, *Approximation algorithms for classification problems with pairwise relationships: Metric labeling and Markov random fields*, J. ACM, 49 (2002), pp. 616–630.
[19] R. Raz, *A parallel repetition theorem*, SIAM J. Comput., 27 (1998), pp. 763–803.

SIAM J. COMPUT.
Vol. 36, No. 5, pp. 1387–1403

PSEUDORANDOM BITS FOR CONSTANT-DEPTH CIRCUITS WITH FEW ARBITRARY SYMMETRIC GATES*

EMANUELE VIOLA†

Abstract. We exhibit an explicitly computable pseudorandom generator stretching l bits into $m(l) = l^{\Omega(\log l)}$ bits that look random to constant-depth circuits of size $m(l)$ with $\log m(l)$ arbitrary symmetric gates (e.g., PARITY, MAJORITY). This improves on a generator by Luby, Velickovic, and Wigderson [*Proceedings of the Second Israel Symposium on Theory of Computing Systems*, 1993, pp. 18–24] that achieves the same stretch but fools only circuits of depth 2 with one arbitrary symmetric gate at the top. Our generator fools a strictly richer class of circuits than Nisan's generator for constant-depth circuits (but Nisan's generator has a much bigger stretch) [*Combinatorica*, 11 (1991), pp. 63–70]. In particular, we conclude that every function computable by uniform poly(n)-size *probabilistic* constant-depth circuits with $O(\log n)$ arbitrary symmetric gates is in $TIME(2^{n^{o(1)}})$. This seems to be the richest probabilistic circuit class known to admit a subexponential derandomization. Our generator is obtained by constructing an explicit function $f : \{0,1\}^n \to \{0,1\}$ that is very hard *on average* for constant-depth circuits of size $s(n) = n^{\Omega(\log n)}$ with $\log s(n)$ arbitrary symmetric gates, and plugging it into the Nisan–Wigderson pseudorandom generator construction [*J. Comput. System Sci.*, 49 (1994), pp. 149–167]. The proof of the average-case hardness of this function is a modification of arguments by Razborov and Wigderson [*Inform. Process. Lett.,* 45 (1993), pp. 303–307] and Hansen and Miltersen [*Proceedings of the* 29*th International Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Comput. Sci. 3153, Springer-Verlag, Berlin, 2004, pp. 334–345] and combines Håstad's switching lemma [*Computational Limitations of Small-Depth Circuits*, MIT Press, Cambridge, MA, 1987] with a multiparty communication complexity lower bound by Babai, Nisan, and Szegedy [*J. Comput. System Sci.*, 45 (1992), pp. 204–232].

Key words. pseudorandom generator, derandomization, constant-depth circuit, average-case hardness, lower bound, symmetric gate, switching lemma, communication complexity

AMS subject classifications. 68Q01, 68Q10, 68Q15, 68Q17

DOI. 10.1137/050640941

1. Introduction.

A *pseudorandom generator* $G : \{0,1\}^l \to \{0,1\}^m$ is an efficient procedure that stretches l input bits into $m \gg l$ output bits such that the output distribution of the generator *fools* small circuits. That is, for every circuit C of size m we have

$$\left| \Pr_{x \in \{0,1\}^l}[C(G(x)) = 1] - \Pr_{x \in \{0,1\}^m}[C(x) = 1] \right| \leq \frac{1}{m}.$$

Pseudorandom generators have found a striking variety of applications in complexity theory, most notably to *derandomize* probabilistic algorithms.

Starting with the seminal work of Nisan and Wigderson [21], a series of results (e.g., [3, 28, 26, 29]) show how to construct pseudorandom generators starting from an explicit function that requires circuits of superpolynomial size. However, no such function is known to exist.

*Received by the editors September 21, 2005; accepted for publication (in revised form) July 31, 2006; published electronically January 22, 2007. An extended abstract of this paper appeared in *Proceedings of the* 20*th Annual IEEE Conference on Computational Complexity*, 2005 [31]. This research was done while the author was a Ph.D. student at Harvard University, supported by NSF grant CCR-0133096, US-Israel BSF grant 2002246, and ONR grant N-00014-04-1-0478.

http://www.siam.org/journals/sicomp/36-5/64094.html

†Institute for Advanced Study, School of Mathematics, 1 Einstein Drive, Princeton, NJ 08540 (viola@ias.edu).

On the other hand, pseudorandom generators that fool *restricted* kinds of circuits, such as *constant-depth* circuits with unbounded fan-in, are already very interesting. They also have a large variety of applications (e.g., [21, 16]) and are central to understanding the power of randomness in restricted classes of algorithms. While there has been exciting progress in constructing explicit functions that require superpolynomial-size constant-depth circuits with certain kinds of gates (e.g., [14, 23, 27, 15, 22, 13]), no explicit function is known to require superpolynomial-size constant-depth circuits with MAJORITY gates (cf. [25]). This is an obstacle to constructing pseudorandom generators, as most constructions need such a function. This need is due to the fact that the reductions in the proofs of correctness of these constructions use (a polynomial number of) MAJORITY gates (cf. [1], [30, section 10], and [32, Chapter 6]).

But when starting from an *average-case* hard function, the reduction in the proof of correctness of the Nisan–Wigderson construction [21] does not require MAJORITY gates. (A function $f : \{0,1\}^n \to \{0,1\}$ is average-case hard if polynomial-size circuits fail to compute f with probability at least $1/2 - 1/n^{\omega(1)}$ over random input.) Thus, one can plug average-case lower bounds into the Nisan–Wigderson construction to get a generator that fools small constant-depth circuits. This approach is used in a celebrated work by Nisan [20] (which actually predates the more general construction in [21]) in which he exhibits a generator $G : \{0,1\}^l \to \{0,1\}^{2^{l^{\Omega(1)}}}$ that fools small AC^0 circuits (i.e., constant-depth circuits with AND and OR gates). This generator is based on the fact that PARITY is very average-case hard for small AC^0 circuits [14].

Subsequently, Luby, Velickovic, and Wigderson (Theorem 2 in [19]) built a generator $G : \{0,1\}^l \to \{0,1\}^{l^{\Omega(\log l)}}$ that fools small SYM $\circ$ AND circuits, i.e., depth-2 circuits with one *arbitrary symmetric gate* at the top and AND gates at the bottom. By arbitrary symmetric gate we mean a gate that computes an arbitrary function whose value depends only on the number of input bits being 1, important examples being PARITY and MAJORITY. This generator is based on the fact that the generalized inner product function is average-case hard for small SYM $\circ$ AND circuits with small bottom fan-in [4, 15].

The above two generators ([20] and Theorem 2 in [19]) fool two incomparable classes of circuits (i.e., small AC^0 circuits and small SYM $\circ$ AND circuits). In this work we exhibit a generator that fools a class of circuits strictly richer than both of them, namely, small constant-depth circuits with few arbitrary symmetric gates.

1.1. Our results. In this paper we exhibit the following generator.

THEOREM 1. *For every constant d there is a constant $\epsilon > 0$ such that for every l there is a generator $G : \{0,1\}^l \to \{0,1\}^m$, where $m = m(l) := l^{\epsilon \log l}$, such that for every circuit C of size m and depth d with $\log m(l)$ arbitrary symmetric gates, we have*

$$\left| \Pr_{x \in \{0,1\}^m}[C(x) = 1] - \Pr_{x \in \{0,1\}^l}[C(G(x)) = 1] \right| \leq \frac{1}{m},$$

and given $x \in \{0,1\}^l, i \leq m$, we can compute the ith output bit of $G(x)$ in time $\mathrm{poly}(l)$.

The generator in Theorem 1 improves on the generator by Luby, Velickovic, and Wigderson (Theorem 2 in [19]) that achieves the same stretch (up to a different constant ϵ) but fools only circuits of depth 2 (as opposed to any constant depth) with one symmetric gate at the top. We elaborate more on the difference between the two

generators in section 6. The generator in Theorem 1 also fools a strictly richer class of circuits than Nisan's generator, which fools constant-depth circuits [20]. However, Nisan's generator has a much bigger stretch: it stretches l bits to $2^{l^{\Omega(1)}}$ bits, as opposed to $l^{\Omega(\log l)}$ in Theorem 1.

As a standard consequence of Theorem 1 we obtain the following subexponential derandomization of probabilistic polynomial-size constant-depth circuits with a logarithmic number of arbitrary symmetric gates. This seems to be the richest probabilistic circuit class known to admit a subexponential derandomization. See [21] for the connection between generators and derandomization.

COROLLARY 2. *Let a function f be computed by a polynomial-time uniform family of probabilistic* $\mathrm{poly}(n)$*-size constant-depth circuits with $O(\log n)$ arbitrary symmetric gates. Then f can be computed in deterministic time* $\exp(2^{O(\sqrt{\log n})}) = 2^{n^{o(1)}}$.

1.2. Techniques. The generator in Theorem 1 is obtained by plugging into the Nisan–Wigderson pseudorandom generator construction [21] a function that is very hard on average for "small" constant-depth circuits with "few" arbitrary symmetric gates (cf. Theorem 3).

Here a simple and crucial observation is that the reduction in the proof of correctness of the Nisan–Wigderson generator does not increase the number of arbitrary symmetric gates.

Given our average-case hardness result (Theorem 3), the construction of our generator is simpler than the construction of the generator by Luby, Velickovic, and Wigderson (Theorem 2 in [19]) that uses more involved combinatorial arguments than those in [21]. These more involved combinatorial arguments were probably used because the generator in [19] builds on a function that is hard on average for circuits of depth 2 (as opposed to any constant depth), and thus one cannot use directly the Nisan–Wigderson construction [21] since the reduction in its proof of correctness increases the depth by 1.

We now state our average-case hardness result.

THEOREM 3. *There is a function $f : \{0,1\}^* \to \{0,1\}$ computable in polynomial time such that for every constant d there is a constant $\epsilon > 0$ such that for every n and every circuit C of size $n^{\epsilon \log n}$ and depth d with $\epsilon \log^2 n$ arbitrary symmetric gates, the following holds:*

$$\Pr_{x \in \{0,1\}^n}[C(x) \neq f(x)] \geq 1/2 - 1/n^{\epsilon \log n}.$$

We now explain the techniques involved in proving Theorem 3.

To simplify the discussion we first focus on how to prove an average-case hardness result for small constant-depth circuits with *one* arbitrary symmetric gate at the top, i.e., small $\mathrm{SYM} \circ \mathrm{AC}^0$ circuits (Theorem 4). The extension to circuits with more arbitrary symmetric gates is deferred to the later subsection on circuits with more arbitrary symmetric gates.

We obtain our average-case hardness result for small $\mathrm{SYM} \circ \mathrm{AC}^0$ circuits through a modification of previous lower bounds. We now discuss these previous lower bounds, then discuss why they are not sufficient for our purposes, and then sketch the proof of our average-case hardness result for small $\mathrm{SYM} \circ \mathrm{AC}^0$ circuits.

Previous lower bounds. Babai, Nisan, and Szegedy [4] proved that the generalized inner product function (i.e., $GIP_{n,s}(x) := \bigoplus_{i \leq n} \bigwedge_{j \leq s} x_{i,j}$) is very hard *on average* for multiparty communication complexity protocols among few parties that communicate little.

Håstad and Goldmann [15] noticed that any function computed by a small depth-2 circuit with an arbitrary symmetric gate of unbounded fan-in at the top and (arbitrary) gates of small fan-in at the bottom can be computed by a multiparty communication complexity protocol among few parties communicating little.

Thus, by the above result [4], they obtain that *GIP* is average-case hard for that class of circuits. Now, by the so-called ϵ-discriminator lemma[1] of Hajnal et al. [12] they conclude that *GIP* cannot be computed, in the *worst case*, by small depth-3 circuits with one majority gate of unbounded fan-in at the top, arbitrary symmetric gates of unbounded fan-in in the middle, and (arbitrary) gates of small fan-in at the bottom.

Razborov and Wigderson [22] eliminated the constraint on the bottom fan-in: they exhibited a new function *RW* that cannot be computed, in the worst case, by small depth-3 circuits with one majority gate at the top, symmetric gates in the middle, and AND gates at the bottom, where all the gates have unbounded fan-in (MAJ $\circ$ SYM $\circ$ AND circuits). Their function *RW* is obtained from *GIP* by replacing each input variable with a parity function, i.e., $RW(x) := \bigoplus_{i \le n} \bigwedge_{j \le \log n} \bigoplus_{k \le n} x_{i,j,k}$.

To explain their argument we introduce *restrictions* [11]. A restriction on m variables $x_1, x_2, \ldots, x_m$ is a map $\rho : \{x_1, x_2, \ldots, x_m\} \to \{0, 1, *\}$. For a circuit C we denote by $C|_\rho$ the circuit we get by doing the substitutions prescribed by ρ, followed by all obvious cancellations made possible by applying ρ. The input variables of $C|_\rho$ are the variables which were given the value * by ρ.

The argument in [22] is as follows. Suppose that *RW* is computable by a small MAJ $\circ$ SYM $\circ$ AND circuit C. Then there is a restriction ρ that accomplishes simultaneously two things: (1) $C|_\rho$ has small bottom fan-in and (2) $C|_\rho$ is still computing *GIP* as a subfunction. Note that by definition of *RW* and by the nature of parity, (2) happens whenever for every i, j there is k such that $\rho(x_{i,j,k}) = *$. But (1) and (2) contradict the above result by Håstad and Goldmann.

Finally, Hansen and Miltersen [13] observed that *RW* actually cannot be computed by small circuits of *any* constant depth with one majority gate at the top, and one layer of arbitrary symmetric gates immediately below it, where all the gates have unbounded fan-in (MAJ $\circ$ SYM $\circ$ AC^0 circuits). The argument in [13] goes as follows. Suppose that *RW* is computable by a small MAJ $\circ$ SYM $\circ$ AC^0 circuit C. Then there is a restriction ρ that accomplishes simultaneously two things: (1′) $C|_\rho$ is equivalent to a small MAJ $\circ$ SYM $\circ$ AND circuit and (2′) $C|_\rho$ is still computing *RW* on an input of polynomially related size. (1′) is obtained through Håstad's switching lemma [14], and for (2′) they show that for every i, j there are many k such that $\rho(x_{i,j,k}) = *$. But (1′) and (2′) contradict the above result by Razborov and Wigderson.

Why previous lower bounds are not sufficient to our purposes. The main problem with these previous lower bounds is that they give only a function that is *worst-case* hard for SYM $\circ$ AC^0 circuits, while as explained before we need a function that is *average-case* hard. In fact, the choice of parameters in the definition of *RW* implies that $\Pr_x[RW(x) = 0] = 1/2 + \Omega(1)$, and thus *RW* cannot be average-case hard (since the constant-size circuit that always outputs 0 computes the function fairly well on average). Moreover the choice of parameters for the restrictions in [22] does not guarantee that the reduction holds with high probability, which is needed to establish average-case hardness.

Proof sketch of our average-case hardness result for $\mathbf{SYM \circ AC^0}$ *circuits.* We

[1] This lemma states that if a function is computed by a small circuit with a MAJORITY gate at the top, then some input circuit to the MAJORITY gate computes the function well on average.

define a function f (similar to RW, but with a different choice of parameters), and we show that f is average-case hard for $\mathrm{SYM} \circ \mathrm{AC}^0$ circuits. Our argument simplifies the previous ones and goes as follows. Suppose that C is a small $\mathrm{SYM} \circ \mathrm{AC}^0$ circuit computing f. We argue that with high probability $\left(1 - n^{-\Omega(\log n)}\right)$ over the choice of a random restriction ρ, both of the following events happen:

- Event $E_1 :=$ the function computed by $C|_\rho$ is computable by a multiparty communication complexity protocol among few parties communicating little.
- Event $E_2 := C|_\rho$ is computing GIP as a subfunction.

To show E_1 we use Håstad's switching lemma to argue that with high probability over ρ, $C|_\rho$ is equivalent to a small depth-2 circuit with a symmetric gate at the top (of unbounded fan-in) and AND gates of small fan-in at the bottom, and then we use Håstad and Goldmann's connection [15] between these circuits and multiparty communication complexity protocols (cf. the subsection on previous lower bounds). Now, when ρ satisfies both E_1 and E_2 we have that $\Pr_y[C|_\rho(y) \neq GIP(y)] \geq 1/2 - n^{-\Omega(\log n)}$ by the multiparty communication complexity lower bound by Babai, Nisan, and Szegedy [4]. Since we can think of a random input x as being generated by first choosing a random restriction ρ and then choosing a random input y for the * of ρ (so that $C(x) = C|_\rho(y)$), we have that

$$\begin{aligned}
&\Pr_x[C(x) \neq f(x)] \\
&\geq \Pr_y\left[C|_\rho(y) \neq GIP(y) \;\middle|\; \rho \text{ satisfies } E_1 \text{ and } E_2\right] \cdot \Pr_\rho\left[\rho \text{ satisfies } E_1 \text{ and } E_2\right] \\
&\geq \left(1/2 - n^{-\Omega(\log n)}\right) \cdot \left(1 - n^{-\Omega(\log n)}\right) \\
&= 1/2 - n^{-\Omega(\log n)}.
\end{aligned}$$

We show that the above argument goes through for $\mathrm{SYM} \circ \mathrm{AC}^0$ circuits C of size $n^{\Omega(\log n)}$ and this proves our average-case hardness result for $\mathrm{SYM} \circ \mathrm{AC}^0$ circuits.

Circuits with more arbitrary symmetric gates. We now discuss how to extend our techniques to obtain an average-case hardness result for constant-depth circuits of size $n^{\epsilon \cdot \log n}$ with $\epsilon \log^2 n$ arbitrary symmetric gates (Theorem 3). The proof of this result has the same structure of our result for $\mathrm{SYM} \circ \mathrm{AC}^0$ circuits discussed in the previous subsection. The only difference is proving that if C is a small constant-depth circuit with $\epsilon \log^2 n$ arbitrary symmetric gates, then with high probability over a random restriction ρ the function computed by $C|_\rho$ is computable by a multiparty communication complexity protocol P among few parties communicating little (cf. event E_1 in the previous subsection). The idea is to let the protocol P compute the outputs of each arbitrary symmetric gate in order. Specifically, first fix a topological order of the arbitrary symmetric gates. (The simple order induced by reading the gates level by level from the inputs to the output node will do.) Now consider the $\mathrm{SYM} \circ \mathrm{AC}^0$ subcircuit C_1 whose root is the first arbitrary symmetric gate in this order. We know that with high probability over the restriction ρ, the function computed by $C_1|_\rho$ is computable by a multiparty communication complexity protocol P_1 exchanging few bits (cf. event E_1 in the previous subsection). Our protocol P first simulates P_1 to determine the output b_1 of $C_1|_\rho$. Then it considers the $\mathrm{SYM} \circ \mathrm{AC}^0$ circuit C_2 whose root is the second arbitrary symmetric gate, and where the first arbitrary symmetric gate is replaced with the constant b_1. Again, we argue that the function computed by $C_2|_\rho$ is computable by a multiparty communication complexity protocol P_2 exchanging few bits. Our protocol P now simulates P_2 to determine the output b_2 of $C_2|_\rho$. We continue in this way until all the arbitrary symmetric gates are computed.

Assuming without loss of generality that the output gate of the circuit is included in the arbitrary symmetric gates, the protocol P computes $C|_\rho$.

Comparison with the work by Chattopadhyay and Hansen. The work described in this paper proceeded at the same time as closely related work by Chattopadhyay and Hansen. These next paragraphs give the chronology of the papers that led to the current results, along with relevant citations. A similar discussion appears at the end of section 1 in [10].

An earlier version of this work, (I, unpublished, 2005) proved an average-case hardness result for constant-depth circuits with a constant number of arbitrary symmetric gates. This result was obtained by first proving an average-case hardness result for constant-depth circuits with *one* arbitrary symmetric gate (Theorem 4) and then combining this with a result by Beigel [6, Theorem 5.1] that shows that for every circuit of size S and depth d with σ arbitrary symmetric gates there is another circuit of size $S^{2^\sigma+1}$ and depth $d+1$ with *one* arbitrary symmetric gate at the top computing the same function.

Independently of (I), Chattopadhyay and Hansen [10, Theorem 1] proved a *worst-case* hardness result for constant-depth circuits of size $n^{\epsilon \cdot \log n}$ with $\epsilon \log^2 n$ arbitrary symmetric gates. (Note that the results in [10, Theorem 1] and (I) are incomparable.)

Subsequent to (I), Chattopadhyay and Hansen (II, personal communication, 2005) proved an average-case hardness result for constant-depth circuits of size $n^{\epsilon \cdot \log n}$ but having only $\epsilon \log n$ arbitrary symmetric gates. (Note that the average-case hardness result of Chattopadhyay and Hansen (II) does not directly imply their worst-case hardness result [10, Theorem 1], because it deals with a smaller number of arbitrary symmetric gates.)

The current paper takes inspiration from the average-case hardness result of Chattopadhyay and Hansen (II) and proves an average-case hardness result for constant-depth circuits of size $n^{\epsilon \cdot \log n}$ with $\epsilon \log^2 n$ arbitrary symmetric gates (Theorem 3); thus we obtain results that are stronger than those of either (I) or (II).

1.3. Organization. This paper is organized as follows. In section 2 we fix some notation. In section 3 we show how our average-case hardness result (Theorem 3) implies our generator (Theorem 1). In section 4 we prove our average-case hardness result for $\mathrm{SYM} \circ \mathrm{AC}^0$ circuits. In section 5 we extend this to our average-case hardness result for constant-depth circuits with few arbitrary symmetric gates, thus proving Theorem 3. In section 6 we elaborate on why our generator improves on the generator by Luby, Velickovic, and Wigderson (Theorem 2 in [19]). In section 7 we discuss some open problems.

2. Preliminaries. A *symmetric gate* is a gate that computes a symmetric function (i.e., a function whose value depends only on the number of input bits being 1, such as PARITY or MAJORITY). Our theorems hold regardless of which symmetric functions are computed by the gates; different functions can be used for circuits for inputs of different lengths, or even for different gates within the same circuit. To emphasize that our results hold *no matter what* symmetric functions are computed by the gates, we call them *arbitrary symmetric gates*. We use standard definitions of constant-depth circuits, which we now briefly recall. Constant-depth circuits consist of AND, OR, and possibly other gates (e.g., arbitrary symmetric gates). It is intended that all gates whose type is not specified are either AND or OR and that AND and OR gates are not counted toward arbitrary symmetric gates. All circuit gates, unless specified otherwise, have unbounded fan-in. Circuits take both input variables and their negations as input. *Bottom* gates are the one adjacent to the input bits. The

top gate is the output gate. Levels are numbered from the bottom. So the input bits are at level 0, the bottom gates at level 1, and so on. Gates at level i are connected to gates at levels $i-1$ and $i+1$ only. The *depth* of a circuit is the longest path from any input to the output. The *size* of a circuit is the number of gates in it. Multiple edges between pairs of nodes in the circuit are not allowed (otherwise an arbitrary symmetric gate can compute any function; this convention is standard in the literature, e.g., [15]).

3. From average-case hardness to pseudorandomness. In this section we show how our average-case hardness result (Theorem 3) implies our generator (Theorem 1). We restate the theorems for the reader's convenience.

THEOREM 1 (restated). *For every constant d there is a constant $\epsilon > 0$ such that for every l there is a generator $G : \{0,1\}^l \to \{0,1\}^m$, where $m = m(l) := l^{\epsilon \log l}$, such that for every circuit C of size m and depth d with $\log m(l)$ arbitrary symmetric gates, we have*

$$\left| \Pr_{x \in \{0,1\}^m}[C(x) = 1] - \Pr_{x \in \{0,1\}^l}[C(G(x)) = 1] \right| \leq \frac{1}{m},$$

and given $x \in \{0,1\}^l, i \leq m$, we can compute the ith output bit of $G(x)$ in time $\mathrm{poly}(l)$.

THEOREM 3 (restated). *There is a function $f : \{0,1\}^* \to \{0,1\}$ computable in polynomial time such that for every constant d there is a constant $\epsilon > 0$ such that for every n and every circuit C of size $n^{\epsilon \cdot \log n}$ and depth d with $\epsilon \log^2 n$ arbitrary symmetric gates, the following holds:*

$$\Pr_{x \in \{0,1\}^n}[C(x) \neq f(x)] \geq 1/2 - 1/n^{\epsilon \cdot \log n}.$$

Proof of Theorem 1*, assuming Theorem* 3. The generator is obtained by plugging the function from Theorem 3 into Nisan and Wigderson's pseudorandom generator construction [21]. Specifically, they show how given a function $f : \{0,1\}^{\sqrt{l/2}} \to \{0,1\}$ and a parameter m (which we set to be $m(l) := l^{\epsilon \cdot \log l}$) to construct a generator $G : \{0,1\}^l \to \{0,1\}^m$ such that every circuit C for which

$$\left| \Pr_{x \in \{0,1\}^m}[C(x) = 1] - \Pr_{x \in \{0,1\}^l}[C(G(x)) = 1] \right| > 1/m$$

can be transformed into another circuit C' of size $|C| + \mathrm{poly}(m)$ that computes the function f correctly with probability (over random input) greater than $1/2 + 1/m^2 = 1/2 + 1/l^{2\epsilon \log l}$.

As observed in [20, 21], C' is simply C with one more layer of AND (or OR) gates at the bottom, and possibly negating the output. Adding one layer of AND (or OR) gates at the bottom clearly does not increase the number of arbitrary symmetric gates in C, and negations can be removed using De Morgan's laws for AND and OR gates and absorbing them in the arbitrary symmetric gates. Thus, if C is a circuit of size $m = m(l) = l^{\epsilon \log l}$ of depth d with $\log m(l) = \epsilon \log^2 l$ arbitrary symmetric gates, we obtain another circuit C' of size $l^{O(\epsilon \log l)}$ of depth $d+1$ with $1 + \epsilon \log^2 l$ arbitrary symmetric gates that computes $f : \{0,1\}^{\sqrt{l/2}} \to \{0,1\}$ with probability greater than $1/2 + 1/l^{2\epsilon \log l}$. This contradicts Theorem 3 for sufficiently small ϵ.

The complexity of the generator follows from the arguments in [20, 21] and the fact that f is computable in time $\mathrm{poly}(l)$. □

4. Average-case hardness for SYM ∘ AC^0 circuits. In this section we prove our average-case hardness result for small constant-depth circuits with one arbitrary symmetric gate at the top.

THEOREM 4. *There is a function $f : \{0,1\}^* \to \{0,1\}$ computable in polynomial time such that for every constant d there is a constant $\epsilon > 0$ such that for every n and every circuit C of size $n^{\epsilon \cdot \log n}$ and depth d with 1 arbitrary symmetric gate at the top, the following holds:*

$$\Pr_{x \in \{0,1\}^n}[C(x) \neq f(x)] \geq 1/2 - 1/n^{\epsilon \cdot \log n}.$$

In the rest of this section we prove Theorem 4. In the proof we use two results which we describe in the following two subsections. The first is Håstad's switching lemma [14], and the second is the multiparty communication complexity lower bound for *GIP* by Babai, Nisan, and Szegedy [4].

4.1. Switching lemma. We now describe the switching lemma we use in the proof of Theorem 4. As in [13], the crucial property that we need is that the DNF obtained after applying the restriction is such that all the terms are mutually contradictory, i.e., no input satisfies more than one term. This allows us to merge the top OR gate of the DNF in the symmetric gate at the top (cf. Fact 6). That this property holds for Håstad's switching lemma was noted by Boppana and Håstad in [14] (inside the proof of Lemma 8.3). A proof of this fact appears in [9]. In this paper we instead use a version of Håstad's switching lemma which is due to Beame [5] and is inspired by a proof given by Razborov [24]. We prefer to use this version because its proof [5, 24] seems simpler than the proof in [14, 9]. A *restriction* on m variables $x_1, x_2, \ldots, x_m$ is a map $\rho : \{x_1, x_2, \ldots, x_m\} \to \{0, 1, *\}$. For a function $f : \{0,1\}^m \to \{0,1\}$ we denote by $f|_\rho$ the function we get by doing the substitutions prescribed by ρ. $f|_\rho$ will be a function of the variables that were given the value * by ρ. Similar conventions hold for circuits. If ρ and ρ' are restrictions, and ρ' is defined on the variables mapped to * by ρ, we write $\rho\rho'$ for the restriction obtained by combining ρ and ρ', so that $f|_{\rho\rho'} = (f|_\rho)|_{\rho'}$. Let $R_m^{\delta \cdot m}$ denote the uniform distribution on restrictions on m variables assigning exactly δm variables to $*$ and assigning random values to the others. A *decision tree* on m variables is a labeled binary tree where edges and leaves are labeled with 0 or 1 and internal nodes with variables. A decision tree computes a function in the intuitive way, starting at the root and following the path according to the values of the input variables and outputting the value at the reached leaf.

LEMMA 5 (see [5]). *Let φ be a DNF or a CNF formula in m variables with bottom fan-in at most r. For every $s \geq 0, p < 1/7$, the probability over $\rho \in R_m^{p \cdot m}$ that the function computed by $\varphi|_\rho$ is not computable by a decision tree of height strictly less than s is less than $(7pr)^s$.*

We will use Lemma 5 in combination with the following fact.

FACT 6. *Let f be a symmetric function of S decision trees of height h. Then f is computable by a depth-2 circuit of size $S \cdot 2^h + 1$ with a symmetric gate of unbounded fan-in at the top and AND gates of fan-in h at the bottom.*

Proof. Write each decision tree as a DNF with bottom fan-in h, where each term corresponds to a path leading to 1. The number of terms in each DNF is at most 2^h, i.e., at most the number of paths in a decision tree of height h. Because every input to a decision tree follows a unique path, each DNF we construct has the property that every input satisfies at most one term. Thus we can merge the top OR gate of all these DNFs with the top symmetric gate of the circuit. Specifically, if the original symmetric

gate was $\psi(x_1, x_2, \ldots, x_S) = g(\sum_{i \le S} x_i)$ for some arbitrary function $g : [S] \to \{0,1\}$, the new symmetric gate is simply $\overline{\psi}'(x_1, x_2, \ldots, x_{S \cdot 2^h}) := g(\sum_{i \le S \cdot 2^h} x_i)$. □

4.2. Multiparty communication complexity. In this section we describe some results on communication complexity that will be used in the proof of our main results. The model of interest is the *multiparty communication complexity model*. In this model there are s parties, each having unlimited computational power, who wish to collaboratively compute a certain function. The input bits to the function are partitioned in s blocks, and the ith party knows all the input bits except those corresponding to the ith block in the partition. The communication between the parties is by "writing on a blackboard" (broadcast): any bit sent by any party is seen by all the others. The parties exchange messages according to a fixed protocol. The measure of interest is the number of bits exchanged by the parties. See the book by Kushilevitz and Nisan [17] for background on this model.

Babai, Nisan, and Szegedy [4] proved a multiparty communication complexity lower bound for the generalized inner product function $GIP_{n,s} : \{0,1\}^{n \cdot s} \to \{0,1\}$, which is defined as follows:

$$GIP_{n,s}(x) := \bigoplus_{i=1}^{n} \bigwedge_{j=1}^{s} x_{i,j}.$$

LEMMA 7 (see [4]). *There is a partition of the inputs to $GIP_{n,s}$ in s blocks such that the following holds. Let P be an s-party communication complexity protocol exchanging at most $0.1 \cdot (n/4^s - \log(1/\gamma))$ bits of communication; then*

$$\Pr_{x \in \{0,1\}^{n \cdot s}} \Big[P(x) \neq GIP_{n,s}(x) \Big] \geq 1/2 - \gamma.$$

Håstad and Goldmann [15] showed that the function computed by a small SYM ∘ AND circuit with small bottom fan-in can be computed by a multiparty communication complexity protocol among few parties exchanging few bits.

LEMMA 8 (see [15]). *Let C be a depth-2 circuit of size S with an arbitrary symmetric gate (of unbounded fan-in) at the top and AND gates of fan-in strictly less than s at the bottom. Then the function computed by C can be computed (under any partition of the input) by an s-party communication complexity protocol exchanging $1 + s \log S$ bits.*

The idea in Lemma 8 is that since each bottom AND gate has fan-in strictly less than s, then, for any partition of the input in s blocks, the input bits to each AND can lie in at most $s - 1$ distinct blocks. Therefore we can assign each AND gate to some party that knows all the input bits necessary to compute it. Now each party broadcasts the number of AND gates assigned to him that evaluate to 1, which takes at most $\log S$ bits. Since the top gate is symmetric this information is sufficient to compute the output of the circuit.

Our next lemma combines the above observation by Håstad and Goldmann with the switching lemma results from the previous section to argue the following: for every small SYM ∘ AC^0 circuit, with high probability over a suitable restriction ρ, the function computed by $C|_\rho$ can be computed by a multiparty communication complexity protocol among few parties exchanging few bits.

LEMMA 9. *For every constant d there is a constant $\epsilon > 0$ such that the following holds. Let $C : \{0,1\}^n \to \{0,1\}$ be a circuit of size $n^{\epsilon \cdot \log n}$ and depth d with 1 arbitrary symmetric gate at the top. Let ρ be a random restriction on the n input variables*

*that assigns * to a subset of the variables of relative size* $1/n^{0.1}$*; i.e., let* $\rho \in R_n^{n/n^{0.1}}$*. Then with probability at least* $1 - n^{-\Omega(\log n)}$ *over* ρ*, the function computed by* $C|_\rho$ *is computable (under any partition of the input) by a* $0.3 \log n$*-party communication complexity protocol exchanging* $\log^3 n$ *bits of communication.*

Proof. The proof amounts to a combination of the previous lemmas for some specific setting of parameters. ∎

CLAIM 10. *With probability* $1-n^{-\Omega(\log n)}$ *over* $\rho \in R_n^{n/n^{0.1}}$*, the function computed by* $C|_\rho$ *is computable by a depth-2 circuit of size* $|C| \cdot 2^{0.3 \log n}$ *with a symmetric gate (of unbounded fan-in) at the top and AND gates of fan-in strictly less than* $0.3 \log n$ *at the bottom.*

The lemma follows by the above claim using Lemma 8, which implies that the function computed by a depth-2 circuit of size $S = |C| \cdot 2^{0.3 \log n} \leq n^{\log n}$ with a symmetric gate (of unbounded fan-in) at the top and AND gates of fan-in strictly less than $0.3 \log n$ at the bottom is computable by a $0.3 \log n$-party communication complexity protocol exchanging $1 + (0.3 \log n) \log S \leq \log^3 n$ bits.

We now prove Claim 10. Similar calculations have already been done elsewhere (e.g., Lemma 2 in [18]). However, we have not found the exact claim we need in the literature.

Proof of Claim 10. We see the restriction ρ as $d-1$ successive applications of restrictions $\rho_1, \rho_2, \ldots, \rho_{d-1}$ each mapping to * a subset of variables of relative size $1/n^\alpha$ of the (remaining) variables. Taking $\alpha = 0.1/(d-1)$ we have that, after applying all $d-1$ restrictions, the total number of variables mapped to * is $n \cdot (1/n^\alpha)^{d-1} = n/n^{0.1}$, and so this distribution on restrictions is exactly $R_n^{n/n^{0.1}}$.

For every $i \in [d-1]$ let DT_i be the event that, after applying the first i restrictions $\rho_1, \rho_2, \ldots, \rho_i$, the function computed by every gate at level i is computable by a decision tree of height strictly less than $0.3 \log n$. We now bound $\Pr_\rho[\text{ not } DT_{d-1}]$. Note that it is at most

$$\Pr_{\rho_1}[\text{ not } DT_1] + \Pr_{\rho_1,\rho_2}[\text{ not } DT_2 | DT_1] + \cdots + \Pr_{\rho_1,\rho_2,\ldots,\rho_{d-1}}[\text{ not } DT_{d-1} | DT_{d-2}].$$

We now bound each term. Fix any $i \leq d-1$ and consider $\Pr_{\rho_1,\rho_2,\ldots,\rho_i}[\text{ not } DT_i | DT_{i-1}]$. (If $i = 1$, think of the input variables as functions computed by decision trees of depth 1, and define $DT_0 :=$ TRUE.) Fix any gate φ at level i. Without loss of generality, assume φ is an OR gate (otherwise we can consider its negation, apply the same reasoning, and then negate again). Since we are conditioning over DT_{i-1}, all the functions computed by gates at level $i-1$ can be computed by decision trees of height (strictly) less than $0.3 \log n$. Write each such function as a DNF with terms of size at most $0.3 \log n$ (where each term corresponds to a path in the decision tree leading to 1). Merging the top OR gates of all these DNFs with φ we see that, given DT_{i-1}, the function computed by φ is a DNF with terms of size at most $r = 0.3 \log n$. By Lemma 5 the probability over the choice of the ith restriction ρ_i that the function computed by $\varphi|_{\rho_1\rho_2\cdots\rho_i}$ cannot be computed by a decision tree of depth strictly less than $s = 0.3 \log n$ is at most

$$(7pr)^s = (7 \cdot (1/n^\alpha) \cdot (0.3 \log n))^{0.3 \log n} = n^{-\Omega(\log n)}.$$

Thus by a union bound we have that

$$\Pr_{\rho_1,\rho_2,\ldots,\rho_i}[\text{ not } DT_i | DT_{i-1}]$$

is at most $n^{-\Omega(\log n)}$ times the number of gates at level i. Therefore, if the circuit C has size $n^{\epsilon \log n}$ for sufficiently small ϵ we have

$$\Pr_{\rho}[\text{ not } DT_{d-1}] \leq n^{-\Omega(\log n)} \cdot |C| = n^{-\Omega(\log n)}.$$

We have shown that with probability $1-n^{-\Omega(\log n)}$ (over ρ) the function computed by $C|_\rho$ is computable by a symmetric function of $|C|$ decision trees of height strictly less than $0.3 \log n$. By Fact 6 we can write each decision tree as a DNF and merge the top OR gates of these DNFs into the top symmetric gate of C, thus proving the claim. □

4.3. Proof of Theorem 4. We now prove Theorem 4. We restate the theorem for the reader's convenience.

THEOREM 4 (restated). *There is a function $f : \{0,1\}^* \to \{0,1\}$ computable in polynomial time such that for every constant d there is a constant $\epsilon > 0$ such that for every n and every circuit C of size $n^{\epsilon \cdot \log n}$ and depth d with 1 arbitrary symmetric gate at the top, the following holds:*

$$\Pr_{x \in \{0,1\}^n}[C(x) \neq f(x)] \geq 1/2 - 1/n^{\epsilon \cdot \log n}.$$

Proof of Theorem 4. Similarly to [22], we consider the function obtained by attaching PARITY gates on n bits at the bottom of $GIP_{n,0.3\log n}$. That is, let $f_n : \{0,1\}^{n^2(0.3\log n)} \to \{0,1\}$ be defined as

$$f_n(x) := \bigoplus_{i=1}^{n} \bigwedge_{j=1}^{0.3\log n} \bigoplus_{k=1}^{n} x_{i,j,k}.$$

We will prove Theorem 4 with f_n as the hard function. While f_n is a function on $m = m(n) := n^2(0.3 \log n)$ bits, it will be convenient to parameterize it by n. Since we will prove $n^{\Omega(\log n)}$ lower bounds for f_n and the input length of f_n is $m = \text{poly}(n)$, we also obtain $m^{\Omega(\log m)}$ lower bounds for f_n (for a different hidden constant in the $\Omega(\cdot)$).

It is easy to see that f_n is computable in polynomial time.

Let $C : \{0,1\}^m \to \{0,1\}$ be a circuit of size $n^{\epsilon \cdot \log n}$ and depth d with 1 arbitrary symmetric gate at the top, for a sufficiently small constant ϵ. Let ρ be a random restriction on the m input variables that assigns * to a subset of the variables of relative size $1/m^{0.1}$; i.e., let $\rho \in R_m^{m/m^{0.1}}$.

Consider the following two events:

- Event E_1 := the function computed by $C|_\rho$ is computable (under every partition of the input) by a $0.3 \log n$-party communication complexity protocol exchanging $n^{0.2}$ bits.
- Event E_2 := for every $i \in [n], j \in [0.3 \log n]$ there is $k \in [n]$ such that $\rho(x_{i,j,k}) = *$. (In other words, for each of the $n \cdot (0.3 \log n)$ bottom parity functions of f_n, ρ maps some of its input variable to *.)

CLAIM 11. $\Pr_{\rho \in R_m^{m/m^{0.1}}}[E_1 \wedge E_2] \geq 1 - n^{-\Omega(\log n)}$.

Before proving Claim 11 let us see how we can use it to prove Theorem 4. Suppose that some $\rho \in R_m^{m/m^{0.1}}$ satisfies both E_1 and E_2. Then

$$\Pr_{y \in \{0,1\}^{m/m^{0.1}}}[C|_\rho(y) \neq f_n|_\rho(y)] \geq 1/2 - n^{-\Omega(\log n)}. \tag{1}$$

This holds by Lemma 7. Specifically, fix any restriction ρ' taken on the variables mapped to * by ρ, such that for every $i \in [n], j \in [0.3 \log n]$ there is *exactly one* $k \in [n]$ such that $\rho\rho'(x_{i,j,k}) = *$. We then have that $f_n|_{\rho\rho'}$ equals $GIP_{n,0.3\log n}$ (up to possibly negating some input variables). If the function computed by $C|_\rho$ is computable by an s-party communication complexity protocol exchanging $n^{0.2}$ bits, then clearly the same holds for the function computed by $C|_{\rho\rho'}$. Therefore by the multiparty communication complexity lower bound for *GIP* (Lemma 7) we obtain (noticing that for $s = 0.3 \log n, \gamma = 2^{-n^{0.3}}$ we have $0.1 \cdot (n/4^s - \log(1/\gamma)) = \Omega(n^{0.4} - n^{0.3}) > n^{0.2}$)

$$\Pr_{z \in \{0,1\}^{n(0.3\log n)}}[C|_{\rho\rho'}(z) \neq f_n|_{\rho\rho'}(y)] \geq 1/2 - 1/2^{n^{\Omega(1)}} \geq 1/2 - n^{-\Omega(\log n)}.$$

Equation (1) follows, noticing that we can think of a random y as choosing first a random ρ' as above and then a random $z \in \{0,1\}^{n(0.3\log n)}$ for the *'s of ρ' (so that $C|_\rho(y) = C|_{\rho\rho'}(z)$).

Thus we have

$$\begin{aligned}
&\Pr_x[C(x) \neq f_n(x)] \\
&= \Pr_{\rho \in R_m^{m/m^{0.1}}, y \in \{0,1\}^{m/m^{0.1}}}[C|_\rho(y) \neq f_n|_\rho(y)] \\
&\geq \Pr_{\rho \in R_m^{m/m^{0.1}}, y \in \{0,1\}^{m/m^{0.1}}}[C|_\rho(x) \neq f_n|_\rho(x) | E_1 \wedge E_2] \cdot \Pr_{\rho \in R_m^{m/m^{0.1}}}[E_1 \wedge E_2] \\
&\geq \left(1/2 - n^{-\Omega(\log n)}\right) \cdot \left(1 - n^{-\Omega(\log n)}\right) \qquad \text{(by (1) and Claim 11)} \\
&= 1/2 - n^{-\Omega(\log n)},
\end{aligned}$$

which proves Theorem 4. □

It is only left to prove Claim 11.

Proof of Claim 11. We show that E_1 and E_2 each do not happen with probability at most $n^{-\Omega(\log n)}$.

The bound on $\Pr_\rho[\text{not } E_1]$ is given by Lemma 9. (The direct application of Lemma 9 gives communication complexity $\text{poly}\log(n) \ll n^{0.2}$ for circuits of size $m^{\epsilon \log m} \geq n^{\epsilon \log n}$.)

We now bound $\Pr_\rho[\text{not } E_2]$. Fix $i \in [n], j \in [0.3 \log n]$. The probability that for every $k \in [n]$ we have $\rho(x_{i,j,k}) \neq *$ is the probability that a random subset $A \subseteq [m]$ of size $m/m^{0.1} = m^{0.9}$ does not intersect a fixed subset $B \subseteq [m]$ of size n. This probability is at most the probability that $m^{0.9}$ *independent* random elements uniformly distributed in $[m]$ all fall outside B. (To see this, think of choosing the random subset A one element at a time, and note that when an element falls outside B it is more likely for the next element to fall inside B.) This latter probability is

$$\left(1 - \frac{n}{m}\right)^{m^{0.9}} \leq \exp(-m^{0.9} n/m) \leq \exp(-m^{\Omega(1)}) \ll n^{-\Omega(\log n)},$$

where we used that $m = n^2 \cdot (0.3 \log n)$. By a union bound we have

$$\Pr[\text{not } E_2] \leq n \cdot (0.3 \log n) \cdot n^{-\Omega(\log n)} = n^{-\Omega(\log n)}. \qquad \square$$

We point out that Theorem 4 is tight for the particular choice of

$$f_n(x) = \bigoplus_{i=1}^{n} \bigwedge_{j=1}^{0.3\log n} \bigoplus_{k=1}^{n} x_{i,j,k}.$$

Namely, f_n is computable by PARITY $\circ$ AND circuits of size $n^{O(\log n)}$. This can be seen by writing the function computed by each AND as a PARITY of $n^{O(\log n)}$ ANDs (cf. [22]).

5. Fooling circuits with more arbitrary symmetric gates. In this section we prove our average-case hardness result for constant-depth circuits of size $n^{\epsilon \log n}$ with $\epsilon \log^2 n$ arbitrary symmetric gates (Theorem 3). The proof has the same structure as the proof of our average-case hardness result for circuits with *one* arbitrary symmetric gate (Theorem 4). The only difference is that now we want to argue that event E_1 happens with high probability even for circuits with $\epsilon \log^2 n$ arbitrary symmetric gates; i.e., we want to show that with high probability over the restriction ρ, the function computed by $C|_\rho$ is computable by a multiparty communication complexity protocol among few parties exchanging few bits. Thus the proof of Theorem 3 follows from the next lemma.

LEMMA 12. *For every constant d there is a constant $\epsilon > 0$ such that the following holds. Let $C : \{0,1\}^n \to \{0,1\}$ be a circuit of size $n^{\epsilon \cdot \log n}$ and depth d with $\epsilon \log^2 n$ arbitrary symmetric gates. Let ρ be a random restriction on the n input variables that assigns * to a subset of the variables of relative size $1/n^{0.1}$; i.e., let $\rho \in R_n^{n/n^{0.1}}$. Then with probability at least $1 - n^{-\Omega(\log n)}$ over ρ, the function computed by $C|_\rho$ is computable (under every partition of the input) by a $0.3 \log n$-party communication complexity protocol exchanging $\log^5 n$ bits of communication.*

Proof. Assume without loss of generality that the output gate of the circuit C is included in the arbitrary symmetric gates. Fix a topological order of the arbitrary symmetric gates (the simple order induced by reading the gates level by level from the inputs to the output node will do). For every $i \in \{1, \ldots, \epsilon \log^2 n\}$, $z \in \{0,1\}^{i-1}$, define $C_{i,z}$ as the subcircuit of C whose output gate is the ith arbitrary symmetric gate but where the previous arbitrary symmetric gates are replaced with z (i.e., the jth gate is replaced with the jth bit in z). Note that $C_{i,z}$ is a SYM $\circ$ AC^0 circuit. □

CLAIM 13. *For a sufficiently small constant $\epsilon > 0$, with probability $1 - n^{-\Omega(\log n)}$ over $\rho \in R_n^{n/n^{0.1}}$ we have that for every $i \in \{1, \ldots, \epsilon \log^2 n\}$ and $z \in \{0,1\}^{i-1}$ the function computed by $C_{i,z}|_\rho$ is computable (under every partition of the input) by a $0.3 \log n$-party communication complexity protocol $P_{i,z}$ exchanging $\log^3 n$ bits of communication.*

Proof. The claim follows by noting that the number of SYM $\circ$ AC^0 circuits $C_{i,z}$ is at most

$$\left(\epsilon \log^2 n\right) \cdot 2^{\epsilon \log^2 n} \leq n^{1+\epsilon \log n},$$

and then using a union bound and Lemma 9, which states that for each fixed circuit $C_{i,z}$, with probability $1 - n^{-\Omega(\log n)}$ over ρ, the function computed by $C_{i,z}|_\rho$ is computable by a $0.3 \log n$-party communication complexity protocol exchanging $\log^3 n$ bits. □

The lemma follows by noting that whenever ρ satisfies the conclusion of the above claim we have (under any partition of the input bits) the following $0.3 \log n$-party communication complexity protocol P for $C|_\rho$. On input x compute $C|_\rho(x)$ as follows. Simulate P_1 to compute $b_1 = C_1|_\rho(x)$. Then simulate P_{2,b_1} to compute $b_2 = C_{2,b_1}|_\rho(x)$. Then simulate $P_{3,b_1 \circ b_2}$ to compute $b_3 = C_{3,b_1 \circ b_2}|_\rho(x)$. Continue in this way until $C_{\epsilon \log^2 n, z}(x) = C|_\rho(x)$. (This last equality is easy to verify.)

Since each protocol $P_{i,z}$ exchanges at most $\log^3 n$ bits of communication, and we simulate $\epsilon \log^2 n$ of these protocols, the total number of bits exchanged by the protocol P is at most $\log^5 n$.

It is perhaps interesting to note that, unlike the corresponding protocol in the proof of Theorem 4, the protocol in the above lemma is not simultaneous; i.e., the bits sent by a party in general depend on the bits previously sent by other parties (cf. [17] for background on simultaneous protocols). Thus in our proof we are taking advantage of the fact that the lower bound for *GIP* (Lemma 7) holds even for non-simultaneous protocols. We do not know how to prove the same result starting from a multiparty communication complexity lower bound for simultaneous protocols.

6. Our generator versus Luby, Velickovic, and Wigderson's. In this section we elaborate on why our generator (Theorem 1) improves on the generator by Luby, Velickovic, and Wigderson (Theorem 2 in [19]). Recall that the generator in [19] fools small depth-2 circuits with one arbitrary symmetric gate at the top (SYM $\circ$ AND circuits). On the other hand, our generator fools small circuits of any constant depth with few arbitrary symmetric gates.

We note that there are several results (e.g., [23, 27, 2, 33, 7, 8]) showing that small circuits in certain rich constant-depth circuit classes can be converted into not-too-big SYM $\circ$ AND circuits. Thus one may wonder whether we can use these results to deduce that the generator in [19] is already powerful enough to give our main result (Theorem 1), i.e., whether it can fool small constant-depth circuits with few arbitrary symmetric gates.

The problem with this idea is that in all these conversion results, *the blow-up in the circuit size is bigger than the saving of the generator.* More specifically, these conversion results show how to convert, say, an AC^0 circuit of size S into a SYM $\circ$ AND circuit of size *quasi*-polynomial, i.e., $S^{\log^{O(1)} S}$, where the constant in the $O(1)$ depends on the depth of the original circuit. However, to fool a circuit of size $S^{\log^{O(1)} S}$, the generator in [19] needs a seed of length at least S, and therefore it is of no use in this particular setting.

It seems natural to ask whether the known conversion results are the best possible, i.e., if the quasi-polynomial blow-up is inherent in the conversion. There are works (e.g., [2, 22]) suggesting that this is indeed the case. We give another result of this flavor.

Specifically, we show how to modify the lower bound in Theorem 4 to get a function computable by polynomial-size PARITY $\circ$ AC^0 circuits that is average-case hard for superpolynomial-size SYM $\circ$ AND circuits. The idea is to change the fan-in of the bottom parities of f so that they are computable by polynomial-size AC^0 circuits (specifically, we change their fan-in from n to $\log^3 n$). While our lower bound is only slightly superpolynomial (i.e., $n^{\Omega(\log\log n)}$), it shows that the parameters of our generator (Theorem 1) *cannot* be obtained combining a conversion result with Theorem 2 in [19], even if we only want to fool PARITY $\circ$ AC^0 circuits.

THEOREM 14. *There is a function $f : \{0,1\}^* \to \{0,1\}$ computable by polynomial-time uniform polynomial-size* PARITY $\circ$ AC^0 *circuits and a constant $\epsilon > 0$ such that for every n and every* SYM $\circ$ AND *circuit C of size $n^{\epsilon \cdot \log\log n}$, the following holds:*

$$\Pr_{x \in \{0,1\}^n}[C(x) \neq f(x)] \geq 1/2 - 1/n^{\epsilon \cdot \log\log n}.$$

Proof of Theorem 14. The proof follows closely the proof of Theorem 4. Let

$$g_n : \{0,1\}^{n(0.3\log n)\log^3 n} \to \{0,1\}$$

be defined as

$$g_n(x) := \bigoplus_{i=1}^{n} \bigwedge_{j=1}^{0.3\log n \log^3 n} \bigoplus_{k=1} x_{i,j,k}.$$

We will prove Theorem 14 with g_n as the hard function. While g_n is a function on $m = m(n) := n \cdot (0.3 \log n) \cdot (\log^3 n)$ bits, it will be convenient to parameterize it by n. Since we will prove $n^{\Omega(\log \log n)}$ lower bounds for g_n and the input length of g_n is $m = n \cdot \operatorname{poly} \log n$, we also obtain $m^{\Omega(\log \log m)}$ lower bounds for g_n (for a different hidden constant in the $\Omega(\cdot)$).

Note that g_n is computable by a (uniform) polynomial-size circuit of depth 5 with one PARITY gate at the top. To see this, note that each of the bottom parities in the definition of g_n is only on $\log^3 n$ bits, and therefore it can be computed by a (uniform) circuit of size $\operatorname{poly}(n)$ and depth 4 (see, e.g., [14, Theorem 2.2]).

Let $C : \{0,1\}^m \to \{0,1\}$ be a SYM $\circ$ AND circuit of size $n^{\epsilon \cdot \log \log n}$ for a sufficiently small constant ϵ. Let ρ be a random restriction on the m input variables that assigns * to a subset of the variables of relative size $1/\log(n)$; i.e., let $\rho \in R_m^{m/\log(n)}$.

Consider the following two events:

- Event E_1' := the function computed by $C|_\rho$ is computable (under any partition of the input) by a $0.3 \log n$-party communication complexity protocol exchanging $n^{0.2}$ bits.
- Event E_2' := for every $i \in [n], j \in [0.3 \log n]$ there is $k \in [\log^3 n]$ such that $\rho(x_{i,j,k}) = *$. (In other words, for each of the $n \cdot (0.3 \log n)$ bottom parity functions of f_n, ρ maps some of its input variable to *.)

As before, Theorem 14 follows from the next claim (cf. the proof of Theorem 4). □

CLAIM 15. $\Pr_{\rho \in R_m^{m/\log(n)}}[E_1' \wedge E_2'] \geq 1 - n^{-\Omega(\log \log n)}$.

Proof. We show that E_1' and E_2' each do not happen with probability at most $n^{-\Omega(\log \log n)}$.

We now bound $\Pr_\rho[\text{not } E_1']$. Analogously to the proof of Lemma 9, the main step is proving the following claim: with high probability $\left(1 - n^{-\Omega(\log \log n)}\right)$ over $\rho \in R_m^{m/\log(n)}$, the function computed by $C|_\rho$ is computable by a depth-2 circuit of size $|C| \cdot 2^{0.3 \log n} = n^{\epsilon \cdot \log \log n} \cdot 2^{0.3 \log n}$ with a single symmetric gate (of unbounded fan-in) at the top and AND gates of fan-in strictly less than $0.3 \log n$ at the bottom. While this probability can be bounded directly, similarly to what is done in [22], it seems simpler to use again the switching lemma. Fix a bottom AND gate φ of C, and think of the input variables to φ as clauses of size $r = 1$. By Lemma 5 the probability over the choice of ρ that the function computed by $\varphi|_\rho$ cannot be computed by a decision tree of depth strictly less than $s = 0.3 \log n$ is at most

$$(7pr)^s = (7 \cdot (1/\log(n)) \cdot 1)^{0.3 \log n} = n^{-\Omega(\log \log n)}.$$

Therefore if the circuit C has size $n^{\epsilon \log \log n}$ for sufficiently small ϵ we have, by a union bound, that with probability $1 - n^{-\Omega(\log \log n)}$ (over ρ) the function computed by $C|_\rho$ is computable by a symmetric function of $|C|$ decision trees of height strictly less than $0.3 \log n$. By Fact 6 we can write each decision tree as a DNF and merge the top OR gates of these DNFs into the top symmetric gate of C, and thus E_1' holds. We now bound $\Pr_\rho[\text{not } E_2']$. Fix $i \in [n], j \in [0.3 \log n]$. The probability that for every $k \in [\log^3 n]$ we have $\rho(x_{i,j,k}) \neq *$ is the probability that a random subset $A \subseteq [m]$ of size $m/\log(n)$ does not intersect a fixed subset $B \subseteq [m]$ of size $\log^3 n$. This probability is at most the probability that $m/\log(n)$ *independent* random elements uniformly distributed in $[m]$ all fall outside B. (To see this, think of choosing the

random subset A one element at a time, and note that when an element falls outside B it is more likely for the next element to fall inside B.) This latter probability is

$$\left(1 - \frac{\log^3 n}{m}\right)^{m/\log(n)} \leq \exp(-\Omega(\log^2 n)) \ll n^{-\Omega(\log\log n)}.$$

By a union bound we have

$$\Pr[\text{not } E_2'] \leq n \cdot (0.3 \log n) \cdot n^{-\Omega(\log\log n)} = n^{-\Omega(\log\log n)}. \qquad \square$$

7. Open problems. Can we improve the stretch of our generator? We remark that an explicitly computable generator $G : \{0,1\}^l \to \{0,1\}^m$ that fools circuits of size m in some class $\mathcal{C}$ (e.g., $\mathcal{C} = \text{SYM} \circ \text{AC}^0$) implies the existence of an explicit function $f : \{0,1\}^{l+1} \to \{0,1\}$ that requires $\mathcal{C}$ circuits of size m (see, e.g., [21]). Thus, improving the stretch of our generator requires establishing stronger lower bounds on the size of circuits computing explicit functions, a notoriously difficult problem. In particular, using a famous result by Yao [33] and Beigel and Tarui [8], we obtain that improving the stretch of our generator to $m(l) = l^{\log^{\omega(1)} l}$, even if we only want to fool $\text{SYM} \circ \text{AND}$ circuits, would imply the existence of an explicit function that cannot be computed by polynomial-size constant-depth circuits with MOD_a gates for constant a (i.e., ACC^0 circuits), thus answering a long-standing open question (cf. [17, section 11.4]). We note that the size lower bound obtained with the approach in [15, 22, 13] and in this paper is limited by the number of players in the known multiparty communication complexity lower bounds (cf. [17, Problems 6.21]).

A direction that seems to have been investigated less is that of trying to increase the number of arbitrary symmetric gates in our results, which is currently logarithmic in the circuit size. Specifically, we ask whether the techniques in this paper can be used to prove (average-case) hardness results for small constant-depth circuits with $\omega(\log^2 n)$ arbitrary symmetric gates. Such a hardness result would follow from a positive answer to the following open question. Let C be a constant-depth circuit of size $n^{\epsilon \log n}$ with $\omega(\log^2 n)$ arbitrary symmetric gates, and let ρ be a restriction as in the statement of Lemma 9. Is it true that with high probability over ρ the function computed by $C|_\rho$ is computable by a $0.9 \log n$-party communication complexity protocol exchanging $n^{0.9}$ bits?

Acknowledgments. We thank Salil Vadhan for his helpful reading of this paper. We thank Arkadev Chattopadhyay and Kristoffer Arnsfelt Hansen for sending us their paper [10] and the anonymous referees for helpful comments.

REFERENCES

[1] M. Agrawal, *Hard sets and pseudo-random generators for constant depth circuits*, in Proceedings of the 21st Annual Foundations of Software Technology and Theoretical Computer Science, Bangalore, India, Springer-Verlag, Berlin, 2001, pp. 58–69.

[2] E. Allender and U. Hertrampf, *Depth reduction for circuits of unbounded fan-in*, Inform. and Comput., 112 (1994), pp. 217–238.

[3] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson, *BPP has subexponential time simulations unless EXPTIME has publishable proofs*, Comput. Complexity, 3 (1993), pp. 307–318.

[4] L. Babai, N. Nisan, and M. Szegedy, *Multiparty protocols, pseudorandom generators for logspace, and time-space trade-offs*, J. Comput. System Sci., 45 (1992), pp. 204–232.

[5] P. Beame, *A Switching Lemma Primer*, Technical report UW-CSE-95-07-01, Department of Computer Science and Engineering, University of Washington, Seattle, WA, 1994; also available online from http://www.cs.washington.edu/homes/beame/.

[6] R. Beigel, *When do extra majority gates help?* polylog(N) *majority gates are equivalent to one*, Comput. Complexity, 4 (1994), pp. 314–324.

[7] R. Beigel, N. Reingold, and D. A. Spielman, *The perceptron strikes back*, in Proceedings of the Structure in Complexity Theory Conference, 1991, pp. 286–291.
[8] R. Beigel and J. Tarui, *On ACC*, Comput. Complexity, 4 (1994), pp. 350–366.
[9] C. Berg and S. Ulfberg, *A lower bound for perceptrons and an oracle separation of the* $\mathrm{PP}^{\mathrm{PH}}$ *hierarchy*, J. Comput. System Sci., 56 (1998), pp. 263–271.
[10] A. Chattopadhyay and K. A. Hansen, *Lower bounds for circuits with few modular and symmetric gates*, in Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP), Lisboa, Portugal, Lecture Notes in Comput. Sci. 3580, L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, eds., Springer-Verlag, Berlin, 2005.
[11] M. L. Furst, J. B. Saxe, and M. Sipser, *Parity, circuits, and the polynomial-time hierarchy*, Math. Systems Theory, 17 (1984), pp. 13–27.
[12] A. Hajnal, W. Maass, P. Pudlák, M. Szegedy, and G. Turán, *Threshold circuits of bounded depth*, J. Comput. System Sci., 46 (1993), pp. 129–154.
[13] K. A. Hansen and P. B. Miltersen, *Some meet-in-the-middle circuit lower bounds*, in Proceedings of the 29th International Symposium on Mathematical Foundations of Computer Science (MFCS), Lecture Notes in Computer Science 3153, Springer-Verlag, Berlin, 2004, pp. 334–345.
[14] J. Håstad, *Computational Limitations of Small-Depth Circuits*, MIT Press, Cambridge, MA, 1987.
[15] J. Håstad and M. Goldmann, *On the power of small-depth threshold circuits*, Comput. Complexity, 1 (1991), pp. 113–129.
[16] A. Healy, S. P. Vadhan, and E. Viola, *Using nondeterminism to amplify hardness*, SIAM J. Comput., 35 (2006), pp. 903–931.
[17] E. Kushilevitz and N. Nisan, *Communication Complexity*, Cambridge University Press, Cambridge, UK, 1997.
[18] N. Linial, Y. Mansour, and N. Nisan, *Constant depth circuits, Fourier transform, and learnability*, J. Assoc. Comput. Mach., 40 (1993), pp. 607–620.
[19] M. Luby, B. Velickovic, and A. Wigderson, *Deterministic approximate counting of depth-2 circuits*, in Proceedings of the Second Annual Israel Symposium on Theory of Computing Systems (ISTCS), 1993, pp. 18–24.
[20] N. Nisan, *Pseudorandom bits for constant depth circuits*, Combinatorica, 11 (1991), pp. 63–70.
[21] N. Nisan and A. Wigderson, *Hardness versus randomness*, J. Comput. System Sci., 49 (1994), pp. 149–167.
[22] A. Razborov and A. Wigderson, $n^{\Omega(\log n)}$ *lower bounds on the size of depth-3 threshold circuits with AND gates at the bottom*, Inform. Process. Lett., 45 (1993), pp. 303–307.
[23] A. A. Razborov, *Lower bounds on the dimension of schemes of bounded depth in a complete basis containing the logical addition function*, Mat. Zametki, 41 (1987), pp. 598–607, 623.
[24] A. A. Razborov, *Bounded arithmetic and lower bounds in Boolean complexity*, in Feasible Mathematics, II, Progr. Comput. Sci. Appl. Logic 13, Birkhäuser Boston, Boston, 1995, pp. 344–386.
[25] A. A. Razborov and S. Rudich, *Natural proofs*, J. Comput. System Sci., 55 (1997), pp. 24–35.
[26] R. Shaltiel and C. Umans, *Simple extractors for all min-entropies and a new pseudorandom generator*, J. ACM, 52 (2005), pp. 172–216.
[27] R. Smolensky, *Algebraic methods in the theory of lower bounds for Boolean circuit complexity*, in Proceedings of the 19th Annual ACM Symposium on Theory of Computing, New York, 1987, pp. 77–82.
[28] M. Sudan, L. Trevisan, and S. Vadhan, *Pseudorandom generators without the XOR lemma*, J. Comput. System Sci., 62 (2001), pp. 236–266.
[29] C. Umans, *Pseudo-random generators for all hardnesses*, J. Comput. System Sci., 67 (2003), pp. 419–440.
[30] E. Viola, *The complexity of constructing pseudorandom generators from hard functions*, Comput. Complexity, 13 (2004), pp. 147–188.
[31] E. Viola, *Pseudorandom bits for constant-depth circuits with few arbitrary symmetric gates*, in Proceedings of the 20th Annual Conference on Computational Complexity, IEEE, 2005, pp. 198–209.
[32] E. Viola, *The Complexity of Hardness Amplification and Derandomization*, Ph.D. thesis, Harvard University, Cambridge, MA, 2006.
[33] A. C. Yao, *On ACC and threshold circuits*, in Proceedings of the 31st Annual IEEE Symposium Foundation on Computer Science, 1990, pp. 619–627.

SIAM J. COMPUT.
Vol. 36, No. 5, pp. 1404–1434

LOCALLY DECODABLE CODES WITH TWO QUERIES AND POLYNOMIAL IDENTITY TESTING FOR DEPTH 3 CIRCUITS*

ZEEV DVIR† AND AMIR SHPILKA‡

Abstract. In this work we study two, seemingly unrelated, notions. *Locally decodable codes* (LDCs) are codes that allow the recovery of each message bit from a constant number of entries of the codeword. *Polynomial identity testing* (PIT) is one of the fundamental problems of algebraic complexity: we are given a circuit computing a multivariate polynomial and we have to determine whether the polynomial is identically zero. We improve known results on LDCs and on polynomial identity testing and show a relation between the two notions. In particular we obtain the following results: (1) We show that if $E : \mathbb{F}^n \mapsto \mathbb{F}^m$ is a linear LDC with two queries, then $m = \exp(\Omega(n))$. Previously this was known only for fields of size $\ll 2^n$ [O. Goldreich et al., *Comput. Complexity*, 15 (2006), pp. 263–296]. (2) We show that from every depth 3 arithmetic circuit ($\Sigma\Pi\Sigma$ circuit), $\mathcal{C}$, with a bounded (constant) top fan-in that computes the zero polynomial, one can construct an LDC. More formally, assume that $\mathcal{C}$ is minimal (no subset of the multiplication gates sums to zero) and simple (no linear function appears in all the multiplication gates). Denote by d the degree of the polynomial computed by $\mathcal{C}$ and by r the rank of the linear functions appearing in $\mathcal{C}$. Then we can construct a linear LDC with two queries that encodes messages of length $r/\text{polylog}(d)$ by codewords of length $O(d)$. (3) We prove a structural theorem for $\Sigma\Pi\Sigma$ circuits, with a bounded top fan-in, that compute the zero polynomial. In particular we show that if such a circuit is simple, minimal, and of polynomial size, then its rank, r, is only polylogarithmic in the number of variables (a priori it could have been linear). (4) We give new PIT algorithms for $\Sigma\Pi\Sigma$ circuits with a bounded top fan-in: (a) a deterministic algorithm that runs in quasipolynomial time, and (b) a randomized algorithm that runs in polynomial time and uses only a polylogarithmic number of random bits. Moreover, when the circuit is multilinear, our deterministic algorithm runs in polynomial time. Previously deterministic subexponential time algorithms for PIT in bounded depth circuits were known only for depth 2 circuits (in the black box model) [D. Grigoriev, M. Karpinski, and M. F. Singer, *SIAM J. Comput.*, 19 (1990), pp. 1059–1063; M. Ben-Or and P. Tiwari, *Proceedings of the* 20*th Annual ACM Symposium on Theory of Computing*, ACM Press, New York, 1988, pp. 301–309; A. R. Klivans and D. Spielman, *Proceedings of the* 33*rd Annual ACM Symposium on Theory of Computing*, ACM Press, New York, 2001, pp. 216–223]. In particular, for the special case of depth 3 circuits with three multiplication gates our result resolves an open question asked by Klivans and Spielman.

Key words. derandomization, polynomial identity test, arithmetic circuits, depth 3, locally decodable codes

AMS subject classifications. 68Q25, 94B65

DOI. 10.1137/05063605X

1. Introduction.

Locally decodable codes (LDCs) are error correcting codes that allow the recovery of each symbol of the message from a constant number of entries of the codeword. Polynomial identity testing (PIT) is one of the fundamental problems of algebraic complexity: we are given a circuit computing a multivariate polynomial, and we have to determine whether the polynomial is identically zero. In this paper we show a relation between these two notions—roughly, from every depth 3 circuit which is identically zero, one can construct an LDC. Using this relation and a new lower bound on LDCs, we devise new PIT algorithms for depth 3 circuits.

*Received by the editors March 14, 2006; accepted for publication (in revised form) July 14, 2006; published electronically January 26, 2007.

http://www.siam.org/journals/sicomp/36-5/63605.html

†Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel (zeev.dvir@weizmann.ac.il).

‡Faculty of Computer Science, Technion, Haifa, Israel (shpilka@cs.technion.ac.il).

1.1. Locally decodable codes. LDCs are error correcting codes that allow the recovery of each symbol of the message, from a corrupted codeword, by looking at only a constant number of entries of the corrupted word. Roughly, a (q, δ, ϵ)-*locally decodable code* encodes $x \in \mathbb{F}^n$ to $E(x) \in \mathbb{F}^m$ such that for each index $i \in [n]$, x_i can be recovered from $E(x)$ with probability[1] $> \frac{1}{|\mathbb{F}|} + \epsilon$ by reading only q (random) entries, even if $E(x)$ was corrupted in δm positions.

LDCs have many applications—they are related to private information retrieval (PIR) schemes [13, 26, 18], and they can be used for amplification of hardness [19, 20, 3] and for the construction of hard-core predicates for one-way permutations [30, 15]. (See [49] for a survey on LDCs.)

The notion of LDCs was explicitly discussed in [4] and explicitly defined in [26]. Implicit constructions of local decoders can be found in the context of random self-reducibility and self-correcting computations (see, e.g., [32, 6, 16, 17, 15]). There are two main questions related to LDCs: finding explicit constructions and proving limits of such constructions (i.e., proving lower bounds on the length of the encoding). Explicit constructions were given by [4, 7, 8]. The best current construction is due to Beimel et al. [8], who gave an LDC with q queries of length $m = \exp(n^{O(\log\log q/q \log q)})$.

The problem of proving lower bounds was first studied by Katz and Trevisan [26], who proved that for every LDC with q queries, the length of the codeword, m, is at least $n^{1+\frac{1}{q-1}}$. This is currently the best lower bound for general LDCs (see also [14]). It is a very challenging open question to give tight lower bounds (or upper bounds) on the length of LDCs. Due to the difficulty of the problem many works focused on the case of codes with two queries ($q = 2$). Exponential lower bounds were first proved for linear codes [18, 37] and then, by techniques from quantum computation, for nonlinear codes over $GF(2)$ [28]. The bound of Goldreich et al. [18] actually holds for linear LDCs with two queries over any finite field, namely, that m is at least $2^{\Omega(n)-\log(|\mathbb{F}|)}$, where $\mathbb{F}$ is the underlined field. This result is (nearly) tight when the field is of constant size; however, it gives no significant bound for infinite fields.

1.2. Polynomial identity testing. PIT is a fundamental problem in algebraic complexity: we are given a multivariate polynomial (in some representation) over some field $\mathbb{F}$, and we have to determine whether it is identically zero.[2] The importance of this problem follows from its many applications: algorithms for primality testing [1, 2], for deciding if a graph contains a perfect matching [33, 34, 11], and more, are based on reductions to the PIT problem. (See the introduction of [31] for more applications.)

Determining the complexity of PIT is one of the greatest challenges of theoretical computer science. It is one of a few problems (and in some sense PIT is the most general problem) for which we have *coRP* algorithms but no deterministic subexponential time algorithms. Kabanets and Impagliazzo [25] suggested an explanation for the lack of algorithms. They showed that efficient deterministic algorithms for PIT imply that $NEXP$ does not have polynomial size arithmetic circuits. Specifically, if PIT has deterministic polynomial time algorithms, then either the permanent cannot be computed by polynomial size arithmetic circuits or $NEXP \not\subset P/\text{poly}$.

The first randomized algorithm for PIT was discovered independently by Schwartz [42] and Zippel [50]. Their well-known algorithm simply evaluates the polynomial at a random point and accepts iff the polynomial vanishes at the point. If the polynomial

[1] If $\mathbb{F}$ is infinite, then the probability of success is $> \epsilon$.

[2] Note that we want the polynomial to be identically zero and not just to be equal to the zero function. For example, $x^2 - x$ is the zero function over $GF(2)$ but not the zero polynomial.

is of degree d and each variable is randomly chosen from a domain S, then the error probability is bounded by $d/|S|$. Two kinds of works followed the Schwartz–Zippel algorithm: randomized algorithms that use fewer random bits [12, 31, 1] and algorithms for restricted models of arithmetic circuits. In [22, 9, 29] polynomial time deterministic PIT algorithms for depth 2 arithmetic circuits were given. More recently, [41] gave a polynomial time PIT algorithm for noncommutative formulas. All algorithms, with the exception of [1, 41], are black box algorithms. That is, these algorithms do not have access to a circuit computing the polynomial, and they can evaluate it only on different inputs (as in the Schwartz–Zippel algorithm).

A result of a different nature was proved by Kabanets and Impagliazzo [25]. They designed a deterministic quasi-polynomial time algorithm based on unproved hardness assumptions. Namely, in Theorem 7.7 of [25] it is shown that if there is a family $\{p_n\}$ of exponential time computable polynomials in n variables over $\mathbb{Z}$ such that the arithmetic circuit complexity of p_n is $\exp(n^{\Omega(1)})$, then there is an $\exp(\text{poly}(\log n))$ time algorithm for identity testing for any polynomial size arithmetic circuit that computes polynomials with at most a polynomial degree and polynomial size coefficients.

1.3. Depth 3 arithmetic circuits. Proving lower bounds for general arithmetic circuits is the greatest challenge of algebraic complexity. Unfortunately, except for the lower bounds of Strassen [47] and Baur and Strassen [5], no lower bounds are known for general arithmetic circuits. Due to the difficulty of the problem, research focused on restricted models such as monotone circuits and bounded depth circuits. Exponential lower bounds were proved on the size of monotone arithmetic circuits [43, 24], and linear lower bounds were proved on their depth [44, 48]. However, unlike the situation in the Boolean case, only weak lower bounds were proved for bounded depth arithmetic circuits [38, 40]. Thus, a more restricted model was considered—the model of depth 3 arithmetic circuits (also known as $\Sigma\Pi\Sigma$ circuits). A $\Sigma\Pi\Sigma$ circuit computes a polynomial of the form

$$\mathcal{C} = \sum_{i=1}^{k} \prod_{j=1}^{d_i} L_{ij}(x), \tag{1}$$

where the L_{ij} are linear functions. Grigoriev and Karpinski [21] and Grigoriev and Razborov [23] proved exponential lower bounds on the size of $\Sigma\Pi\Sigma$ circuits computing the permanent and determinant over finite fields. Over infinite fields exponential lower bounds are known only for the restricted models of *multilinear*[3] $\Sigma\Pi\Sigma$ circuits and for *homogeneous* $\Sigma\Pi\Sigma$ circuits [35, 36]. For general $\Sigma\Pi\Sigma$ circuits over infinite fields only the quadratic lower bound of [46] is known. Thus, proving exponential lower bounds for $\Sigma\Pi\Sigma$ circuits over $\mathbb{C}$ is a major open problem in arithmetic circuit complexity.

In this work we are interested in the problem of PIT for depth 3 circuits. As mentioned earlier there are no efficient PIT algorithms for arithmetic circuits, even if we just consider bounded depth circuits. Thus, finding efficient algorithms for PIT in $\Sigma\Pi\Sigma$ circuits seems like the first step toward proving more general results.

1.4. Our results. Lower bounds for linear LDCs with two queries. We study linear LDCs with two queries over arbitrary fields and prove lower bounds on their length. The first such lower bound was proved by Goldreich et al. [18], as follows.

[3]More accurately for pure multilinear $\Sigma\Pi\Sigma$ circuits.

THEOREM 1.1 (Theorem 1.4 of [18]). *Let $\delta, \epsilon \in [0,1]$, $\mathbb{F}$ be a field, and let $E : \mathbb{F}^n \to \mathbb{F}^m$ be a linear $(2, \delta, \epsilon)$-LDC. Then*

$$m \geq 2^{\frac{\epsilon\,\delta\,n}{16} - 1 - \log_2 |\mathbb{F}|}.$$

Note that this result makes sense only when $|\mathbb{F}|$ is finite. We prove the following theorem.

THEOREM 1.2. *Let $\delta, \epsilon \in [0,1]$, $\mathbb{F}$ be a field, and let $E : \mathbb{F}^n \to \mathbb{F}^m$ be a linear $(2, \delta, \epsilon)$-LDC. Then*

$$m \geq 2^{\frac{\epsilon\,\delta\,n}{4} - 1}.$$

Compared with Theorem 1.4 of [18], our result removes the dependance on the size of the field in the exponent and works for every field size, finite and infinite. The idea of the proof is similar to the one in [18]—we show that, given a linear 2-LDC over an arbitrary field $\mathbb{F}$, we can construct from it a linear 2-LDC over $GF(2)$, with almost the same parameters, and then we use the lower bound of [18] for codes over $GF(2)$.

Relation between depth 3 circuits and LDCs. The main result of the paper is that from every $\Sigma\Pi\Sigma$ circuit that computes the zero polynomial, one can construct a linear LDC with two queries. Relations between arithmetic circuits and error correcting codes were known before [10, 45]; however, this is the first time that LDCs appear in the context of arithmetic circuits. More formally, let $\mathcal{C}$ be a $\Sigma\Pi\Sigma$ circuit, as in (1), computing the zero polynomial. We say that $\mathcal{C}$ is minimal if no proper subset of the multiplication gates sums to zero. We say that $\mathcal{C}$ is simple if there is no linear function that appears in all the multiplication gates (up to a multiplicative constant). Denote with r the rank of the linear functions appearing in $\mathcal{C}$.

THEOREM 1.3. *Let $k \geq 3$, $d \geq 2$, and let $\mathcal{C} \equiv 0$ be a simple and minimal $\Sigma\Pi\Sigma$ circuit of degree d, with k multiplication gates and n inputs. Let $r = \operatorname{rank}(\mathcal{C})$. Then we can construct a linear $(2, \frac{1}{12}, \frac{1}{4})$-LDC $E : \mathbb{F}^{n_1} \to \mathbb{F}^{n_2}$ with*

$$\frac{r}{2^{O(k^2)} \log(d)^{k-3}} \leq n_1 \quad \text{and} \quad n_2 \leq k \cdot d.$$

Thus, if k is a constant, then we can construct a linear $(2, \frac{1}{12}, \frac{1}{4})$-LDC that encodes messages of length $r/\text{polylog}(d)$ by codewords of length $O(d)$. As a corollary of Theorems 1.2 and 1.3 we get the next theorem.

THEOREM 1.4. *Let $k \geq 3$, $d \geq 2$, and let $\mathcal{C} \equiv 0$ be a simple and minimal $\Sigma\Pi\Sigma$ circuit of degree d with k multiplication gates and n inputs; then $r \leq 2^{O(k^2)} \log(d)^{k-2}$.*

Notice that the bound on r depends only on the degree and the number of multiplication gates and not on the number of variables! If the degree is polynomial in n (i.e., the circuit is of polynomial size), then the rank is bounded by $\text{polylog}(n)$, where a priori the rank could have been n.

PIT algorithms for depth 3 circuits. We design algorithms for PIT of depth 3 circuits with a constant number of multiplication gates. In particular we get a deterministic quasi-polynomial time algorithm and a randomized polynomial time algorithm that uses only polylog random bits. If the circuit is multilinear, i.e., every multiplication gate computes a multilinear polynomial, then we give a deterministic polynomial time algorithm for PIT. Our algorithms are non black box—all of them use the circuit computing the polynomial. The basic idea is to look for a minimal zero subcircuit and then, using Theorem 1.4, to write the linear functions in the circuit

as linear functions in $r \leq 2^{O(k^2)} \log(d)^{k-2}$ variables. Then we expand the monomials computed by the circuit and verify in a brute force manner that the resulting polynomial is zero. Thus the running time of our algorithm is the combined time that it takes to go over all subcircuits and the time that it takes to write all the monomials of a degree d polynomial in $\leq 2^{O(k^2)} \log(d)^{k-2}$ variables. We thus obtain the following result.

THEOREM 1.5. *Let $\mathcal{C}$ be a $\Sigma\Pi\Sigma$ circuit of degree d, with k multiplication gates and n inputs. Then we can check if $\mathcal{C} \equiv 0$:*

1. *Deterministically, in time* $\exp\left(2^{O(k^2)} \log^{k-1}(d)\right)$. *Thus, for a constant k the running time is* $\exp(\text{polylog}(d))$.
2. *Probabilistically, in time* $2^{O(k)} \text{poly}(d, \frac{1}{\epsilon})$, *using* $2^{O(k^2)} \log^{k-2}(d) \log(1/\epsilon)$ *random bits, with error probability ϵ. For constant k the running time is* $\text{poly}(d, \frac{1}{\epsilon})$, *and the number of random bits is* $\text{polylog}(d) \log(1/\epsilon)$.
3. *If $\mathcal{C}$ is also multilinear, then we can check if $\mathcal{C}$ is identically zero deterministically in time* $\exp(2^{O(k^2)}) \cdot \text{poly}(d)$. *For constant k the running time is* $\text{poly}(d)$.

Prior to our work the only algorithms that were designed for bounded depth circuits were the deterministic algorithm of [41] for pure multilinear depth 3 circuits and the black box algorithms of [22, 9, 29] for polynomials computed by depth 2 circuits (also known as sparse polynomials). None of the algorithms for sparse polynomials work in the case of depth 3 circuits, as such circuits can compute polynomials with exponentially many monomials. In fact, Klivans and Spielman [29] ask whether one could derandomize PIT for $\Sigma\Pi\Sigma$ circuits with only three multiplication gates ($k = 3$ in our notation). We give a deterministic algorithm that runs in quasi-polynomial time for this case, thus resolving the question of [29]. We note that a complete derandomization is to give a polynomial time algorithm for the problem, as was recently achieved by Kayal and Saxena [27]. We discuss their result in the next subsection.

1.5. Recent results. Kayal and Saxena [27] managed to give a polynomial time algorithm for PIT of depth 3 circuits with bounded top fan-in. Namely, they give an algorithm that runs in time polynomial in d^k, n, where k is the top fan-in, d is the degree of the circuit, and n is the number of variables. This result gives a complete derandomization of identity testing for depth 3 circuits with bounded top fan-in. In addition Kayal and Saxena give constructions of identically zero depth 3 circuits over $\mathbb{F} = GF(p)$ with $k = p$ for odd p, and $k = 3$ for $p = 2$, of degree d and rank $r = \log_p(d)$ (see Theorem 1.4).

We note, however, that for multilinear depth 3 circuits we give a polynomial time algorithm even when the top fan-in is $O(\sqrt{\log \log n})$ (Theorem 1.5, item 3), whereas [27] is polynomial time only when the top fan-in is constant.

1.6. Organization. In section 2 we analyze linear LDCs and derive Theorem 1.2. Section 3 is devoted to $\Sigma\Pi\Sigma$ circuits and their properties and serves as an introduction to the main part of the paper. In section 4 we give the proof of Theorem 1.3 and discuss the relation between $\Sigma\Pi\Sigma$ circuits and LDCs. Finally, in sections 5 and 6 we use our results to prove a structural theorem for zero $\Sigma\Pi\Sigma$ circuits and devise PIT algorithms based on this theorem.

2. Locally decodable codes. In this section we prove Theorem 1.2. We start by formally defining LDCs.

For a natural number n, let $[n] \triangleq \{1, \ldots, n\}$. Let $\mathbb{F}$ be a field. For a vector $x \in \mathbb{F}^n$ we write x_i for the ith coordinate of x. We denote by e_i the ith unit vector. For two

vectors $y, z \in \mathbb{F}^m$, denote by $\Delta(y,z)$ the number of coordinates in which y and z differ.

DEFINITION 2.1. *Let $\delta, \epsilon \in [0,1]$, and let q be an integer. We say that $E : \mathbb{F}^n \to \mathbb{F}^m$ is a* (q,δ,ϵ)-locally decodable code *if there exists a probabilistic oracle machine A such that*

- *in every invocation, A makes at most q queries (nonadaptively);*
- *for every $x \in \mathbb{F}^n$, for every $y \in \mathbb{F}^m$ with $\Delta(y, E(x)) < \delta m$, and for every $i \in [n]$, we have*

$$|\mathbb{F}| < \infty : \quad \boldsymbol{Pr}[A^y(i) = x_i] \geq \tfrac{1}{|\mathbb{F}|} + \epsilon,$$
$$|\mathbb{F}| = \infty : \quad \boldsymbol{Pr}[A^y(i) = x_i] \geq \epsilon,$$

where the probability is taken over the internal coin tosses of A.

We say that the code E is a linear *code if E is a linear transformation between $\mathbb{F}^n$ and $\mathbb{F}^m$.*

We are now ready to prove Theorem 1.2. We repeat its formulation here.

THEOREM 1.2 (restated). *Let $\delta, \epsilon \in [0,1]$, $\mathbb{F}$ be a field, and let $E : \mathbb{F}^n \to \mathbb{F}^m$ be a linear $(2,\delta,\epsilon)$-LDC. Then*

$$m \geq 2^{\frac{\epsilon\,\delta\,n}{4}-1}.$$

Our proof will build on the methods of [18], together with a novel reduction from LDCs over arbitrary fields to LDCs over $GF(2)$. We start by reviewing the results of [18]. The first step of their proof, given by the following lemma, is a reduction from the problem of proving lower bounds for LDCs to a graph-theoretic problem. The first such reduction was given in [26], where it was used to prove lower bounds on general LDCs. We note that in [18] the lemma was proved only over finite fields; however, it is easy to modify the proof to work for infinite fields as well.

LEMMA 2.2 (*implicit in* [18]). *Let $E : \mathbb{F}^n \to \mathbb{F}^m$ be a linear $(2,\delta,\epsilon)$-LDC, and let $a_1, \ldots, a_m \in \mathbb{F}^m$ be vectors such that*

$$E(x) = (\langle a_1, x\rangle, \ldots, \langle a_m, x\rangle)$$

($\langle\cdot,\cdot\rangle$ denotes the standard inner product). Then, for every $i \in [n]$, there exists a set $M_i \subset [m] \times [m]$ of at least $\frac{\epsilon\,\delta\,m}{4}$ disjoint pairs such that for every $(j_1, j_2) \in M_i$, $e_i \in Span\{a_{j_1}, a_{j_2}\}$.

From Lemma 2.2 we see that to prove lower bounds for two-query LDCs, it is sufficient to deal with the more combinatorial setting in which a given multiset of vectors contains many disjoint pairs spanning each unit vector.

The next step in the proof of [18] is a reduction from arbitrary finite fields to $GF(2)$. The next lemma summarizes the reduction given by [18].

LEMMA 2.3 (*implicit in* [18]). *Let $\mathbb{F}$ be a finite field, and let $a_1, \ldots, a_m \in \mathbb{F}^n$. For every $i \in [n]$ let $M_i \subset [m] \times [m]$ be a set of disjoint pairs of indices such that $e_i \in Span\{a_{j_1}, a_{j_2}\}$ for every $(j_1, j_2) \in M_i$. Then, there exist m' vectors $b_1, \ldots, b_{m'} \in \{0,1\}^n$ and n sets $M'_1, \ldots, M'_n \subset [m'] \times [m']$ of disjoint pairs such that*

1. *for every $(j_1, j_2) \in M'_i$, $b_{j_1} \oplus b_{j_2} = e_i$,*
2. *$m' = (|\mathbb{F}| - 1)m$, and*
3. *$\sum_{i=1}^n |M_i| \leq 2m + \frac{2}{|\mathbb{F}|-1} \sum_{i=1}^n |M'_i|$.*

The third and final step in the proof of [18] is a lemma which bounds the size of the matchings M_i, when the underlying field is $GF(2)$.

LEMMA 2.4 (*see* [18]). *Let $a_1, \ldots, a_m$ be elements of $\{0,1\}^n$. For every $i \in [n]$ let $M_i \subset [m] \times [m]$ be a set of disjoint pairs of indices such that $e_i = a_{j_1} \oplus a_{j_2}$ for every $(j_1, j_2) \in M_i$. Then*

$$\sum_{i=1}^{n} |M_i| \leq \frac{1}{2} m \log(m).$$

Notice that by Lemma 2.3 we have that $m = m'/|\mathbb{F}|$. Therefore, to get significant bounds from the combination of Lemmas 2.2, 2.3, and 2.4, we need $|\mathbb{F}|$ to be much smaller than 2^n. Thus, for very large fields (in particular, infinite fields) we do not get a significant result.

Our proof differs from that of [18] only in its second part—the reduction from $\mathbb{F}$ to $GF(2)$. Our reduction holds for any field, in particular for infinite $\mathbb{F}$, and does not involve the field size as a parameter.

LEMMA 2.5. *Let $\mathbb{F}$ be any field, and let $a_1, \ldots, a_m \in \mathbb{F}^n$. For every $i \in [n]$ let $M_i \subset [m] \times [m]$ be a set of disjoint pairs of indices such that $e_i \in Span\{a_{j_1}, a_{j_2}\}$ for every $(j_1, j_2) \in M_i$. Then, there exist m vectors $b_1, \ldots, b_m \in \{0,1\}^n$, and n sets $M_1'', \ldots, M_n'' \subset [m] \times [m]$ of disjoint pairs, such that*

1. *for every $(j_1, j_2) \in M_i''$, $b_{j_1} \oplus b_{j_2} = e_i$, and*
2. *$\sum_{i=1}^{n} |M_i| \leq 2 \sum_{i=1}^{n} |M_i''| + m$.*

Before giving the proof of the lemma we combine Lemmas 2.2, 2.5, and 2.4 to prove Theorem 1.2.

Proof of Theorem 1.2. Let $a_1, \ldots, a_m \in \mathbb{F}^n$ be vectors such that

$$E(x) = (\langle a_1, x \rangle, \ldots, \langle a_m, x \rangle).$$

From Lemma 2.2, we know that there exist n sets, $M_1, \ldots, M_n \subset [m] \times [m]$, of disjoint pairs of indices, such that for every $(j_1, j_2) \in M_i$ we have $e_i \in Span\{a_{j_1}, a_{j_2}\}$. We also know that

$$\forall\, i \in [n],\ |M_i| \geq \frac{\epsilon\, \delta\, m}{4}.$$

Now, let $b_1, \ldots, b_m \in \{0,1\}^n$ and $M_1'', \ldots, M_n'' \subset [m] \times [m]$ be as in Lemma 2.5. That is,

1. for every $(j_1, j_2) \in M_i''$, $b_{j_1} \oplus b_{j_2} = e_i$, and
2. $\sum_{i=1}^{n} |M_i| \leq 2 \sum_{i=1}^{n} |M_i''| + m$.

Using Lemma 2.4, we now have

$$\sum_{i=1}^{n} |M_i''| \leq \frac{1}{2} m \log(m).$$

This implies

$$n \cdot \frac{\epsilon\, \delta\, m}{4} \leq \sum_{i=1}^{n} |M_i| \leq 2 \sum_{i=1}^{n} |M_i''| + m \leq m \log(m) + m,$$

which, after division by m, gives the bound stated by the theorem. □

We now give the proof of Lemma 2.5.

Proof of Lemma 2.5. The proof will consist of two stages. First, we will remove a relatively small number of "bad" pairs from the given matchings $\{M_i\}$; then we

will transform the vectors $a_1, \ldots, a_m$ to vectors in $\{0,1\}^n$, while preserving a large portion of the pairs spanning the unit vectors.

Let (j_1, j_2) be a pair in M_i for some i such that either a_{j_1} or a_{j_2} are parallel to the unit vector e_i. Without loss of generality (w.l.o.g.) assume $a_{j_1} = c \cdot e_i$. We replace this pair with the pair (j_1, j_1). We do the same for all pairs containing a vector parallel to the unit vector spanned by this pair. This change does not affect the parameters of the lemma and is done only to simplify the analysis.

Next, we define a function $\theta : \mathbb{F}^n \backslash \{0\} \to [n]$ by

$$\theta(v) = \min\{i : v_i \neq 0\}.$$

For the rest of the proof we assume w.l.o.g. that in each pair (j_1, j_2) we have $\theta(a_{j_1}) \leq \theta(a_{j_2})$. (Note that we can assume w.l.o.g. that the vectors $a_1, \ldots, a_m$ are all different from zero.) We remove from each matching M_i all the pairs (j_1, j_2) in which $\theta(a_{j_1}) = i$. (This includes all pairs (j_1, j_1) described in the previous paragraph, and more.) Denote the resulting matching by M_i'. We claim that the total number of pairs removed in this stage is at most m.

Claim 2.6.

$$\sum_{i=1}^{n} |M_i| \leq \sum_{i=1}^{n} |M_i'| + m. \tag{2}$$

Proof. Let $p_1 = (j_1, j_2)$ and $p_2 = (k_1, k_2)$ be two removed pairs. If p_1 and p_2 were in the same matching M_i, then they would be disjoint, and so $j_1 \neq k_1$. If the pairs belonged to two different matchings, say, M_{i_1} and M_{i_2}, then $\theta(a_{j_1}) = i_1$ and $\theta(a_{k_1}) = i_2$, and again we get that $j_1 \neq k_1$. It follows that every removed pair has a distinct first element in the set $[m]$. Therefore, the total number of removed pairs cannot exceed m. ∎

In the following we assume w.l.o.g. that the first nonzero coordinate of each a_j is one. (We can assume that because we are allowed to use arbitrary linear combinations of the a_j when spanning the e_i.) The next claim asserts an important property of the matchings M_i'.

Claim 2.7. *For every $i \in [n]$ and $(j_1, j_2) \in M_i'$,*

$$e_i \in \mathit{Span}\{a_{j_1} - a_{j_2}\}.$$

Proof. Let $u = a_{j_1}$, $v = a_{j_2}$. We know that there exist two nonzero coefficients $\alpha, \beta \in \mathbb{F}$ such that $\alpha u + \beta v = e_i$. (Both coefficients are nonzero because we removed from M_i all pairs containing a vector parallel to e_i.) From this property it is clear that $\theta(u) \leq i$ (remember that $\theta(u) \leq \theta(v)$). As we removed all pairs in which $\theta(a_{j_1}) = i$ we conclude that $\theta(u) < i$. This in turn implies that $\theta(u) = \theta(v) < i$, because if $\theta(v) > \theta(u)$, then the vector $\alpha u + \beta v = e_i$ would have a nonzero coordinate in position $\theta(u) < i$. Now, since $v_{\theta(v)} = u_{\theta(u)} = 1$ we have that $\alpha + \beta = (\alpha u + \beta v)_{\theta(u)} = (e_i)_{\theta(u)} = 0$. Hence $e_i \in \mathit{Span}\{a_{j_1} - a_{j_2}\}$. ∎

Let us now proceed to the second stage of the proof of Lemma 2.5, in which we move from the field $\mathbb{F}$ to $GF(2)$. We will use a probabilistic argument to show the existence of a transformation that maps $\mathbb{F}$ to $GF(2)$, while preserving a large portion of the pairs that span a given unit vector.

For each $i \in [n]$, let a_{ji} denote the ith coordinate of the vector a_j. Let $V = \{a_{ji}\}_{j \in [m],\, i \in [n]}$ be the set of all field elements appearing in one of the vectors $a_1, \ldots, a_m$.

We pick a random function $f : V \to \{0,1\}$ and apply f to all the coordinates in all the vectors. Let

$$b_j = (f(a_{j1}), \ldots, f(a_{jn}))$$

be the vector in $\{0,1\}^n$ obtained from a_j after the transformation. We say that a pair $(j_1, j_2) \in M_i'$ "survived" the transformation if $e_i = b_{j_1} \oplus b_{j_2}$.

CLAIM 2.8. *The expected number of surviving pairs is* $\frac{1}{2}\sum_{i=1}^{n} |M_i'|$.

Proof. Consider a pair $(j_1, j_2) \in M_i'$. Since $e_i \in Span\{a_{j_1} - a_{j_2}\}$ we know that the vectors a_{j_1}, a_{j_2} are identical in all coordinates different from i. Hence, the vectors b_{j_1}, b_{j_2} will also be identical in those coordinates. From this we see that $e_i = b_{j_1} \oplus b_{j_2}$ iff b_{j_1} and b_{j_2} differ in their ith coordinate. This happens with probability of one-half. By linearity of expectation we can conclude that the expected number of surviving pairs is at least half the number of original pairs, which was $\sum_{i=1}^{n} |M_i'|$. ∎

From the above claim we can assert that there exists a function f for which the number of surviving pairs is at least $\frac{1}{2}\sum_{i=1}^{n} |M_i'|$. Thus, we have shown that there exist a set of vectors $b_1, \ldots, b_m \in \{0,1\}^n$ and matchings $M_i'' \subset [m] \times [m]$ such that for every $(j_1, j_2) \in M_i''$, we have $e_i = b_{j_1} \oplus b_{j_2}$. Furthermore, we can assume that

$$\sum_{i=1}^{n} |M_i'| \leq 2 \sum_{i=1}^{n} |M_i''|, \tag{3}$$

which completes the proof of the lemma, since now

$$\sum_{i=1}^{n} |M_i| \leq \sum_{i=1}^{n} |M_i'| + m \leq 2 \sum_{i=1}^{n} |M_i''| + m. \qquad \square$$

The next corollary combines the results of Lemmas 2.5 and 2.4 in a compact form. This corollary will be used in the proof given in section 4.

COROLLARY 2.9. *Let* $\mathbb{F}$ *be any field, and let* $a_1, \ldots, a_m \in \mathbb{F}^n$. *For every* $i \in [n]$ *let* $M_i \subset [m] \times [m]$ *be a set of disjoint pairs of indices* (j_1, j_2) *such that* $e_i \in Span\{a_{j_1}, a_{j_2}\}$. *Then*

$$\sum_{i=1}^{n} |M_i| \leq m \log(m) + m.$$

3. ΣΠΣ circuits. In this section we give some definitions related to ΣΠΣ circuits and describe some elementary operations that can be preformed on them. These definitions and operations will be used in the following sections.

3.1. Definitions. In the following we treat vectors in $\mathbb{F}^n$ also as linear forms in $\mathbb{F}[x_1, \ldots, x_n]$.

DEFINITION 3.1. *Let* $u \in \mathbb{F}^n$, $u = (u_1, \ldots, u_n)$. *Then*

$$u(x) = u_1 x_1 + u_2 x_2 + \cdots + u_n x_n.$$

DEFINITION 3.2. *Let* $v, u \in \mathbb{F}^n \backslash \{0\}$. *We write* $u \sim v$ *if there exists* $c \in \mathbb{F}$ *such that* $u = c \cdot v$.

We proceed to the main definition of this section.

DEFINITION 3.3. *Let $\mathbb{F}$ be a field. A $\Sigma\Pi\Sigma$ circuit, $\mathcal{C}$, over $\mathbb{F}$, with n inputs and k multiplication gates (i.e., top fan-in is k), is the formal expression*

$$\mathcal{C}(x) = \sum_{i=1}^{k} c_i \prod_{j=1}^{d_i} L_{ij}(x),$$

where for each $i \in [k]$, $j \in [d_i]$, L_{ij} is a nonconstant linear function,

$$L_{ij}(x) = L_{ij}^0 + L_{ij}^1 \cdot x_1 + \cdots + L_{ij}^n \cdot x_n,$$

and $c_i, L_{ij}^t \in \mathbb{F}$ for all i, j, t.

For every $i \in [k]$ define N_i to be the ith multiplication gate of $\mathcal{C}$:

$$N_i(x) \triangleq \prod_{j=1}^{d_i} L_{ij}(x).$$

For each $i \in [k]$, d_i is the degree of N_i. The number k denotes the number of different multiplication gates and is referred to as the top fan-in of the circuit. The total degree of $\mathcal{C}$ is $\max\{d_i\}$, and the size of $\mathcal{C}$ is $\sum_{i=1}^{k} d_i$. We denote with $\operatorname{rank}(\mathcal{C})$ the rank of $\mathcal{C}$:

$$\operatorname{rank}(\mathcal{C}) \triangleq \dim(Span\{L_{ij} : i \in [k],\, j \in [d_i]\}).$$

Remark. When dealing with $\Sigma\Pi\Sigma$ circuits, we will always assume that all the linear functions appearing in the circuit are different from zero.

We are interested in $\Sigma\Pi\Sigma$ circuits that compute the zero polynomial in $\mathbb{F}[x_1, \ldots, x_n]$. If $\mathcal{C}$ is such a circuit, we write $\mathcal{C} \equiv 0$. When dealing with circuits of this kind, it is sufficient to consider circuits of limited structure. This notion is made precise by the following definition and the lemma that follows.

DEFINITION 3.4. *Let $k, d > 0$ be integers. A $\Sigma\Pi\Sigma$ circuit $\mathcal{C}$ is called a $\Sigma\Pi\Sigma(k, d)$ circuit if the following three conditions hold:*

- *the top fan-in of $\mathcal{C}$ is k;*
- *$d_1 = d_2 = \cdots = d_k = d$; and*
- *for every $i \in [k]$ and $j \in [d]$, L_{ij} is a homogeneous linear form, that is, $L_{ij}(x) = L_{ij}^1 \cdot x_1 + \cdots + L_{ij}^n \cdot x_n$. (The free coefficient in each linear function is zero.)*

When dealing with $\Sigma\Pi\Sigma(k, d)$ circuits we will treat the linear functions L_{ij} also as vectors in $\mathbb{F}^n$, that is, $L_{ij} = (L_{ij}^1, \ldots, L_{ij}^n)$.

LEMMA 3.5. *There exists a polynomial time algorithm such that, given as input a $\Sigma\Pi\Sigma$ circuit $\mathcal{C}$, with top fan-in k and total degree $d > 0$, it outputs a $\Sigma\Pi\Sigma(k, d)$ circuit $\mathcal{C}'$ such that $\mathcal{C} \equiv 0$ iff $\mathcal{C}' \equiv 0$. The circuit $\mathcal{C}'$ is called the corresponding $\Sigma\Pi\Sigma(k, d)$ circuit of $\mathcal{C}$.*

Proof. We introduce a new variable y and define $\mathcal{C}'$ to be a circuit with input variables $x_1, \ldots, x_n, y$. Let

$$L_{ij}(x) = L_{ij}^0 + \sum_{t=1}^{n} L_{ij}^t \cdot x_t$$

be a linear function appearing in $\mathcal{C}$. Define

$$L'_{ij}(x, y) = L_{ij}^0 \cdot y + \sum_{t=1}^{n} L_{ij}^t \cdot x_t,$$

and define $\mathcal{C}'$ to be

$$\mathcal{C}'(x,y) = \sum_{i=1}^{k} c_i\, y^{d-d_i} \prod_{j=1}^{d_i} L'_{ij}(x,y).$$

Clearly, $\mathcal{C}'$ is a $\Sigma\Pi\Sigma(k,d)$ circuit and can be computed from $\mathcal{C}$ in time polynomial in the size of $\mathcal{C}$. Note that if we write

$$\mathcal{C}(x) = \sum_{i=0}^{d} P_i(x),$$

where $P_i(x)$ denotes the homogeneous part of degree i of $\mathcal{C}(x)$, then

$$\mathcal{C}'(x,y) = \sum_{i=0}^{d} P_i(x) y^{d-i}.$$

Therefore $\mathcal{C} \equiv 0$ iff $\mathcal{C}' \equiv 0$. □

Lemma 3.5 shows that to achieve our final goal, which is to derive PIT algorithms for $\Sigma\Pi\Sigma$ circuits, it is sufficient to consider $\Sigma\Pi\Sigma(k,d)$ circuits. For the rest of the paper we will deal only with $\Sigma\Pi\Sigma(k,d)$ circuits, and we shall sometimes refer to them simply as $\Sigma\Pi\Sigma$ circuits, omitting the suffix (k,d) where it is not needed.

3.2. Identically zero $\Sigma\Pi\Sigma$ circuits.

Simple circuits. It might be the case that there exists a linear function, L, that appears (up to a constant) in all multiplication gates of $\mathcal{C}$. In this case, we can divide each multiplication gate by L and get a simpler circuit $\mathcal{C}'$, whose degree is smaller than that of $\mathcal{C}$ by one. Clearly $\mathcal{C} \equiv 0$ iff $\mathcal{C}' \equiv 0$. The next two definitions deal with this case in a more general way.

DEFINITION 3.6. *Let $\mathcal{C}$ be a $\Sigma\Pi\Sigma$ circuit, and let $N_1, \ldots, N_k$ be its multiplication gates. Define*[4]

$$\gcd(\mathcal{C}) \triangleq \text{g.c.d.}(N_1(x), \ldots, N_k(x)).$$

Since each multiplication gate is a product of linear forms, $N_i(x) = \prod_{j=1}^{d_i} L_{ij}(x)$, we get that $\gcd(\mathcal{C})$ is the product of all the linear forms that appear in all the multiplication gates (up to multiplication by constants). Note also that $\gcd(\mathcal{C})$ can be easily computed from $\mathcal{C}$.

It is clear that $\mathcal{C} \equiv 0$ iff $\frac{\mathcal{C}}{\gcd(\mathcal{C})} \equiv 0$. This fact motivates the following definition.

DEFINITION 3.7. *A $\Sigma\Pi\Sigma$ circuit $\mathcal{C}$ is called* simple *if $\gcd(\mathcal{C}) = 1$. Let us also define $\mathrm{sim}(\mathcal{C})$ to be the simple circuit obtained from $\mathcal{C}$ by dividing each multiplication gate by $\gcd(\mathcal{C})$. It is clear that $sim(\mathcal{C})$ is always simple and that*

$$\mathcal{C}(x) = sim(\mathcal{C})(x) \cdot \gcd(\mathcal{C})(x).$$

Example 3.8. Let

$$\begin{aligned}\mathcal{C}(x) = {} & (\mathbf{x_1} + \mathbf{2x_2} + \mathbf{x_3} + \mathbf{1})(2x_1 + 4x_2 + 5x_3 + 2)(2x_1 + 4x_2 + 2x_3) \\ & + (\mathbf{x_1} + \mathbf{2x_2} + \mathbf{x_3} + \mathbf{1})(6x_1 + 4x_2 + 5x_3)(1x_1 + 1x_2 + 2x_3 + 4) \\ & + (\mathbf{2x_1} + \mathbf{4x_2} + \mathbf{2x_3} + \mathbf{2})(4x_2 + 1x_3)(7x_1 + 4x_2 + 2x_3).\end{aligned}$$

[4] g.c.d. stands for greatest common divisor.

Then

$$\gcd(\mathcal{C}) = x_1 + 2x_2 + x_3 + 1,$$

and

$$\begin{aligned} sim(\mathcal{C})(x) = &(2x_1 + 4x_2 + 5x_3 + 2)(2x_1 + 4x_2 + 2x_3) \\ &+ (6x_1 + 4x_2 + 5x_3)(1x_1 + 1x_2 + 2x_3 + 4) \\ &+ 2 \cdot (4x_2 + 1x_3)(7x_1 + 4x_2 + 2x_3). \end{aligned}$$

Minimal circuits. Suppose we have two $\Sigma\Pi\Sigma$ circuits $\mathcal{C}_1$ and $\mathcal{C}_2$, both of them equal to zero. Let k_1, k_2 denote the top fan-in of $\mathcal{C}_1$ and of $\mathcal{C}_2$, respectively. We can add $\mathcal{C}_1$ to $\mathcal{C}_2$ to create a new circuit $\mathcal{C} = \mathcal{C}_1 + \mathcal{C}_2$, with top fan-in $k_1 + k_2$, that will also be equal to zero. This new circuit $\mathcal{C}$, however, can be broken down into two smaller subcircuits that are zero. In the following we will be interested in circuits that *cannot* be broken down into smaller subcircuits that are equal to zero. The next two definitions deal with circuits of this type.

DEFINITION 3.9. *Let $\mathcal{C}$ be a $\Sigma\Pi\Sigma$ circuit, and let $\emptyset \neq T \subseteq [k]$. Then $\mathcal{C}_T$ is defined to be the subcircuit of $\mathcal{C}$ composed of the multiplication gates whose indices appear in T:*

$$\mathcal{C}_T(x) \triangleq \sum_{i \in T} c_i \prod_{j=1}^{d_i} L_{ij}(x) = \sum_{i \in T} c_i N_i(x).$$

DEFINITION 3.10. *Let $\mathcal{C} \equiv 0$ be a $\Sigma\Pi\Sigma$ circuit. We say that $\mathcal{C}$ is* minimal *if for every nonempty subset $T \subset [k]$, apart from $[k]$ itself, we have $\mathcal{C}_T \not\equiv 0$.*

The following easy claim shows that most properties of a $\Sigma\Pi\Sigma$ circuit $\mathcal{C}$ remain when we move to the corresponding $\Sigma\Pi\Sigma(k, d)$ circuit. The proof is immediate from the proof of Lemma 3.5.

CLAIM 3.11. *Let $\mathcal{C}$ be a $\Sigma\Pi\Sigma$ circuit, and let $\mathcal{C}'$ be the corresponding $\Sigma\Pi\Sigma(k, d)$ circuit (as defined in Lemma 3.5). Then we have the following:*

- $\operatorname{rank}(\mathcal{C}) \leq \operatorname{rank}(\mathcal{C}') \leq \operatorname{rank}(\mathcal{C}) + 1$.
- *$\mathcal{C}$ is simple iff $\mathcal{C}'$ is simple.*
- *$\mathcal{C}$ is minimal iff $\mathcal{C}'$ is minimal.*

Taking a linear transformation. We start with a simple operation of setting one of the variables to zero. This operation can be looked at as projecting all the linear functions in the circuit on a subspace of codimension 1.

DEFINITION 3.12. *Let $\mathcal{C}$ be a $\Sigma\Pi\Sigma$ circuit, and let $t \in [n]$. Define $\mathcal{C}|_{x_t=0}$ to be the circuit obtained from $\mathcal{C}$ by setting the variable x_t to zero. (This is the same as changing the tth coordinate in each linear form L_{ij} to zero.) The polynomial computed by $\mathcal{C}|_{x_t=0}$ is therefore*

$$(\mathcal{C}|_{x_t=0})(x) = \mathcal{C}(x_1, \ldots, x_{t-1}, 0, x_{t+1}, \ldots, x_n).$$

We can generalize the operation just defined by applying a general linear transformation on the linear functions of the circuit.

DEFINITION 3.13. *Let*

$$\mathcal{C}(x) = \sum_{i=1}^{k} c_i \prod_{j=1}^{d} L_{ij}(x)$$

be a $\Sigma\Pi\Sigma(k,d)$ circuit on n variables, and let $\pi : \mathbb{F}^n \to \mathbb{F}^n$ be a linear transformation. Define $\pi(\mathcal{C})$ to be the circuit obtained from $\mathcal{C}$ by applying π on all linear forms appearing in the circuit.[5] *That is,*

$$\pi(\mathcal{C})(x) = \sum_{i=1}^{k} c_i \prod_{j=1}^{d} \pi(L_{ij})(x).$$

The following claim is easy to verify.

CLAIM 3.14. *Let $\mathcal{C}$ be a $\Sigma\Pi\Sigma(k,d)$ circuit, and let $\pi : \mathbb{F}^n \to \mathbb{F}^n$ be an invertible linear transformation. Then*

- *$\mathcal{C} \equiv 0$ iff $\pi(\mathcal{C}) \equiv 0$,*
- *$\mathcal{C}$ is simple iff $\pi(\mathcal{C})$ is simple,*
- *$\mathcal{C}$ is minimal iff $\pi(\mathcal{C})$ is minimal, and*
- *$\operatorname{rank}(\mathcal{C}) = \operatorname{rank}(\pi(\mathcal{C}))$.*

4. ΣΠΣ circuits and LDCs. In this section we prove Theorem 1.3, which is the main result of the paper. This theorem shows the relation between $\Sigma\Pi\Sigma$ circuits and linear LDCs. It is more convenient to us to prove the theorem for $\Sigma\Pi\Sigma(k,d)$ circuits instead of general $\Sigma\Pi\Sigma$ circuits. From Claim 3.11, we know that moving from $\mathcal{C}$ to its corresponding $\Sigma\Pi\Sigma(k,d)$ circuit does not affect any of the relevant properties of $\mathcal{C}$, so the following theorem is equivalent to Theorem 1.3.

THEOREM 4.1. *Let $k \geq 3$, $d \geq 2$, and let $\mathcal{C} \equiv 0$ be a simple and minimal $\Sigma\Pi\Sigma(k,d)$ circuit, on n inputs, over a field $\mathbb{F}$. Then, there exists a linear $(2, \frac{1}{12}, \frac{1}{4})$-LDC $E : \mathbb{F}^{n_1} \to \mathbb{F}^{n_2}$, with*

$$\frac{\operatorname{rank}(\mathcal{C})}{P(k)\log(d)^{k-3}} \leq n_1 \quad \text{and} \quad n_2 \leq k \cdot d, \quad \text{where} \quad P(k) = 2^{O(k^2)}.$$

We prove Theorem 4.1 by induction on k. We devote section 4.1 to the base case of $k = 3$ and give the proof of the inductive step in section 4.2.

Before moving on to the proof of Theorem 4.1 we should explain why we are dealing only with circuits whose top fan-in is at least 3. The reason for this is that the structure of a zero $\Sigma\Pi\Sigma(k,d)$ circuit with $k = 1, 2$ is trivial. If $\mathcal{C}$ has only one multiplication gate ($k = 1$), then it is zero iff one of the linear functions appearing in it is the zero function. The case of $k = 2$ is equally trivial, as seen by the next claim.

CLAIM 4.2. *Let $\mathcal{C} = c_1 N_1(x) + c_2 N_2(x)$ be a $\Sigma\Pi\Sigma(2,d)$ circuit. Suppose $\mathcal{C} \equiv 0$. Then, the linear functions, appearing in the two multiplication gates N_1 and N_2, are the same, up to an ordering and multiplication by constants.*

Proof. Since $\mathcal{C} \equiv 0$, we have that $c_1 N_1(x) \equiv -c_2 N_2(x)$. Each multiplication gate N_i is a product of linear functions. Since every polynomial can be written, in a unique way, as a product of irreducible polynomials, and since every linear function is irreducible, we have that the linear functions in the two gates must be the same (up to an ordering and multiplication by constants). ∎

4.1. Proof of Theorem 4.1 for $k = 3$. Let $r = \operatorname{rank}(\mathcal{C})$. Then there exist r linearly independent functions $L_1, \ldots, L_r$ in $\mathcal{C}$. Using Claim 3.14, we can assume w.l.o.g. that for every $t \in [r]$, $L_t(x) = x_t$ (or in other words, $L_t = e_t$). Consider the circuit $\mathcal{C}|_{x_t=0}$ for some $t \in [r]$. Clearly $\mathcal{C}|_{x_t=0} \equiv 0$. From the fact that the function $L_t = e_t$ appears in one of the multiplication gates, we know that this gate

[5] Remember that we identify linear forms with vectors in $\mathbb{F}^n$.

will become zero in $\mathcal{C}|_{x_t=0}$. The following claim assures us that neither of the other two multiplication gates will become zero in $\mathcal{C}|_{x_t=0}$.

CLAIM 4.3. *Let L and L' be two linear functions appearing in two different multiplication gates of $\mathcal{C}$. Then $L \not\sim L'$.*

Proof. Assume for a contradiction that L divides both N_1 and N_2. As $c_3N_3(x) = -c_1N_1(x) - c_2N_2(x)$ we get that $N_3(x)$ is also divisible by L. But $\mathcal{C}$ is simple, so this is a contradiction. □

How can a circuit with two nonzero multiplication gates be zero? From Claim 4.2, this is possible only if the two gates contain the same linear functions, up to an ordering and multiplication by constants.

We thus get that every variable x_t, $t \in [r]$, induces a matching on the linear functions of the circuit. This matching contains d pairs of linear functions such that for every pair (L, L') in the matching, we have that L and L' belong to two different multiplication gates and that $L|_{x_t=0} \sim L'|_{x_t=0}$. Denote with M_t the matching induced by x_t. The next claim gives us more information about the pairs appearing in those matchings.

CLAIM 4.4. *Let $t \in [n]$, and let $L, L' \in \mathbb{F}^n$ such that $L \not\sim L'$, and $L|_{x_t=0} \sim L'_{x_t=0}$. Then*

$$e_t \in Span\{L, L'\}.$$

Proof. Let $L = (a_1, \ldots, a_n)$, $L' = (b_1, \ldots, b_n)$. Since $L|_{x_t=0} \sim L'|_{x_t=0}$, we know that there exists a constant $c \in \mathbb{F}$ such that for all $j \neq t$ we have $a_j = c \cdot b_j$. The fact that $L \not\sim L'$ implies that $a_t \neq c \cdot b_t$. It follows that $e_t \sim L - c \cdot L'$. In particular we get that $e_t \in Span\{L, L'\}$. □

From Claim 4.4 we see that every pair $(L, L') \in M_t$ spans the vector e_t. We also have that all the matchings $\{M_t\}_{t \in r}$ are contained in a set of $3d$ linear functions and that each matching contains d pairs. We can now construct a linear LDC in the following way. For each $i \in [3]$, $j \in [d]$, let $l_{ij} \in \mathbb{F}^r$ be the projection of L_{ij} on the first r coordinates. Define $E : \mathbb{F}^r \to \mathbb{F}^{3d}$ by

$$E_{ij}(x) = l_{ij}(x).$$

To show that E is a $(2, \frac{1}{12}, \frac{1}{4})$-LDC, we need to show a decoding algorithm for it. For each $t \in [r]$ we know that there are d disjoint pairs of code positions that span e_t. (Note that taking the projection on the first r coordinates does not affect this property.) To decode x_t we simply pick a random pair, uniformly, among these d pairs, and compute the linear combination giving e_t. Suppose we picked $l_{ij}(x)$ and $l_{i'j'}(x)$. We know that there exist constants $a, b \in \mathbb{F}$ such that

$$a \cdot l_{ij} + b \cdot l_{i'j'} = e_t.$$

Therefore

$$a \cdot E_{i,j}(x) + b \cdot E_{i',j'}(x) = a \cdot l_{ij}(x) + b \cdot l_{i'j'}(x) = e_t(x) = x_t.$$

If our codeword has at most $\frac{1}{12}(3d) = \frac{d}{4}$ corrupted positions, then at least $\frac{3}{4}$ of the d pairs are uncorrupted, and our algorithm will succeed with probability greater than $\frac{3}{4}$.

In the notation of the theorem, we have $n_1 = r$ and $n_2 = 3d = kd$. Let $P(3) = 1$; then

$$n_1 = r \geq \frac{r}{P(k)\log(d)^{k-3}},$$

and the theorem follows for $k = 3$.

4.2. Proof of Theorem 4.1 for $k \geq 4$. The proof is by induction on k. The idea behind the proof is the following. Assume that x_1 appears as a linear function in the circuit. A natural thing to do is to consider $\mathcal{C}|_{x_1=0}$. This circuit contains fewer multiplication gates, and so we would like to find an LDC in it by induction. A possible problem is that the rank of every minimal subcircuit is low. We can overcome this problem by showing that there are many variables $x_1, \ldots, x_m$ ($m \geq r/2^k$) such that there exists $I \subset [k]$ for which $\mathcal{C}_I \not\equiv 0$, but for every $t \in [m]$, $(\mathcal{C}_I)|_{x_t=0}$ is identically zero and minimal. In particular we show that this implies that the rank of $\mathcal{C}_I$ is at least m. We would like to construct a code from $\mathcal{C}_I$, so we consider, say, $(\mathcal{C}_I)|_{x_1=0}$. This circuit is identically zero and minimal, but it is not necessarily simple. Therefore we take $sim((\mathcal{C}_I)|_{x_t=0})$. However, it might be the case that the rank of this circuit is very small, i.e., that we lost a lot of rank when we removed the g.c.d. We overcome this difficulty by proving that there are relatively few ($\approx \log d$) variables, say, $x_1, \ldots, x_{\log d}$, such that the span of the linear functions in $sim((\mathcal{C}_I)|_{x_t=0})_{t=1,\ldots,\log d}$ contains almost all the functions of $\mathcal{C}_I$. In particular, for some t, the rank of $sim((\mathcal{C}_I)|_{x_t=0})$ is relatively high, so we can apply the induction hypothesis on this circuit. Proving the existence of such t is the main technical difficulty of the proof (Claim 4.8). We now give the formal proof.

Let $k \geq 4$, and assume the correctness of Theorem 4.1 for all $3 \leq k' < k$. Let

$$\mathcal{C}(x) = \sum_{i=1}^{k} c_i \prod_{j=1}^{d} L_{ij}(x)$$

be a $\Sigma\Pi\Sigma(k,d)$ circuit satisfying the conditions of the theorem. As in the proof for $k = 3$, let $r = \text{rank}(\mathcal{C})$, and w.l.o.g. assume that the circuit contains the first r unit vectors $e_1, \ldots, e_r$. We can also assume that

$$r \geq P(k) \log(d)^{k-3}, \tag{4}$$

for otherwise the theorem is trivially true, since we can always construct a two-query LDC whose message size is 1, satisfying the requirements of the theorem.

CLAIM 4.5. *For every $t \in [r]$ there exists a set $I_t \subset [k]$ such that*

1. $2 \leq |I_t| \leq k-1$ *and*
2. $(\mathcal{C}|_{x_t=0})_{I_t}$ *is identically zero and minimal.*

Proof. Let $t \in [r]$. Clearly $\mathcal{C}|_{x_t=0} \equiv 0$. Denote with k' the number of multiplication gates in $\mathcal{C}$ that become zero when $x_t = 0$. (These are exactly those multiplication gates that contain a linear function parallel to e_t.) Since we assumed that $\mathcal{C}$ contains e_t, we know that $k' \geq 1$. It is also easy to verify that $k' \leq k-2$. (If $k' = k$, then $\mathcal{C}$ is not simple, and if $k' = k-1$, then $\mathcal{C}$ is not divisible by x_t—as in Claim 4.3.) Therefore, the circuit $\mathcal{C}|_{x_t=0}$ is identically zero and contains at least two (and at most $k-1$) nonzero multiplication gates. Hence, we can decompose $\mathcal{C}|_{x_t=0}$ into minimal subcircuits, each of top fan-in at least two and at most $k-1$. Take I_t to be the index set of any one of these minimal subcircuits. ∎

From Claim 4.5 we can conclude that there are $m \geq \frac{r}{2^k}$ variables (w.l.o.g. $x_1, \ldots, x_m$) that have the same set I_t. Let $I = I_1 = \cdots = I_m$, and define

$$\hat{\mathcal{C}} = sim(\mathcal{C}_I).$$

The next claim summarizes several facts we know about the circuit $\hat{\mathcal{C}}$.

CLAIM 4.6.
1. $\hat{\mathcal{C}}$ *is a* $\Sigma\Pi\Sigma(\hat{k},\hat{d})$ *circuit with* $2 \le \hat{k} \le k-1$, $0 < \hat{d} \le d$.
2. $\hat{\mathcal{C}}$ *is simple.*
3. $\hat{\mathcal{C}} \not\equiv 0$.
4. *For all* $t \in [m]$, $\hat{\mathcal{C}}|_{x_t=0} \equiv 0$ *and is minimal.*
5. *For all* $t \in [m]$, e_t *does not appear in* $\hat{\mathcal{C}}$.

Proof. Parts 1 and 2 follow from the definition of $\hat{\mathcal{C}}$ (the fact that $0 < \hat{d}$ follows from 3 and 4). Part 3 is true because we assumed that $\mathcal{C}$ is minimal. Part 4 follows from the fact that $\hat{\mathcal{C}} = sim(\mathcal{C}_I)$ and that $(\mathcal{C}_I)|_{x_t=0} \equiv 0$ is minimal for all $t \in [m]$. Finally, 5 is a direct consequence of 4. □

Let $\hat{r} \triangleq \text{rank}(\hat{\mathcal{C}})$. The next claim shows that the rank of our chosen subcircuit $\hat{\mathcal{C}}$ is not considerably smaller than the rank of $\mathcal{C}$.

CLAIM 4.7. $\hat{r} \ge m$ $(\ge \frac{r}{2^k})$.

Proof. To prove the claim, we will show that the linear functions of $\hat{\mathcal{C}}$ span the unit vectors $e_1, \ldots, e_m$. Suppose, on the contrary, that there exists an index $t \in [m]$ for which e_t is not spanned by the linear functions of $\hat{\mathcal{C}}$. Assume w.l.o.g. that $t = 1$. There exists an invertible linear transformation $\pi : \mathbb{F}^n \to \mathbb{F}^n$ that satisfies the following two constraints:

- $\pi(e_1) = e_1$.
- The variable x_1 does not appear in the circuit $\pi(\hat{\mathcal{C}})$. (Equivalently, all the linear functions in $\pi(\hat{\mathcal{C}})$ are orthogonal to e_1.)

From Claim 4.6 we know that $\hat{\mathcal{C}} \not\equiv 0$ and that $\hat{\mathcal{C}}|_{x_1=0} \equiv 0$. Hence $\hat{\mathcal{C}}(x)$ can be written as

$$\hat{\mathcal{C}}(x) \equiv x_1 \cdot g(x),$$

where $g(x)$ is a nonzero polynomial. We can look at the transformation π as a linear change of variables and denote with $\pi(g)$ the polynomial obtained from $g(x)$ after this change. Thus,

$$\pi(\hat{\mathcal{C}})(x) \equiv \pi(x_1) \cdot \pi(g)(x) \equiv x_1 \cdot \pi(g)(x). \tag{5}$$

Now, since $g(x) \not\equiv 0$, and since π is invertible, Claim 3.14 implies[6] that $\pi(g)(x) \not\equiv 0$. From this and from (5) we see that $\pi(\hat{\mathcal{C}})(x)$ is a nonzero polynomial divisible by x_1. This is a contradiction, since we assumed that x_1 does not appear in $\pi(\hat{\mathcal{C}})$. □

We would like to use the inductive hypothesis on a well-chosen circuit among $\hat{\mathcal{C}}|_{x_1=0}, \ldots, \hat{\mathcal{C}}|_{x_m=0}$. However, there are two obstacles in the way. The first is that the top fan-in of $\hat{\mathcal{C}}$ might be equal to 2 (the theorem holds only for $k \ge 3$). This case is rather simple, since we can use the analysis given in section 4.1 to construct an LDC satisfying the conditions of the theorem. (A detailed analysis of this special case is deferred to the end of this section.) From now on we assume that $\hat{k} \ge 3$. The second obstacle is that these circuits are not necessarily simple. We overcome this obstacle by using the inductive hypothesis on $sim(\hat{\mathcal{C}}|_{x_t=0})$ instead. The next claim, whose proof is deferred to section 4.3, tells us which of these circuits we should pick.

For each $t \in [m]$, let $r_t \triangleq \text{rank}(sim(\hat{\mathcal{C}}|_{x_t=0}))$.

CLAIM 4.8. *There exists* $t \in [m]$ *such that*

$$r_t \ge \frac{\hat{r}}{2^{k+1}\log(d)}.$$

[6] It is easy to see that this part of Claim 3.14 holds also for general polynomials and not just $\Sigma\Pi\Sigma$ circuits.

Claim 4.8 assures us that one of the r_t is large (we assume w.l.o.g. that $t = 1$). We get that

$$r_1 \geq \frac{\hat{r}}{2^{k+1}\log(d)}. \tag{6}$$

Our next step is to apply the induction hypothesis to the circuit $sim(\hat{\mathcal{C}}|_{x_1=0})$. However, to use Theorem 4.1, we require that the degree of the given circuit be at least two. The next claim shows that the degree of $sim(\hat{\mathcal{C}}|_{x_1=0})$ is indeed at least two.

CLAIM 4.9. *Let d_1 denote the degree of $sim(\hat{\mathcal{C}}|_{x_1=0})$. Then $d_1 \geq 2$.*

Proof. If $d_1 < 2$, then $r_1 < k$. (The number of linear functions is at most $\hat{k} < k$.) By (6) we get that

$$\hat{r} \leq k2^{k+1}\log(d).$$

Now, using the fact that $\hat{r} \geq m \geq \frac{r}{2^k}$ (Claim 4.7), we conclude that

$$r \leq 2^k\hat{r} \leq k2^{2k+1}\log(d),$$

contradicting (4), for an appropriate choice of $P(k) = 2^{O(k^2)}$. ∎

Therefore $sim(\hat{\mathcal{C}}|_{x_1=0})$ satisfies all the conditions of Theorem 4.1. The induction hypothesis, applied on $sim(\hat{\mathcal{C}}|_{x_1=0})$, asserts that there exists a $(2, \frac{1}{12}, \frac{1}{4})$-LDC, $E : \mathbb{F}^{n_1} \to \mathbb{F}^{n_2}$, with

$$n_1 \geq \frac{r_1}{P(\hat{k})\log(d_1)^{\hat{k}-3}} \quad \text{and} \quad n_2 \leq \hat{k} \cdot d_1 \ (\leq k \cdot d).$$

Using (6) and the facts that $\hat{k} \leq k - 1$ and $\hat{r} \geq m \geq \frac{r}{2^k}$, we derive the following inequalities:

$$\begin{aligned} n_1 &\geq \frac{r_1}{P(\hat{k})\log(d_1)^{\hat{k}-3}} \\ &\geq \frac{r_1}{P(k-1)\log(d)^{k-4}} \\ &\geq \frac{\hat{r}}{2^{k+1}P(k-1)\log(d)^{k-3}} \\ &\geq \frac{r}{2^{2k+1}P(k-1)\log(d)^{k-3}} \\ &\geq \frac{r}{P(k)\log(d)^{k-3}} \end{aligned}$$

(for an appropriate choice of $P(k) = 2^{O(k^2)}$). This completes the proof of the inductive step and of Theorem 4.1.

4.2.1. A special case: $\hat{k} = 2$. In this subsection we analyze a special case of the proof of Theorem 4.1. This case is when $\hat{k}$ (the top fan-in of the circuit $\hat{C}$, whose properties are detailed in Claim 4.6) is equal to 2. The analysis of this case differs from the analysis of the general ($\hat{k} \geq 3$) case because we cannot apply the inductive hypothesis on $\hat{C}$ (or more precisely, on the circuits $C|_{x_t=0}$). We now show how to complete the proof of the theorem (that is, to construct an LDC satisfying the requirements of the thorem) in this case.

Denote by $\hat{N}_1$ and $\hat{N}_2$ the two multiplication gates of $\hat{\mathcal{C}}$. We can write

$$\hat{\mathcal{C}}(x) \equiv c_1\hat{N}_1(x) + c_2\hat{N}_2(x).$$

Now, since $\hat{\mathcal{C}}$ is simple and nonzero, we have

$$\gcd(\hat{\mathcal{C}}) \equiv \text{g.c.d.}(\hat{N}_1(x), \hat{N}_2(x)) \equiv 1.$$

Next, let $t \in [m]$, and consider what happens to $\hat{\mathcal{C}}$ after we set x_t to zero. We know that $\hat{\mathcal{C}}|_{x_t=0} \equiv 0$, and so

$$c_1\hat{N}_1|_{x_t=0} \equiv -c_2\hat{N}_2|_{x_t=0}.$$

Now, since $\hat{N}_1|_{x_t=0}$ and $\hat{N}_2|_{x_t=0}$ are both nonzero (by Claim 4.6, $e_1, \ldots, e_m$ do not appear in $\hat{\mathcal{C}}$), we can deduce, as we did in section 4.1, that there exist m matchings M_t, $t \in [m]$, of size $|M_t| = \hat{d}$, of linear functions appearing in $\hat{\mathcal{C}}$, such that for every pair $(L, L') \in M_t$, $e_t \in Span\{L, L'\}$. Projecting each linear function in $\hat{\mathcal{C}}$ on the first m coordinates, and using the construction from section 4.1, we see that there exists a $(2, \frac{1}{12}, \frac{1}{4})$-LDC,[7] $E : \mathbb{F}^m \to \mathbb{F}^{2\hat{d}}$. In the notation of the theorem, we have

$$n_2 = 2\hat{d} \leq kd$$

and

$$n_1 = m \geq \frac{r}{2^k} \geq \frac{r}{P(k)\log(d)^{k-3}},$$

as required by the theorem.

4.3. Proof of Claim 4.8. In this section we prove Claim 4.8. The following notation is required for the proof.

4.3.1. Notation. Let $\hat{N}_1, \ldots, \hat{N}_{\hat{k}}$ denote the multiplication gates of $\hat{\mathcal{C}}$. We will treat $\hat{\mathcal{C}}, \hat{N}_1, \ldots, \hat{N}_{\hat{k}}$ also as sets of indices. We shall abuse notation and write

$$\hat{\mathcal{C}} = \{(i,j) \mid i \in [\hat{k}],\, j \in [\hat{d}]\},$$

$$\hat{N}_i = \{(i,j) \mid j \in [\hat{d}]\}.$$

For a set $H \subset \hat{\mathcal{C}}$, we denote with $\text{rank}(H)$ the dimension of the vector space spanned by the linear functions whose indices appear in H. That is,

$$\text{rank}(H) \triangleq \dim\left(Span\{L_{ij} : (i,j) \in H\}\right).$$

For the rest of the proof we will treat subsets of $\hat{\mathcal{C}}$ interchangeably as sets of indices and as (multi)sets of linear functions.

We would next like to define, for each $t \in [m]$, certain subsets of $\hat{\mathcal{C}}$ that capture the structure of $\hat{\mathcal{C}}|_{x_t=0}$. Fix some $t \in [m]$, and consider what happens to $\hat{\mathcal{C}}$ when we set x_t to be zero. The resulting circuit $\hat{\mathcal{C}}|_{x_t=0}$ is generally not simple and can therefore be partitioned (see Definition 3.7) into two disjoint sets: a set containing the indices of the linear functions appearing in $\gcd(\hat{\mathcal{C}}|_{x_t=0})$, and a set containing the indices of the

[7] We could have taken δ to be $\frac{1}{8}$ instead of $\frac{1}{12}$, because the number of multiplication gates is two and not three.

remaining linear functions (these are the linear functions appearing in $sim(\hat{\mathcal{C}}|_{x_t=0})$). To be more precise, denote by δ_t the degree of $\gcd(\hat{\mathcal{C}}|_{x_t=0})$. In every multiplication gate $\hat{N}_i$, there are δ_t linear functions such that the restriction of their product to the linear space defined by the equation $x_t = 0$ is equal to $\gcd(\hat{\mathcal{C}}|_{x_t=0})$. In other words, the product of these δ_t linear functions is equal to $\gcd(\hat{\mathcal{C}})$ under the restriction $x_t = 0$. Denote the set of indices of these functions by G_t^i, and let $R_t^i \triangleq \hat{N}_i \backslash G_t^i$ be the set of indices of the remaining linear functions of this multiplication gate. We thus have (for some choice of constants $\{c_i\}$)

$$sim(\hat{\mathcal{C}}|_{x_t=0}) = \sum_{i=1}^{\hat{k}} c_i \prod_{(i,j)\in R_t^i} (L_{ij}|_{x_t=0})$$

and

$$\forall i \in [\hat{k}], \qquad \gcd(\hat{\mathcal{C}}|_{x_t=0}) = \prod_{(i,j)\in G_t^i} (L_{ij}|_{x_t=0}).$$

We now define, for each $t \in [m]$, the sets $R_t \triangleq \bigcup_{i=1}^{\hat{k}} R_t^i$ and $G_t \triangleq \bigcup_{i=1}^{\hat{k}} G_t^i$. The following claim summarizes some facts that we will later need.

CLAIM 4.10. *For every $t \in [m]$,*

1. $R_t \cap G_t = \emptyset$.
2. $\hat{\mathcal{C}} = R_t \cup G_t$.
3. $|G_t^i| = |G_t^{i'}|$ *for all* i, i'.
4. $|G_t| = \hat{k} \cdot deg(\gcd(\hat{\mathcal{C}}|_{x_t=0})) = \hat{k} \cdot \delta_t$.
5. R_t *contains the indices of the linear functions appearing in* $sim(\hat{\mathcal{C}}|_{x_t=0})$.
6. $r_t = \mathrm{rank}(sim(\hat{\mathcal{C}}|_{x_t=0})) = \mathrm{rank}(R_t)$.

Proof. Items 1 and 2 follow directly from the definition of R_t and G_t as R_t^i and G_t^i give a partition of the indices in $\hat{N}_i$. Items 3 and 4 hold as the linear factors of $\gcd(\hat{\mathcal{C}}|_{x_t=0})$ belong to all the multiplication gates. By definition, R_t^i is the set of linear functions in $\hat{N}_i$ that belong to $sim(\hat{\mathcal{C}}|_{x_t=0})$, which implies item 5. Finally, by definition, $r_t = \mathrm{rank}(sim(\hat{\mathcal{C}}|_{x_t=0}))$ and by item 5 we have that R_t is the set of linear functions appearing in $sim(\hat{\mathcal{C}}|_{x_t=0})$. ∎

4.3.2. The proof. We finally give the proof of Claim 4.8. For convenience we restate it here.

CLAIM 4.8 (restated). *There exists $t \in [m]$ such that*

$$r_t \geq \frac{\hat{r}}{2^{k+1}\log(d)}.$$

We start by assuming that the claim is false. In other words, we assume that for every $t \in [m]$

$$r_t < \frac{\hat{r}}{2^{k+1}\log(d)}. \tag{7}$$

Having defined, for each $t \in [m]$, the sets R_t and G_t, we would now like to show that there exist a small ($\sim \log(d)$) number of sets R_t such that their union covers almost all of $\hat{\mathcal{C}}$. As $\mathrm{rank}(\hat{\mathcal{C}})$ is relatively high, and for each t, $r_t = \mathrm{rank}(R_t)$ is (assumed to be) relatively small, we will get a contradiction. We construct the cover step by step, and in each step we will find an index $t \in [m]$ such that the set R_t covers

at least half the linear functions not yet covered. This idea is made precise by the following claim.

CLAIM 4.11. *For every integer* $1 \le q \le \log(\hat{d})$ *there exist* q *indices* $t_1, \ldots, t_q \in [m]$ *for which*

$$\left| \bigcup_{s=1}^{q} R_{t_s} \right| \ge \hat{k}\hat{d}(1 - 2^{-q}).$$

Proof. The proof proceeds by induction on q.

Base case $q = 1$. To prove the claim for $q = 1$, it is sufficient to show that there exists $t \in [m]$ for which $|R_t| \ge \frac{1}{2}\hat{k}\hat{d}$. Suppose, on the contrary, that for all $t \in [m]$, $|R_t| < \frac{1}{2}\hat{k}\hat{d}$. Claim 4.10 implies that for all $t \in [m]$, $|G_t| \ge \frac{1}{2}\hat{k}\hat{d}$. This in turn implies (by item 3 of Claim 4.10) that for all $t \in [m]$

$$|G_t^1| \ge \frac{1}{2}\hat{d}. \tag{8}$$

The next lemma shows that, under the conditions just described, the linear functions of $\hat{\mathcal{C}}$ "contain" a two-query LDC. We will then apply our results on LDCs from section 2 (namely, Corollary 2.9) to derive a contradiction. Lemma 4.12 is more general than what is required at this point; however, we will need it in its full generality when we handle $q > 1$.

LEMMA 4.12. *Let* $\mathcal{C}$ *be a simple* $\Sigma\Pi\Sigma(k, d)$ *circuit with* n *inputs. Let* $t \in [n]$, $i_t \in [k]$. *Denote* $\delta_t = deg(\gcd(\mathcal{C}|_{x_t=0}))$. *Suppose that the linear functions in* N_{i_t} *are ordered such that*

$$\gcd(\mathcal{C}|_{x_t=0}) = (L_{i_t 1}|_{x_t=0})(x) \cdot (L_{i_t 2}|_{x_t=0})(x) \cdot \cdots \cdot (L_{i_t \delta_t}|_{x_t=0})(x).$$

Then, there exists a matching, $M = \{\mathcal{P}_1, \ldots, \mathcal{P}_g\} \subseteq \mathcal{C} \times \mathcal{C}$, *consisting of* δ_t *disjoint pairs of linear functions, such that for each* $j \in [\delta_t]$,

- *the two linear functions in* $\mathcal{P}_j$ *span* e_t, *and*
- *the first element of* $\mathcal{P}_j$ *is* $L_{i_t j}$.

Proof. As the linear factors of $\gcd(\mathcal{C}|_{x_t=0})$ belong to all the multiplication gates (of $\mathcal{C}|_{x_t=0}$) we can reorder the linear functions in each gate N_i, $i \ne i_t$, such that

$$\forall j \in [\delta_t] \quad : \quad L_{1j}|_{x_t=0} \sim L_{2j}|_{x_t=0} \sim \cdots \sim L_{i_t j}|_{x_t=0} \sim \cdots \sim L_{kj}|_{x_t=0}.$$

As $\mathcal{C}$ is simple, it cannot be the case that, for some j, $L_{i_t j}$ divides all the multiplication gates. Therefore, for every $j \in [\delta_t]$ there exists an index $\alpha(j) \in [k]$ such that $L_{i_t j} \not\sim L_{\alpha(j) j}$. From Claim 4.4 it follows that

$$\forall j \in [\delta_t] \quad : \quad e_t \in Span\{L_{i_t j}, L_{\alpha(j) j}\}.$$

For each $j \in [\delta_t]$ let $\mathcal{P}_j = (L_{i_t j}, L_{\alpha(j) j})$. Set $M = \{\mathcal{P}_1, \ldots, \mathcal{P}_{\delta_t}\}$. It is clear that each $\mathcal{P}_j$ satisfies the two conditions of the lemma and that the $\mathcal{P}_j$ are disjoint. ∎

We continue with the proof of Claim 4.11. From (8) and Lemma 4.12 we conclude that for each $t \in [m]$ there exists a matching $M_t \subset \mathcal{C} \times \mathcal{C}$, containing at least $\frac{1}{2}\hat{d}$ disjoint pairs of linear functions, such that every pair in M_t spans e_t. Corollary 2.9 implies that

$$\frac{1}{2}\hat{d}m \le \sum_{t=1}^{m} |M_t| \le \hat{k}\hat{d}\log(\hat{k}\hat{d}) + \hat{k}\hat{d},$$

which gives

$$m \leq 2\hat{k}\log(\hat{k}\hat{d}) + 2\hat{k} < \log(d)^{k-3}P(k)2^{-k}$$

(for an appropriate choice of $P(k) = 2^{O(k^2)}$). Now, since $m \geq \frac{r}{2^k}$, we have that

$$r < \log(d)^{k-3}P(k),$$

contradicting (4). Therefore our initial assumption was wrong and we conclude that there exists t_1 with $|R_{t_1}| \geq \frac{1}{2}\hat{k}\hat{d}$. This completes the proof of Claim 4.11 for the case of $q = 1$.

Induction step. Let us now assume that we have found $q-1$ indices $t_1, \ldots, t_{q-1} \in [m]$ for which

$$\left|\bigcup_{s=1}^{q-1} R_{t_s}\right| \geq \hat{k}\hat{d}(1 - 2^{-(q-1)}).$$

Let

$$R \triangleq \bigcup_{s=1}^{q-1} R_{t_s}, \tag{9}$$

$$S \triangleq \hat{\mathcal{C}} \backslash R. \tag{10}$$

Then, by our assumption,

$$|S| \leq \hat{k}\hat{d}2^{-(q-1)}. \tag{11}$$

The proof goes along the same lines as the proof for $q = 1$: we show that there exists an index $t \in [m]$ such that R_t covers at least half of S. We will argue that if such an index does not exist, then a contradiction to (4) can be derived. Our main tools in doing so are Lemma 4.12 and Corollary 2.9.

CLAIM 4.13. *There exists $t \in [m]$ such that for all $i \in [\hat{k}]$,*

$$|G_t^i \cap S| < \hat{d}2^{-q}.$$

Roughly, the lemma states that there exists some variable, x_t, such that most of the linear functions in S do not belong to $\gcd(\hat{\mathcal{C}}|_{x_t=0})$. In particular it implies that R_t covers a large fraction of S, as needed.

Proof. Assume, on the contrary, that for every $t \in [m]$ there exists $i_t \in [\hat{k}]$ for which

$$|G_t^{i_t} \cap S| \geq \hat{d}2^{-q}.$$

From Lemma 4.12 we get that, for every $t \in [m]$, there exists a matching M_t, consisting of $\hat{d}2^{-q}$ disjoint pairs of linear functions, such that each pair spans e_t, and that the first element in each pair is in $G_t^{i_t} \cap S$ (from the lemma we actually get that M_t contains $deg(\gcd(\hat{\mathcal{C}}|_{x_t=0}))$ number of pairs, but we are interested only in the pairs whose first element is in $G_t^{i_t} \cap S$).

We would now like to apply Corollary 2.9 on the matchings $\{M_t\}_{t\in[m]}$; however, for our needs, we would also like that all the linear functions in all the matchings will belong to S. We achieve this by projecting all functions in R to zero. As the

dimension of the linear functions in R is small (by our assumption that each r_{t_s} is small) we can find a linear transformation that sends the linear functions in R to zero but leaves many of the linear functions $\{x_t\}$ linearly independent. This is formalized in the next claim.

CLAIM 4.14. *There exists a subset $A \subset [m]$ of size $|A| \geq \frac{m}{2}$ and a linear transformation $\pi : \mathbb{F}^n \to \mathbb{F}^n$ such that*

- $\ker(\pi) = Span(R)$,
- *for all $t \in A$, $\pi(e_t) = e_t$.*

Proof. Calculating, we get that

$$\begin{aligned} \operatorname{rank}(R) &= \operatorname{rank}\left(\bigcup_{s=1}^{q-1} R_{t_s}\right) \leq \sum_{s=1}^{q-1} \operatorname{rank}(R_{t_s}) = \sum_{s=1}^{q-1} r_{t_s} \\ &\leq (q-1)\frac{\hat{r}}{\log(d)2^{k+1}} \leq \frac{r}{2^{k+1}} \leq \frac{m}{2}, \end{aligned} \tag{12}$$

where the second inequality follows from (7), the third inequality follows from the fact that $q \leq \log \hat{d} \leq \log d$ and $\hat{r} \leq r$, and the last inequality follows from the fact that $\frac{r}{2^k} \leq m$. Let $m' = m - \operatorname{rank}(R)$. From (12) we get that $m' \geq m/2$. In particular, there exists a subset $A \subset [m]$, of size $|A| = m'$, such that $Span(\{x_t \mid t \in A\}) \cap Span(R) = \{0\}$. Hence, there exists a linear transformation $\pi : \mathbb{F}^n \to \mathbb{F}^n$ such that

- $\ker(\pi) = Span(R)$,
- for all $t \in A$, $\pi(e_t) = e_t$.

This completes the proof of Claim 4.14. ∎

Let A be the set obtained from the above claim and π the corresponding linear transformation. We assume, w.l.o.g., that $A = [m']$. From here on, we consider only variables x_t such that $t \in [m']$ (i.e., $t \in A$). Fix such $t \in [m']$, and let $M'_t = \pi(M_t)$. In other words, $M'_t = \{(\pi(L), \pi(L'))\}_{(L,L') \in M_t}$. Clearly,

$$|M'_t| = |M_t| \geq \hat{d}2^{-q}. \tag{13}$$

Note that the pairs in M'_t still span e_t, as for any pair $(L.L') \in M_t$, with $e_t = \alpha L + \beta L'$, we have that

$$e_t = \pi(e_t) = \pi(\alpha L + \beta L') = \alpha\pi(L) + \beta\pi(L').$$

Since all the linear functions appearing in R were projected to zero, we know that all the pairs in each M'_t are contained in the multiset[8] $S' \triangleq \{\pi(L) : L \in S\}$.

After this long preparation we apply Corollary 2.9 to the matchings M'_t and derive the following inequality:

$$\sum_{t=1}^{m'} |M'_t| \leq |S'| \log(|S'|) + |S'|. \tag{14}$$

As $|S'| = |S|$ (remember that S' is a multiset), we get by (11) that

$$|S'| \leq \hat{k}\hat{d}2^{-(q-1)}. \tag{15}$$

[8] Note that, as in the proof of Lemma 2.5, we can replace each pair in M'_t that contains the zero vector with a singleton.

By (13), (14), and (15), it follows that

$$m' \cdot (\hat{d}2^{-q}) \leq \sum_{t=1}^{m'} |M'_t| \leq |S'| \log(|S'|) + |S'| \\ \leq \hat{k}\hat{d}2^{-(q-1)} \log(\hat{k}\hat{d}2^{-(q-1)}) + \hat{k}\hat{d}2^{-(q-1)}.$$

From the fact that $k \geq 4$ and $m' \geq m/2$ (and some simple manipulations), we see that for an appropriate choice of $P(k) = 2^{O(k^2)}$

$$m < 2^{-k} P(k) \log(d)^{k-3}.$$

As $m \geq \frac{r}{2^k}$, we get that

$$r < P(k) \log(d)^{k-3},$$

contradicting (4). This completes the proof of Claim 4.13. □

Let us now proceed with the proof of Claim 4.11. Take t_q to be the index described by Claim 4.13, that is,

$$\forall i \in [\hat{k}] \quad : \quad |G^i_{t_q} \cap S| < \hat{d}2^{-q}.$$

In particular,

$$|G_{t_q} \cap S| < \hat{k}\hat{d}2^{-q}.$$

Notice that by (9) and (10) and by the fact that R_{t_q} and G_{t_q} give a partition of $\hat{\mathcal{C}}$, we get that the complement of $\bigcup_{s=1}^{q} R_{t_s}$ is exactly $G_{t_q} \cap S$. From this we get that adding R_{t_q} to R gives

$$\left| \bigcup_{s=1}^{q} R_{t_s} \right| \geq \hat{k}\hat{d}(1 - 2^{-q}).$$

This completes the proof of Claim 4.11. □

Having proved Claim 4.11, we are now just steps away from completing the proof of Claim 4.8. Taking q to be $\lfloor \log(\hat{d}) \rfloor$ in Claim 4.11, we get that there exist indices $t_1, \ldots, t_{\lfloor \log(\hat{d}) \rfloor} \in [m]$ such that

$$\left| \bigcup_{s=1}^{\lfloor \log(\hat{d}) \rfloor} R_{t_s} \right| \geq \hat{k}\hat{d} - 2\hat{k} \ldots.$$

Thus

$$\hat{r} - 2\hat{k} \leq \operatorname{rank} \left(\bigcup_{s=1}^{\lfloor \log(\hat{d}) \rfloor} R_{t_s} \right) \leq \sum_{s=1}^{\lfloor \log(\hat{d}) \rfloor} r_{t_s}.$$

The last inequality tells us that there exists some $t \in [m]$ for which

$$r_t \geq \frac{\hat{r} - 2\hat{k}}{\lfloor \log(\hat{d}) \rfloor} \geq \frac{\hat{r} - 2\hat{k}}{\log(d)}. \tag{16}$$

In order to finish the proof of Claim 4.8 we prove the following inequality.

CLAIM 4.15.

$$\hat{r} - 2\hat{k} \geq \frac{\hat{r}}{2^{k+1}}.$$

Proof. Using (4), we get

$$\hat{r} \geq m \geq 2^{-k} r \geq 2^{-k} P(k) \log(d)^{k-3}.$$

Therefore we can choose $P(k) = 2^{O(k^2)}$ such that

$$\hat{r} > 2\hat{k} \frac{2^{k+1}}{2^{k+1} - 1}.$$

This implies the inequality in the claim. □

Combining Claim 4.15 with (16), we conclude that there exists $t \in [m]$ for which

$$r_t \geq \frac{\hat{r}}{\log(d) 2^{k+1}},$$

which contradicts our initial assumption (7). This completes the proof of Claim 4.8.

5. A structural theorem for zero ΣΠΣ circuits. The main result of this section is a structural theorem for ΣΠΣ circuits which are identically zero. The proof is based on the results of section 4. To ease the notation we will prove our results only for $\Sigma\Pi\Sigma(k, d)$ circuits; however, from Claim 3.11 it will follow that all the results also hold for ΣΠΣ circuits with k multiplication gates of degree d.

THEOREM 5.1 (*structural theorem*). *Let $\mathcal{C} \equiv 0$ be a $\Sigma\Pi\Sigma(k, d)$ circuit. Then, there exists a partition of $[k]$: $T_1, \ldots, T_s \subset [k]$ with the following properties:*

- $\mathcal{C} = \sum_{i=1}^{s} \mathcal{C}_{T_i} = \sum_{i=1}^{s} \gcd(\mathcal{C}_{T_i}) \cdot sim(\mathcal{C}_{T_i})$.
- *For all $i \in [s]$, $sim(\mathcal{C}_{T_i}) \equiv 0$ and is simple and minimal.*
- *For all $i \in [s]$, $\operatorname{rank}(sim(\mathcal{C}_{T_i})) \leq 2^{O(k^2)} \log(d)^{k-2}$.*

In other words, the theorem says that every zero ΣΠΣ circuit can be broken down into zero subcircuits of low rank (ignoring the g.c.d.). This fact will be used in the next section, in which we devise PIT algorithms for ΣΠΣ circuits.

Before giving the proof of the theorem we prove a lemma that bounds the rank of a zero, simple, and minimal ΣΠΣ circuit. Note that Theorem 1.4 follows from this lemma and Claim 3.11.

LEMMA 5.2. *Let $k \geq 3$, $d \geq 2$, and let $\mathcal{C} \equiv 0$ be a simple and minimal $\Sigma\Pi\Sigma(k, d)$ circuit. Then*

$$\operatorname{rank}(\mathcal{C}) \leq 2^{O(k^2)} \log(d)^{k-2}.$$

Proof. From Theorem 4.1 we know that there exists a linear $(2, \frac{1}{12}, \frac{1}{4})$-LDC $E : \mathbb{F}^{n_1} \to \mathbb{F}^{n_2}$ with

$$\frac{\operatorname{rank}(\mathcal{C})}{P(k) \log(d)^{k-3}} \leq n_1 \quad \text{and} \quad n_2 \leq k \cdot d, \quad \text{where} \quad P(k) = 2^{O(k^2)}.$$

Theorem 1.2 now tells us that

$$n_2 \geq 2^{\frac{1}{96} n_1 - 1}.$$

Combining the above inequalities, we get the required bound on $\text{rank}(\mathcal{C})$. ∎

We now use Lemma 5.2 to prove Theorem 5.1.

Proof of Theorem 5.1. Since $\mathcal{C}$ is equal to zero, we can find a partition $T_1, \ldots, T_s \subset [k]$ such that the circuits $\mathcal{C}_{T_1}, \ldots, \mathcal{C}_{T_s}$ are all zero and minimal. Thus, the circuits $sim(\mathcal{C}_{T_1}), \ldots, sim(\mathcal{C}_{T_s})$ are all zero, simple, and minimal. By Lemma 5.2 we get that if $|T_i| \geq 3$ and $\deg(sim(\mathcal{C}_{T_i})) \geq 2$, then

$$\text{rank}(sim(\mathcal{C}_{T_i})) \leq 2^{O(k^2)} \log(d)^{k-2}.$$

If $|T_i| = 2$, then by Claim 4.2 we get that $\deg(sim(\mathcal{C}_{T_i})) = 0$ and so its rank is 1. If $\deg(sim(\mathcal{C}_{T_i})) \leq 1$, then its rank is at most k. Thus, we have covered all the possible cases, and the lemma follows.

6. PIT algorithms. In this section we use the structural theorem (Theorem 5.1), proved in the previous section, to devise the PIT algorithms of Theorem 1.5. Again, to simplify the notation, we give algorithms for $\Sigma\Pi\Sigma(k, d)$ circuits, which work in the same manner also for $\Sigma\Pi\Sigma$ circuits with k multiplication gates of degree d. We state our results for a general k; however, our algorithms will be most applicable when k is a constant.[9]

From Theorem 5.1 we know that every zero $\Sigma\Pi\Sigma$ circuit can be broken down into zero subcircuits whose ranks are small. The next two lemmas show that checking whether these low-rank circuits are zero can be done efficiently.

LEMMA 6.1. *Let $\mathcal{C}$ be a $\Sigma\Pi\Sigma(k, d)$ circuit with $\text{rank}(\mathcal{C}) = r$. Then, there exists a polynomial time algorithm, transforming $\mathcal{C}$ into a $\Sigma\Pi\Sigma(k, d)$ circuit $\mathcal{C}'$, such that*

- *$\mathcal{C} \equiv 0$ iff $\mathcal{C}' \equiv 0$,*
- *$\mathcal{C}'$ contains only r variables.*

Proof. The proof is a direct consequence of Claim 3.14: we apply an invertible linear transformation on $\mathcal{C}$, taking a set of r linearly independent vectors to $e_1, \ldots, e_r$. The transformed circuit will contain only the first r variables and will be zero iff $\mathcal{C}$ is zero. ∎

LEMMA 6.2. *Let $\mathcal{C}$ be a $\Sigma\Pi\Sigma(k, d)$ circuit, and let $r = \text{rank}(\mathcal{C})$, $s = size(\mathcal{C})$. Then we can check if $\mathcal{C} \equiv 0$*

1. *deterministically, in time $\text{poly}(s) \cdot (r + d)^r$;*
2. *probabilistically, in time $\text{poly}(s + \frac{1}{\epsilon})$, using $r \cdot \left(\log(d) + \log(\frac{1}{\epsilon})\right)$ random bits, with error probability ϵ.*

Proof. Using Lemma 6.1, we can transform $\mathcal{C}$ into a circuit $\mathcal{C}'$ with at most r variables, such that $\mathcal{C} \equiv 0$ iff $\mathcal{C}' \equiv 0$. Since $\mathcal{C}'$ contains only r variables, the number of different monomials in $\mathcal{C}'(x)$ is bounded by $\binom{r+d-1}{r-1} < (r + d)^r$. We can thus check if $\mathcal{C}' \equiv 0$ by computing the coefficients of all the monomials and seeing if they are all zero. This can be done in time $\text{poly}(s) \cdot (r + d)^r$. For the second part of the corollary, note that we can also check if $\mathcal{C}' \equiv 0$ probabilistically using the well-known Schwartz–Zippel algorithm [42, 50]. ∎

We are now ready to describe our PIT algorithm for $\Sigma\Pi\Sigma(k, d)$ circuits.

THEOREM 6.3. *Let $\mathcal{C}$ be a $\Sigma\Pi\Sigma(k, d)$ circuit, $s = size(\mathcal{C})$. Then, Algorithm 1 will check if $\mathcal{C} \equiv 0$. Further, the algorithm will run in time $\text{poly}(s) \cdot \exp\left(2^{O(k^2)} \log(d)^{k-1}\right)$.*

Proof. First, note that if $\mathcal{C}$ is nonzero, then the algorithm will never accept. (The algorithm accepts only when a partition of $\mathcal{C}$ into zero subcircuits is found.) Assume that $\mathcal{C}$ is zero. Then, by Theorem 5.1, there exists a partition, $T_1, \ldots, T_s \subset [k]$, of

[9] Our methods give subexponential time ($2^{o(n)}$) algorithms also if $k = o(\sqrt{\log n})$.

ALGORITHM 1. Deterministic algorithm.

input: A $\Sigma\Pi\Sigma(k,d)$ circuit $\mathcal{C}$.

(1) For every subset $T \subset [k]$ do the following:
- (1.1) Compute $r_T = \text{rank}(sim(\mathcal{C}_T))$.
- (1.2) If $r_T \leq 2^{O(k^2)} \log(d)^{k-2}$, then:
- check if $sim(\mathcal{C}_T) \equiv 0$ using part 1 of Lemma 6.2.

(2) If there exists a partition of $[k]$, such that for every set $T \subset [k]$ in the partition $sim(\mathcal{C}_T) \equiv 0$, then **accept**. Otherwise **reject**.

$[k]$ such that the circuits $sim(\mathcal{C}_{T_1}), \ldots, sim(\mathcal{C}_{T_s})$ are all zero and that for all $i \in [s]$ the rank of $sim(\mathcal{C}_{T_i})$ is bounded by $2^{O(k^2)} \log(d)^{k-2}$. Therefore, for every $\mathcal{C}_{T_i}$ we will check whether $sim(\mathcal{C}_{T_i}) \equiv 0$ in step (1.2) of the algorithm. Since we go over all subsets of $[k]$, we are bound to find the above partition and accept.

As for the running time of the algorithm, notice that we apply the algorithm from Lemma 6.2 only on circuits whose rank is smaller than $2^{O(k^2)} \log(d)^{k-2}$. Therefore, by Lemma 6.2, the time spent in each invocation of step (1.2) is at most

$$\text{poly}(s) \cdot \exp\left(2^{O(k^2)} \log(d)^{k-1}\right).$$

Step (1.2) is run at most 2^k times, and so the total running time is also

$$\text{poly}(s) \cdot \exp\left(2^{O(k^2)} \log(d)^{k-1}\right).$$

(The running times of all the other steps of the algorithm are "swallowed up" by the running time of step (1.2).) □

We can modify Algorithm 1 so that it will use a probabilistic check in step (1.2). This will result in a probabilistic PIT algorithm for $\Sigma\Pi\Sigma$ circuits, which uses fewer random bits than previous algorithms.

ALGORITHM 2. Probabilistic algorithm.

input: A $\Sigma\Pi\Sigma(k,d)$ circuit $\mathcal{C}$. An error probability ϵ.

(1) For every subset $T \subset [k]$ do the following:
- (1.1) Compute $r_T = \text{rank}(sim(\mathcal{C}_T))$.
- (1.2) If $r_T \leq 2^{O(k^2)} \log(d)^{k-2}$, then: check if $sim(\mathcal{C}_T) \equiv 0$ probabilistically,
- using part 2 of Lemma 6.2, with error probability $\epsilon 2^{-k}$.

(2) If there exists a partition of $[k]$, such that for every set $T \subset [k]$ in the partition $sim(\mathcal{C}_T) \equiv 0$, then **accept**. Otherwise **reject**.

THEOREM 6.4. *Let $\mathcal{C}$ be a $\Sigma\Pi\Sigma(k,d)$ circuit, $s = size(\mathcal{C})$. Then, Algorithm 2 will check if $\mathcal{C} \equiv 0$. Further, the algorithm will run in time* $\text{poly}\left(s + \frac{2^k}{\epsilon}\right)$, *will use $2^{O(k^2)} \log(d)^{k-1} \log(\frac{1}{\epsilon})$ random bits, and will make an error with probability less than ϵ.*

Proof. Using the same reasoning as in the proof of Theorem 6.3, we see that the algorithm can make an error only if one of the checks in step (1.2) fails. By the union bound, this happens with probability of at most ϵ.

Each check in step (1.2) takes time poly $\left(s+\frac{2^k}{\epsilon}\right)$. And so the total running time is

$$2^k \cdot \text{poly}\left(s+\frac{2^k}{\epsilon}\right) = \text{poly}\left(s+\frac{2^k}{\epsilon}\right).$$

By part 2 of Lemma 6.2, the number of random bits used in step (1.2) is at most $r_T \cdot \left(\log(d)+\log(\frac{1}{\epsilon})\right)$. Since we run the probabilistic check only when $r_T \leq 2^{O(k^2)}\log(d)^{k-2}$, it follows that the number of random bits used in each invocation of step (1.2) is bounded by $2^{O(k^2)}\log(d)^{k-1}\log(\frac{1}{\epsilon})$. As we can use the same random bits in all tests, this is also the total number of random bits needed. □

We restate the last two theorems for the case when k is a constant.

THEOREM 6.5. *Let $\mathcal{C}$ be a $\Sigma\Pi\Sigma(k,d)$ circuit, k a constant, $s = size(\mathcal{C})$. Then we can check if $\mathcal{C} \equiv 0$*

1. *deterministically, in quasi-polynomial time,*
2. *probabilistically, in time* $\text{poly}(s+\frac{1}{\epsilon})$*, using* $O\left(\log(d)^{k-1}\log(\frac{1}{\epsilon})\right)$ *random bits, with error probability ϵ.*

Note that Theorems 6.3, 6.4, and 6.5 imply the first two claims of Theorem 1.5.

6.1. Multilinear circuits. This section deals with a special kind of $\Sigma\Pi\Sigma$ circuit, described by the following definition.

DEFINITION 6.6. *A $\Sigma\Pi\Sigma$ circuit $\mathcal{C}$ is* multilinear *if each of its multiplication gates computes a multilinear polynomial. (A polynomial is multilinear if the degree of every variable is at most one.)*

Let

$$\mathcal{C}(x) = \sum_{i=1}^{k} c_i \prod_{j=1}^{d_i} L_{ij}(x)$$

be a $\Sigma\Pi\Sigma$ circuit. Denote by $V_{ij} \subset [n]$ the set of variables appearing in the linear form L_{ij}. From Definition 6.6 we see that $\mathcal{C}$ is multilinear iff for every $i \in [k]$, and for every $j_1 \neq j_2$, we have

$$V_{ij_1} \cap V_{ij_2} = \emptyset.$$

This condition implies that for every $i \in [k]$ the linear functions $\{L_{ij}\}_{j\in[d_i]}$ are linearly independent. This leads to the following observation.

Observation 6.7. If $\mathcal{C}$ is a multilinear $\Sigma\Pi\Sigma$ circuit of degree d, then $\text{rank}(\mathcal{C}) \geq d$.

Combining this observation and Theorem 1.4, we get the following theorem.

THEOREM 6.8. *Let $\mathcal{C} \equiv 0$ be a multilinear $\Sigma\Pi\Sigma$ circuit with k multiplication gates ($k \geq 3$), which is simple and minimal. Let $d = deg(\mathcal{C})$; then*

$$d \leq 2^{O(k^2)}\log(d)^{k-2}. \tag{17}$$

COROLLARY 6.9. *There exists an integer function $D(k) = 2^{O(k^2)}$ such that every multilinear $\Sigma\Pi\Sigma$ circuit $\mathcal{C}$ with k multiplication gates, which is simple and equal to zero, and of degree $d = deg(\mathcal{C}) > D(k)$, is not minimal.*

Proof. Fix k, and consider (17). This inequality holds only if $d \leq 2^{O(k^2)} = D(k)$. Thus, if $d > D(k)$, then the conditions of Theorem 6.8 are not satisfied. In particular, if $\mathcal{C} \equiv 0$ and is simple, then it is not minimal. □

We can use Corollary 6.9 to improve the algorithm given in section 6, in the case that the given circuit is multilinear.

THEOREM 6.10. *Let $\mathcal{C}$ be a multilinear $\Sigma\Pi\Sigma$ circuit, of size s, with k multiplication gates. We can check if $\mathcal{C} \equiv 0$ in time* $\mathrm{poly}(s) \cdot \exp\left(2^{O(k^2)}\right)$. *Thus, if k is constant, the algorithm runs in polynomial time.*

Proof. The algorithm is the same as Algorithm 1. (It does not matter that our circuit is not a $\Sigma\Pi\Sigma(k,d)$ circuit.) The only difference is that by Corollary 6.9 we only have to consider subcircuits $\mathcal{C}_T$ such that the degree of $sim(\mathcal{C}_T)$ is less than $D(|T|) = 2^{O(k^2)}$. The running time is computed in a similar fashion. In step (1) we go over at most $2^{O(k^2)}$ partitions of $[k]$. Computing the rank of the subcircuit $\mathcal{C}_T$ can be done in polynomial time in s. Finally, by part 1 of Lemma 6.2, step (1.2) requires time $O\left(D(|T|)^{2^{O(k^2)}}\right) = \exp\left(2^{O(k^2)}\right)$. ∎

Theorem 6.10 implies the third claim of Theorem 1.5, thus completing the proof of the theorem.

7. Conclusions and open problems. Finding efficient deterministic PIT algorithms for general arithmetic circuits is a long-standing open problem. We made the first step toward an efficient algorithm for PIT for depth 3 circuits by giving PIT algorithms for depth 3 circuits with bounded top fan-in; however, the general case of depth 3 circuits is still open. In view of [25] it is natural to look for algorithms for PIT for restricted models of arithmetic circuits in which lower bounds are known. Raz [39] proved a quasi-polynomial lower bound for *multilinear* arithmetic formulas computing the determinant and the permanent. Thus, giving PIT algorithms for multilinear formulas is a very interesting, and maybe even a solvable, problem.

The key to our result is the relation we have found between LDCs and depth 3 circuits. Previously, relations between circuits and error correcting codes were known only for bilinear circuits over finite fields [10, 45]. It should be very interesting to find new relations between codes and arithmetic circuits. Another interesting question is whether the relation that we have found is tight. In particular we believe that in Theorem 1.3 one should be able to replace $r/2^{O(k^2)}\log(d)^{k-3}$ with $O(r/k)$. A related question regards how to improve Theorem 1.4. We believe that for minimal and simple circuits over fields of characteristic zero the rank should be $O(k)$. We have found circuits that are minimal and simple, with $r = 3k-2$, and we think that it would be an interesting task to come up with (minimal and simple) circuits that have larger rank. As mentioned in the introduction, [27] showed that over characteristic p there are identically zero depth 3 circuits with top fan-in p for odd p (for $p = 2$ the top fan-in is 3) whose rank is $\log_p(d)$.

We conclude this section with a geometrical problem related to depth 3 circuits with three multiplication gates. The famous Sylvester–Gallai theorem states that every set of n points in the plane having the property that every line that contains two points from the set also contains a third point from the set is contained in a line. Consider the following generalization of the problem (colored version in the projective plane): instead of one set of points we have three different sets. Each set is of size n. The points in the sets correspond to vectors from the r-dimensional sphere, and every two such vectors are linearly independent. The condition on the sets is that every two-dimensional subspace that contains points from two different sets also contains a point from the third set.[10] What can be said about r in this case? Clearly the

[10] Alternatively, the points belong to the r-dimensional projective space, and every line that contains points from two different sets also contains a point from the third set.

r-dimensional sphere can be embedded into the $(r+1)$-dimensional sphere so we only consider "irreducible" arrangements in which the vectors corresponding to the points span the whole space. Using our lower bound on LDCs, we can show that r is at most $O(\log n)$; however, we think that this can be improved. In particular we conjecture that r is bounded (maybe even $r = 2$). If our conjecture is true, then it will serve as evidence that for $k = 3$ the rank of every simple and minimal depth 3 circuit, which is identically zero, is bounded.

We now give an example that shows the relation of the problem to identically zero depth 3 circuits with three multiplication gates. Consider the following equality $x_1^n - x_2^n = \prod_{i=0}^{n-1}(x_1 - w^i x_2)$, where w is a primitive nth root of unity. We get that

$$\sum_{i=1}^{k-1}\prod_{j=0}^{n-1}(x_i - w^j x_{i+1}) + \prod_{j=0}^{n-1}(x_k - w^j x_1) = 0.$$

Notice that this is an identically zero depth 3 circuit with k multiplication gates. For the special case of $k = 3$ we get that

$$\prod_{j=0}^{n-1}(x_1 - w^j x_2) + \prod_{j=0}^{n-1}(x_2 - w^j x_3) + \prod_{j=0}^{n-1}(x_3 - w^j x_1) = 0.$$

Each multiplication gate corresponds to a different set of points. We map each linear function $x_1 - w^j x_2$ from the first gate to the point $(\frac{1}{\sqrt{2}}, \frac{-w^j}{\sqrt{2}}, 0)$; similarly, we map the functions of the second multiplication gate to $\{(0, \frac{1}{\sqrt{2}}, \frac{-w^j}{\sqrt{2}})\}_{j=0,\ldots,n-1}$, etc. Clearly all the points belong to the two-dimensional sphere in $\mathbb{C}^3$. It is easy to see that for each point from the first set (i.e., points coming from the first multiplication gate) and each point from the second set there is a unique point from the third set that belongs to the same two-dimensional space (similarly if we pick the first and third sets, etc.). Therefore this construction satisfies our requirements. Our question is, Can such arrangements be found in higher dimensions?

Acknowledgments. The authors would like to thank Ran Raz and Avi Wigderson for helpful discussions during various stages of this work. A. S. would like to thank Boaz Barak, Valentine Kabanets, and Salil Vadhan for useful conversations on the topic of the work. We are grateful to Ran Raz for many valuable comments that improved the presentation of the results.

REFERENCES

[1] M. Agrawal and S. Biswas, *Primality and identity testing via Chinese remaindering*, J. ACM, 50 (2003), pp. 429–443.

[2] M. Agrawal, N. Kayal, and N. Saxena, *PRIMES is in p*, Ann. of Math., 160 (2004), pp. 781–793.

[3] A. Akavia, S. Goldwasser, and M. Safra, *A unifying approach for proving hardcore predicates using list decoding*, in Proceedings of the 44th IEEE Symposium on Foundations of Computer Science, Cambridge, MA, 2003, pp. 146–155.

[4] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy, *Checking computations in polylogarithmic time*, in Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, ACM Press, New York, 1991, pp. 21–32.

[5] W. Baur and V. Strassen, *The complexity of partial derivatives*, Theoret. Comput. Sci., 22 (1983), pp. 317–330.

[6] D. Beaver and J. Feigenbaum, *Hiding instances in multioracle queries*, in Proceedings of the Seventh Annual Symposium on Theoretical Aspects of Computer Science, Springer, New York, 1990, pp. 37–48.

[7] A. Beimel and Y. Ishai, *Information-theoretic private information retrieval: A unified construction*, in Automata, Languages and Programming, Lecture Notes in Comput. Sci. 2076, Springer, New York, 2001, pp. 912–926.

[8] A. Beimel, Y. Ishai, E. Kushilevitz, and J.-F. Raymond, *Breaking the $O(n^{1/2k-1})$ barrier for information-theoretic private information retrieval*, in Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002, pp. 261–270.

[9] M. Ben-Or and P. Tiwari, *A deterministic algorithm for sparse multivariate polynomial interpolation*, in Proceedings of the 20th Annual ACM Symposium on Theory of Computing, ACM Press, New York, 1988, pp. 301–309.

[10] N. H. Bshouty, *A lower bound for matrix multiplication*, SIAM J. Comput., 18 (1989), pp. 759–765.

[11] S. Chari, P. Rohatgi, and A. Srinivasan, *Randomness-optimal unique element isolation with applications to perfect matching and related problems*, SIAM J. Comput., 24 (1995), pp. 1036–1050.

[12] Z.-Z. Chen and M.-Y. Kao, *Reducing randomness via irrational numbers*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, ACM Press, New York, 1997, pp. 200–209.

[13] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, *Private information retrieval*, in Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science, 1995, pp. 41–50.

[14] A. Deshpande, R. Jain, T. Kavitha, J. Radhakrishnan, and S. V. Lokam, *Lower bounds for adaptive locally decodable codes*, Random Structures Algorithms, 27 (2005), pp. 358–378.

[15] J. Feigenbaum and L. Fortnow, *Random-self-reducibility of complete sets*, SIAM J. Comput., 22 (1993), pp. 994–1005.

[16] P. Gemmell, R. J. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson, *Self-testing/correcting for polynomials and for approximate functions*, in Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, ACM Press, New York, 1991, pp. 33–42.

[17] P. Gemmell and M. Sudan, *Highly resilient correctors for polynomials*, Inform. Process. Lett., 43 (1992), pp. 169–174.

[18] O. Goldreich, H. J. Karloff, L. J. Schulman, and L. Trevisan, *Lower bounds for linear locally decodable codes and private information retrieval*, Comput. Complexity, 15 (2006), pp. 263–296.

[19] O. Goldreich and L. A. Levin, *A hard core predicate for all one way functions*, in Proceedings of the 21st ACM Symposium on Theory of Computing, Seattle, WA, 1989, pp. 25–32.

[20] O. Goldreich, R. Rubinfeld, and M. Sudan, *Learning polynomials with queries: The highly noisy case*, SIAM J. Discrete Math., 13 (2000), pp. 535–570.

[21] D. Grigoriev and M. Karpinski, *An exponential lower bound for depth* 3 *arithmetic circuits*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, ACM Press, New York, 1998, pp. 577–582.

[22] D. Grigoriev, M. Karpinski, and M. F. Singer, *Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields*, SIAM J. Comput., 19 (1990), pp. 1059–1063.

[23] D. Grigoriev and A. A. Razborov, *Exponential complexity lower bounds for depth* 3 *arithmetic circuits in algebras of functions over finite fields*, in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1998, pp. 269–278.

[24] M. Jerrum and M. Snir, *Some Exact Complexity Results for Straight-Line Computations over Semi-Rings*, Technical report CRS-58-80, University of Edinburgh, Edinburgh, UK, 1980.

[25] V. Kabanets and R. Impagliazzo, *Derandomizing polynomial identity tests means proving circuit lower bounds*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, ACM Press, New York, 2003, pp. 355–364.

[26] J. Katz and L. Trevisan, *On the efficiency of local decoding procedures for error-correcting codes*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, ACM Press, New York, 2000, pp. 80–86.

[27] N. Kayal and N. Saxena, *Polynomial identity testing for depth* 3 *circuits*, in Proceedings of the 21st Annual IEEE Conference, 2006, pp. 9–17.

[28] I. Kerenidis and R. de Wolf, *Exponential lower bound for* 2*-query locally decodable codes via a quantum argument*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, ACM Press, New York, 2003, pp. 106–115.

[29] A. R. Klivans and D. Spielman, *Randomness efficient identity testing of multivariate polynomials*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, ACM Press, New York, 2001, pp. 216–223.

[30] L. A. Levin, *One–way functions and pseudorandom generators*, Combinatorica, 7 (1987), pp. 357–363.

[31] D. Lewin and S. Vadhan, *Checking polynomial identities over any field: Towards a derandomization?*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, ACM Press, New York, 1998, pp. 438–447.
[32] R. J. Lipton, *Efficient checking of computations*, in STACS 90: Proceedings of the 7th Annual Symposium on Theoretical Aspects of Computer Science, C. Choffrut and T. Lengauer, eds., Springer, Berlin, Heidelberg, 1990, pp. 207–215.
[33] L. Lovász, *On determinants, matchings, and random algorithms*, in Fundamentals of Comptuation Theory: Proceedings of the Conference on Algebraic, Arithmetic, and Categorial Methods in Computation Theory, Vol. 2, Akademie-Verlag, Berlin, 1979, pp. 565–574.
[34] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani, *Matching is as easy as matrix inversion*, in Proceedings of the Nineteenth Annual ACM Conference on Theory of Computing, ACM Press, New York, 1987, pp. 345–354.
[35] N. Nisan, *Lower bounds for noncommutative computation*, in Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, ACM Press, New York, 1991, pp. 410–418.
[36] N. Nisan and A. Wigderson, *Lower bounds on arithmetic circuits via partial derivatives*, Comput. Complexity, 6 (1997), pp. 217–234.
[37] K. Obata, *Optimal lower bounds for 2-query locally decodable linear codes*, in Proceedings of the Sixth International Workshop on Randomization and Approximation Techniques, Springer, New York, 2002, pp. 39–50.
[38] P. Pudlak, *Communication in bounded depth circuits*, Combinatorica, 14 (1994), pp. 203–216.
[39] R. Raz, *Multi-linear formulas for permanent and determinant are of super-polynomial size*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, ACM Press, New York, 2004, pp. 633–641.
[40] R. Raz and A. Shpilka, *Lower bounds for matrix product, in bounded depth circuits with arbitrary gates*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, ACM Press, New York, 2001, pp. 409–418.
[41] R. Raz and A. Shpilka, *Deterministic polynomial identity testing in non-commutative models*, Comput. Complexity, 14 (2005), pp. 1–19.
[42] J. T. Schwartz, *Fast probabilistic algorithms for verification of polynomial identities*, J. ACM, 27 (1980), pp. 701–717.
[43] E. Shamir and M. Snir, *Lower Bounds on the Number of Multiplications and the Number of Additions in Monotone Computations*, Research report RC6757, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1977.
[44] E. Shamir and M. Snir, *On the depth complexity of formulas*, Math. Systems Theory, 13 (1980), pp. 301–322.
[45] A. Shpilka, *Lower bounds for matrix product*, SIAM J. Comput., 32 (2003), pp. 1185–1200.
[46] A. Shpilka and A. Wigderson, *Depth-3 arithmetic formulae over fields of characteristic zero*, in Proceedings of the 14th Annual IEEE Conference on Computational Complexity, IEEE Computer Society, Piscataway, NJ, 1999, p. 87.
[47] V. Strassen, *Die berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten*, Numer. Math., 20 (1972/73), pp. 238–251.
[48] P. Tiwari and M. Tompa, *A direct version of Shamir and Snir's lower bounds on monotone circuit depth*, Inform. Process. Lett., 49 (1994), pp. 243–248.
[49] L. Trevisan, *Some applications of coding theory in computational complexity*, Quad. Mat. 13 (2004), pp. 347–424.
[50] R. Zippel, *Probabilistic algorithms for sparse polynomials*, in Proceedings of the International Symposium on Symbolic and Algebraic Computation, Springer, New York, 1979, pp. 216–226.

SIAM J. COMPUT.
Vol. 36, No. 5, pp. 1435–1452

THE PROBABILISTIC RELATIONSHIP BETWEEN THE ASSIGNMENT AND ASYMMETRIC TRAVELING SALESMAN PROBLEMS*

ALAN FRIEZE† AND GREGORY B. SORKIN‡

Abstract. We consider the gap between the cost of an optimal assignment in a complete bipartite graph with random edge weights, and the cost of an optimal traveling salesman tour in a complete directed graph with the same edge weights. Using an improved "patching" heuristic, we show that with high probability the gap is $O((\ln n)^2/n)$, and that its expectation is $\Omega(1/n)$. One of the underpinnings of this result is that the largest edge weight in an optimal assignment has expectation $\Theta(\ln n/n)$. A consequence of the small assignment–TSP gap is an $e^{\tilde{O}(\sqrt{n})}$-time algorithm which, with high probability, exactly solves a random asymmetric traveling salesman instance. In addition to the assignment–TSP gap, we also consider the expected gap between the optimal and second-best assignments; it is at least $\Omega(1/n^2)$ and at most $O(\ln n/n^2)$.

Key words. assignment problem, asymmetric traveling salesman problem, average-case analysis of algorithms, random assignment problem, matching, alternating path, patching heuristic, cycle cover, permutation digraph, near-permutation digraph

AMS subject classifications. 90C27, 68Q25, 68W40, 05C80, 60C05

DOI. 10.1137/S0097539701391518

1. Introduction.

The *assignment problem* (AP) is the problem of finding a minimum-weight perfect matching in an edge-weighted bipartite graph. An instance of the AP can be specified by an $n \times n$ matrix $C = (C(i,j))$; here $C(i,j)$ represents the weight (or "cost") of the edge between $i \in X$ and $j \in Y$, where X and Y are disjoint copies of $[n] = \{1, 2, \ldots, n\}$ and X is the set of "left vertices" and Y is the set of "right vertices" in the complete bipartite graph $K_{X,Y}$. The AP can be stated in terms of the matrix C as follows: Find a permutation π of $[n] = \{1, 2, \ldots, n\}$ that minimizes $\sum_{i=1}^{n} C(i, \pi(i))$. Let $\mathrm{AP}(C)$ be the optimal value of the instance of the AP specified by C.

The *asymmetric traveling salesman problem* (ATSP) is the problem of finding a Hamiltonian circuit of minimum weight in an edge-weighted directed graph. An instance of the ATSP can be specified by an $n \times n$ matrix $C = (C(i,j))$ in which $C(i,j)$ denotes the weight of edge (i,j). The ATSP can be stated in terms of the matrix C as follows: Find a *cyclic* permutation π of $[n]$ that minimizes $\sum_{i=1}^{n} C(i, \pi(i))$; here a cyclic permutation is one whose cycle structure consists of a single cycle. Let $\mathrm{ATSP}(C)$ be the optimal value of the instance of the ATSP specified by C.

It is evident from the parallelism between the above two definitions that $\mathrm{AP}(C) \leq \mathrm{ATSP}(C)$. The ATSP is NP-hard, whereas the AP is solvable in time $O(n^3)$. Several authors, e.g., Balas and Toth [5], have investigated whether the AP can be used effectively in a branch-and-bound method to solve the ATSP and have observed that the AP gives extremely good bounds on random instances.

*Received by the editors June 25, 2001; accepted for publication (in revised form) July 21, 2006; published electronically January 26, 2007.

http://www.siam.org/journals/sicomp/36-5/39151.html

†Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh, PA 15213 (alan@random.math.cmu.edu). The research of this author was supported by NSF grant CCR-9818411.

‡Department of Mathematical Sciences, IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 (sorkin@watson.ibm.com).

Karp was able to explain this in an important paper [14]. He assumed that the entries of C were independent uniform [0,1] random variables and proved the surprising result that

$$\mathbf{E}(\mathrm{ATSP}(C) - \mathrm{AP}(C)) = o(1). \tag{1}$$

Since whp[1] $\mathrm{AP}(C) > 1$ we see that this rigorously explains the quality of the assignment bound, a significant victory for probabilistic analysis. Karp proved (1) constructively, analyzing an $O(n^3)$ *patching heuristic* that transformed an optimal AP solution into a good TSP solution. Karp and Steele [15] simplified and sharpened this analysis, and Dyer and Frieze [7] improved the error bound in (1) to $O(\frac{(\ln n)^4}{n \ln \ln n})$. Our first theorem sharpens this further.

THEOREM 1. *Over random cost matrices C,*

$$\mathrm{ATSP}(C) - \mathrm{AP}(C) \le c_1 \frac{(\ln n)^2}{n} \qquad \textit{whp}$$

and

$$\mathbf{E}(\mathrm{ATSP}(C) - \mathrm{AP}(C)) \ge \frac{c_0}{n}.$$

In this paper, $c_0, c_1, \ldots$ are positive absolute constants whose precise values are not too important to us.

As in previous works, we will prove the upper bound in Theorem 1 by analyzing an $O(n^3)$ heuristic which patches an optimal AP solution into a good ATSP solution. We note a related discretized result of Frieze, Karp, and Reed [11], who consider the $C(i,j)$ to be random positive *integers* chosen from a range $[0, L = L(n)]$, and determine for what functions $L(n)$ one has ATSP = AP whp.

Karp and Steele showed that whp the greatest cost of an edge used in the optimal assignment was $O(\frac{(\ln n)^2}{n})$; our next theorem improves upon this. Let $C_{\max} = C_{\max}(C)$ denote the maximum cost of an edge used in an optimal assignment.

THEOREM 2. *Whp over random cost matrices C,*

$$(1 - o(1))\frac{\ln n}{n} \le C_{\max} \le c_2 \frac{\ln n}{n}.$$

It is also of interest to estimate the expected difference Δ_1 between the cheapest and second-cheapest assignments. A better understanding of this difference may be of use in studying the expected performance of branch-and-bound algorithms based on the assignment relaxation.

THEOREM 3. *Over random cost matrices C,*

$$\frac{1}{n^2}(1 - o(1)) \le \mathbf{E}(\Delta_1) \le c_3 \frac{\ln n}{n^2}.$$

The algorithm with the best known worst-case time for solving the ATSP exactly is the $O(n^2 2^n)$ dynamic programming algorithm of Held and Karp [12]. The next theorem describes a probabilistic improvement.

THEOREM 4. *Whp, a random instance of the* ATSP *can be solved exactly in time $e^{\tilde{O}(\sqrt{n})}$.*

Here $\tilde{O}$ is the standard notation for ignoring logarithmic factors.

[1] With high probability, i.e., with probability $1 - o(1)$ as $n \to \infty$.

2. Analysis of the assignment problem. In this section we will prove Theorem 2. The difficult part of the proof—showing that the longest edge in an optimal assignment has length $O(\ln n/n)$—has its essence in Lemma 5 below.

In this section we will exploit the fact that C can be considered to be the matrix giving the edge weights of the complete bipartite graph $K_{X,Y}$. In this interpretation π corresponds to a perfect matching $(i, \pi(i))$, $i \in X$, $\pi(i) \in Y$.

Define the k-neighborhood of a vertex to be the k vertices nearest it, where distance is given by the matrix C; let the k-neighborhood of a set be the union of the k-neighborhoods of its vertices. In particular, for a complete bipartite graph $K_{X,Y}$ and any $S \subseteq X$, $T \subseteq Y$,

(2)
$$N_k(S) \doteq \{y \in Y : \exists s \in S \text{ such that } (s, y) \text{ is one of the } k \text{ shortest arcs out of } s\},$$

(3)
$$N_k(T) \doteq \{x \in X : \exists t \in T \text{ such that } (x, t) \text{ is one of the } k \text{ shortest arcs into } t\}.$$

Given the complete bipartite graph $K_{X,Y}$, any permutation $\pi : X \to Y$ has an associated matching $M_\pi = \{(x, y) : x \in X, y \in Y, y = \pi(x)\}$. Given a cost matrix C and permutation π, define the digraph

$$\vec{D} = \vec{D}_{C,\pi} = (X \cup Y, \vec{E}) \tag{4}$$

consisting of *backward* matching edges and forward "short" edges:

$$\vec{E} = \{(y, x) : y \in Y, x \in X, y = \pi(x)\} \cup \{(x, y) : x \in X, y \in N_{40}(x)\} \cup \{(x, y) : y \in Y, x \in N_{40}(y)\}. \tag{5}$$

We stress that $\vec{D}$ is different from the complete weighted digraph D_n in which we wish to find a minimum length tour.

The arcs of directed paths in $\vec{D}$ are alternately forward $X \to Y$ and backward $Y \to X$ and so they correspond to *alternating paths* with respect to the perfect matching defined by π. Since "adding" an alternating circuit[2] to a matching produces a new matching, finding low-cost alternating paths is key to all our constructions. In particular, an alternating path's backward edges (from the old matching) will be replaced by its forward ones, and so it helps to know (see Lemma 5 below) that given $x \in X$, $y \in Y$ we can find an alternating path from x to y with $O(\log n)$ edges. The forward edges have expected length $O(1/n)$, and we will be able to show (Lemma 7 below) that we can whp be guaranteed to find an alternating path from x to y in which the *difference* in weight between forward and backward edges is $O(\log n/n)$. It is then simple to prove the upper bound in Theorem 2. A long edge can be removed by the use of such an alternating path.

LEMMA 5. *Whp over random cost matrices C, for every permutation π, the (unweighted) diameter of $\vec{D} = \vec{D}_{C,\pi}$ is at most $k_0 = \lceil 3 \log_4 n \rceil$.*

Proof. For $S \subseteq X$, $T \subseteq Y$, let

$$N_{\vec{D}}(S) = \{y \in Y : \exists s \in S \text{ such that } (s, y) \in \vec{E}\},$$
$$N_{\vec{D}}(T) = \{x \in X : \exists t \in T \text{ such that } (x, t) \in \vec{E}\}.$$

[2]We use circuit to distinguish from the cycles of the permutations.

We first prove an expansion property: Whp, for all $S \subseteq X$ with $|S| \le \lceil n/5 \rceil$, $|N_{\vec{D}}(S)| \ge 4|S|$. (Note that only the cheap edges out of S, and not the backward matching edges into it, are involved here. Thus $N_{\vec{D}}(S), N_{\vec{D}}(T)$ do not depend on π.)

$$\begin{aligned}
\mathbf{Pr}(\exists S:\ |S| \le \lceil n/5 \rceil\,,\ |N_{\vec{D}}(S)| < 4|S|) &\le \sum_{s=1}^{\lceil n/5 \rceil} \binom{n}{s}\binom{n}{4s}\left(\frac{\binom{4s}{40}}{\binom{n}{40}}\right)^s \\
&\le \sum_{s=1}^{\lceil n/5 \rceil} \left(\frac{ne}{s}\right)^s \left(\frac{ne}{4s}\right)^{4s} \left(\frac{4s}{n}\right)^{40s} \\
&= \sum_{s=1}^{\lceil n/5 \rceil} \left(\frac{e^5 4^{36} s^{35}}{n^{35}}\right)^s \\
&= o(1). \qquad (6)
\end{aligned}$$

Explanation. Summing over all possible ways of choosing s vertices and $4s$ "targets," we take the probability that for each of the s vertices, all 40 out-edges fall among the $4s$ out of the n possibilities.

Similarly, whp, for all $T \subseteq Y$ with $|T| \le \lceil n/5 \rceil$, $|N_{\vec{D}}(T)| \ge 4|T|$. (Again only the cheap edges, not the matching edges, are involved.) Thus by the union bound, whp both these events hold. In the remainder of this proof we assume that we are in this "good" case, in which all small sets S and T have a large vertex expansion.

Now, choose an arbitrary $x \in X$, and define $S_0, S_1, S_2, \ldots$ as the endpoints of all cheap alternating paths starting from x and of lengths $0, 2, 4, \ldots$. That is,

$$S_0 = \{x\} \text{ and } S_i = \pi^{-1}(N_{\vec{D}}(S_{i-1})).$$

Since we are in the good case, $|S_i| \ge 4|S_{i-1}|$ provided $|S_{i-1}| \le n/5$, and so there exists a smallest index i_S such that $|S_{i_S-1}| > n/5$, and $i_S - 1 \le \log_4(n/5) \le \log_4 n - 1$. Arbitrarily discard vertices from S_{i_S-1} to create a smaller set S'_{i_S-1} with $|S'_{i_S-1}| = \lceil n/5 \rceil$, so that $S'_{i_S} = N_{\vec{D}}(S'_{i_S-1})$ has cardinality $|S'_{i_S}| \ge 4|S'_{i_S-1}| \ge 4n/5$.

Similarly, for an arbitrary $y \in Y$, define $T_0, T_1, \ldots$ by

$$T_0 = \{y\} \text{ and } T_i = \pi(N_{\vec{D}}(T_{i-1})).$$

Again, we will find an index $i_T \le \log_4 n$ whose modified set has cardinality $|T'_{i_T}| \ge 4n/5$.

With both $|S'_{i_S}|$ and $|T'_{i_T}|$ larger than $n/2$, there must be some $x' \in S'_{i_S}$ for which $y' = \pi(x') \in T'_{i_T}$. This establishes the existence of an alternating walk and hence (removing any circuits) an alternating path of length at most $2(i_S + i_T) \le 2\log_4 n$ from x to y in $\vec{D}$.

We have proved there is a short path from any $x \in X$ to any $y \in Y$. A short path from x to x' both in X can be formed by finding a path from x to $y = \pi(x')$ and appending the backward edge to x'; a path from y to x' by starting with the backward edge from y to $x = \pi^{-1}(y)$ and then pursuing a path to x'; and a path from y to y' by taking a path from y to $x' = \pi^{-1}(y')$ and discarding its final backward edge. ∎

Let the weight of a forward edge (x, y) be $C(x, y)$ and the weight of a backward edge (y, x) be $-C(x, y)$.

We will need the following inequality, Lemma 4.2(b) of [10].

LEMMA 6. *Suppose that* $k_1+k_2+\cdots+k_M \le a\ln N$, *and* $Y_1, Y_2, \ldots, Y_M$ *are independent random variables with* Y_i *distributed as the* k_i*th minimum of* N *independent uniform* [0,1] *random variables. If* $\lambda > 1$, *then*

$$\mathbf{Pr}\left(Y_1+\cdots+Y_M \ge \frac{\lambda a \ln N}{N+1}\right) \le N^{a(1+\ln\lambda-\lambda)}.$$

LEMMA 7. *Whp over random* C, *for all* π, *the weighted diameter of* $\vec{D} = \vec{D}_{C,\pi}$ *is* $\le c_2 \frac{\ln n}{n}$.

Proof. Let

$$Z_1 = \max\left\{\sum_{i=0}^{k} C(x_i, y_i) - \sum_{i=0}^{k-1} C(y_i, x_{i+1})\right\}, \tag{7}$$

where the maximum is over sequences $x_0, y_0, x_1, \ldots, x_k, y_k$ where (x_i, y_i) is one of the 40 shortest arcs leaving x_i for $i = 0, 1, \ldots, k \le k_0 = \lceil 3\log_4 n\rceil$, and (y_i, x_{i+1}) is a backward matching edge.

We compute an upper bound on the probability that Z_1 is large. For any $\zeta > 0$ we have

$$\mathbf{Pr}\left(Z_1 \ge \zeta\frac{\ln n}{n}\right) \le \sum_{k=0}^{k_0} n^{2k+2}\frac{1}{(n-1)^{k+1}} \times \int_{y=0}^{\infty}\left[\frac{1}{(k-1)!}\left(\frac{y\ln n}{n}\right)^{k-1} \sum_{\rho_0+\rho_1+\cdots+\rho_k\le 40(k+1)} q(\rho_0,\rho_1,\ldots,\rho_k;\zeta+y)\right] dy,$$

where

$$q(\rho_0,\rho_1,\ldots,\rho_k;\eta) = \mathbf{Pr}\left(X_0+X_1+\cdots+X_k \ge \eta\frac{\ln n}{n}\right),$$

$X_0, X_1, \ldots, X_k$ are independent, and X_j is distributed as the ρ_jth minimum of $n-1$ uniform [0,1] random variables. (When $k=0$, there is no term $\frac{1}{(k-1)!}(\frac{y\ln n}{n})^{k-1}$.)

Explanation. We have $\le n^{2k+2}$ choices for the sequence $x_0, y_0, x_1, \ldots, x_k, y_k$. The term $\frac{1}{(k-1)!}(\frac{y\ln n}{n})^{k-1}dy$ bounds the probability that the sum of k independent uniforms, $C(y_0,x_1)+\cdots+C(y_{k-1},x_k)$, is in $\frac{\ln n}{n}[y, y+dy]$. (We approximate this probability by the area of the simplex face $\{y_1+y_2+\cdots+y_k = \frac{y\ln n}{n},\ y_1, y_2, \ldots, y_k \ge 0\}$ multiplied by dy.) We integrate over y. $\frac{1}{n-1}$ is the probability that (x_i, y_i) is the ρ_ith shortest edge leaving x_i, and these events are independent for $0 \le i \le k$. The final summation bounds the probability that the associated edge lengths sum to at least $\frac{(\zeta+y)\ln n}{n}$.

It follows from Lemma 6 that if ζ is sufficiently large, then for all $y \ge 0$, $q(\rho_1,\ldots,\rho_k;\zeta+y) \le n^{-(\zeta+y)/2}$, and since the number of choices for $\rho_0, \rho_1, \ldots, \rho_k$

is at most $\binom{41k+40}{k}$ (the number of nonnegative integral solutions to $x_0 + x_1 + \cdots + x_{k+1} = 40(k+1)$) we have

$$\begin{aligned}\mathbf{Pr}\left(Z_1 \geq \zeta\frac{\ln n}{n}\right) &\leq 2n^{2-\zeta/2}\sum_{k=0}^{k_0}\frac{(\ln n)^{k-1}}{(k-1)!}\binom{42k}{k}\int_{y=0}^{\infty} y^{k-1}n^{-y/2}dy\\ &\leq 2n^{2-\zeta/2}\sum_{k=0}^{k_0}\frac{(\ln n)^{k-1}}{(k-1)!}\left(\frac{42e}{\ln n}\right)^k\Gamma(k)\\ &\leq 2n^{2-\zeta/2}(k_0+1)(42e)^{k_0+2}\\ &= o(n^{-2}).\end{aligned}$$

Similarly, whp $Z_2 \leq \zeta\frac{\ln n}{n}$, where Z_2 is the maximum of the RHS of expression (7) over sequences $y_0, x_0, y_1, \ldots, y_k, x_k$ where (x_i, y_{i+1}) is one of the 40 shortest arcs leaving y_i.

If $x \in X$ and $y \in Y$, then Lemma 5 implies that whp there is a path of length at most k_0 from x to y, and by the above it will whp have length at most $Z_1 \leq \zeta\frac{\ln n}{n}$. For paths from $y \in Y$ to $x \in X$ we bound the path length with Z_2. For a path from $x \in X$ to $x' \in X$ we find a low weight path P' from x to $y' = \pi(x')$ and extend it to x', at lower cost. (x' cannot be on P'; otherwise y' appears at least twice on P'.) For a path between $y \in Y$ and $y' \in Y$ we add (y, x) to a low weight path from $x = \pi^{-1}(y)$ to y'. □

We can now prove Theorem 2, repeated here for convenience.

THEOREM 2. *Whp over random cost matrices C,*

$$(1 - o(1))\frac{\ln n}{n} \leq C_{\max} \leq c_2\frac{\ln n}{n}.$$

Proof. The lower bound follows easily from the fact that $\frac{\ln n}{n}$ is the threshold probability for a random bipartite graph to have a perfect matching, as shown by Erdős and Rényi [9].

For the upper bound, define $\vec{D} = \vec{D}_{C,\pi}$ as per (4) and (5). From the preceding lemma, we can assume the existence of a cheap alternating path P_x from any x to $\pi(x)$,

$$x = x_0, y_0, x_1, y_1, \ldots, x_k, y_k = \pi(x), \qquad k \leq k_0 + 1, \tag{8}$$

consisting of cheap forward edges and backward matching edges.

Suppose any edge in the optimal matching had cost $C(x, \pi(x)) > \frac{c_2 \ln n}{n}$. $P(x)$, followed by the backward edge $(\pi(x), x)$, is an alternating circuit, which in this case has cost $\leq \frac{c_2 \ln n}{n} - C(x, \pi(x)) < 0$. "Adding" the alternating circuit to the matching (adding its forward edges to the matching and deleting the backward ones from it) results in a new matching of lower cost, contradicting the hypothesis that the original matching was optimal. □

3. Analysis of the traveling salesman problem. Our goal in this section is to prove Theorem 1, recalled here for convenience.

THEOREM 1. *Over random cost matrices C,*

$$\mathrm{ATSP}(C) - \mathrm{AP}(C) \leq c_1\frac{(\ln n)^2}{n} \qquad \textit{whp}$$

and

$$\mathbf{E}(\mathrm{ATSP}(C) - \mathrm{AP}(C)) \geq \frac{c_0}{n}.$$

We prove the theorem's first assertion in sections 3.1 through 3.3, and the second in section 3.4.

If $(i, \pi(i))$, $i \in X$, is a perfect matching of $K_{X,Y}$, then $(i, \pi(i))$ also defines a *permutation digraph* (PD), i.e., a set of vertex-disjoint directed cycles which cover all n vertices of the complete directed graph $\vec{K}_n$ associated with $K_{X,Y}$. The *size* $|\pi|$ of π is the number of cycles in the permutation.

Similarly, a near-perfect matching gives rise to a near-permutation digraph (NPD), i.e., a digraph obtained from a PD by removing one edge. Thus an NPD Γ consists of any number of directed cycles and a single directed path PATH(Γ).

The edges (i, j) will be colored: red for $C(i, j) \in [0, c_2 \frac{\ln n}{n}]$; blue for $C(i, j) \in (c_2 \frac{\ln n}{n}, 2c_2 \frac{\ln n}{n}]$; green for $C(i, j) \in (2c_2 \frac{\ln n}{n}, 3c_2 \frac{\ln n}{n}]$; and black otherwise.

We will use a *three phase* method as outlined below.

Phase 1. Solve the assignment problem to obtain an optimal assignment Π and perfect matching M_π in $K_{X,Y}$; whp, only red edges are used.

Phase 2. Whp, at cost $O(\frac{(\ln n)^2}{n})$ we increase the minimum cycle length in the PD to at least $n_0 = \lceil \frac{n \ln \ln n}{\ln n} \rceil$. We use red and blue edges.

Phase 3. Whp, at cost $O(\frac{(\ln n)^2}{n})$ we convert the Phase 2 PD to a tour. We use green edges.

The point is that each phase uses cheap edges that are essentially probabilistically independent from those in earlier phases. Also, we need Phase 2 because it is hard to cheaply patch together short cycles.

3.1. Phase 1. That whp only red edges are used in an optimal assignment is immediate from Theorem 2. Furthermore, given the optimal assignment, and conditioning upon its use only of red edges, the edges which are not red can be thought of as having *independent* lengths, uniform in $[c_2 \frac{\ln n}{n}, 1]$.

Also, whp, the optimal assignment π's associated PD Π_1 is of size $|\Pi_1| \leq 2 \ln n$. This holds because π is a random permutation; we will elaborate on this in Phase 2.

3.2. Phase 2. In this phase, to increase the minimum cycle length in the PD, we will deal with each small cycle in turn. Let us describe the essence of how one small cycle C of a PD is repaired, setting aside the combinatorial and probabilistic issues. One edge (a, b) of the cycle is chosen. From vertex a, a path $P_a = (x_0 = a, y_0, x_1, y_1, \ldots, x_k)$ is grown, using red forward non-PD edges (starting with an edge out of a) alternating with PD edges traversed backward (see Figure 1). P_a corresponds to an alternating path in the bipartite digraph $\vec{D}$. From b a similar path $P_b = (y'_0 = b, x'_0, y'_1, x'_1, \ldots, y'_l)$ is grown, alternating non-PD edges traversed backward (starting with a nonmatching edge into b traversed backward) with PD edges traversed forward. The a-path, followed by the edge joining its terminal x_k to the terminal y'_l of the b-path, followed by the reversed b-path, followed by the edge (b, a), defines an alternating circuit. The "sum" of this circuit and the original PD is a new PD. If the two paths, and the edge (x_k, y'_l) bridging their endpoints, are cheap, the new PD is not much more expensive than the old one. Furthermore, if done properly, we will have at least one less small cycle.

It is important to see how these changes are viewed in the context of our PD. Consider the sum of the original PD and the path P_a as this path grows. After removing (a, b) alone we have an NPD which contains a path $Q = C \setminus \{(a, b)\}$ from b to a. As we grow P_a we change Q. It will always start at b, and at some stage suppose its other endpoint is x_i. Suppose now that the next y_i lies on a PD cycle A which is disjoint from Q. In this case x_{i+1} is the predecessor of y_i on A, and Q grows by

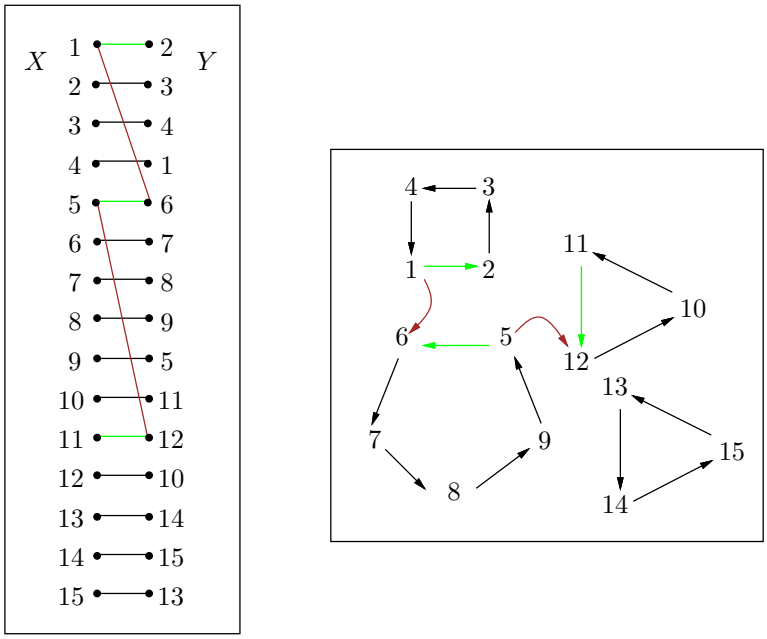


FIG. 1. *In the left box is a bipartite graph with matching edges shown as horizontals (black or grey). The right box shows the corresponding oriented cycle cover indicated by straight arrows (black or grey), for example, the arrow* $1 \longrightarrow 2$ *indicating that* X *vertex* 1 *is matched to* Y *vertex* 2. *We imagine that only the pentagon is a "long" cycle, and all the others are short cycles needing repair.*

Suppose that to repair the cycle $1, 2, 3, 4$ *we had selected edge* $(1, 2)$. *In the bipartite graph we find a path, rooted at* 1, *of cheap (red) forward edges (shown as slanted grey lines in the left box) alternating with matching edges (horizontal solid lines), in this case the path* $x_1, y_6, x_5, y_{12}, x_{11}$. *The right box shows the NPD obtained from this alternating path—the light grey edges being removed from the cycle cover and the bent edges added to it.*

Suppose that in symmetry to the alternating path $1, 6, 5, 12, 11$ *we found a cheap alternating path such as* $2, 7, 8$, *this time taking nonmatching edges backward and matching edges forward. Furthermore suppose the edge* $(11, 8)$ *happened to be cheap (red or blue). Then taking the first path, the edge* $(11, 8)$, *the reversal of the second path, and the reversed edge* $(1, 2)$ *gives a cheap alternating cycle* $1, 6, 5, 12, 11, 8, 7, 2$.

Adding this alternating cycle to the PD repairs the short cycle $1, 2, 3, 4$, *yielding instead the cycle* $1, 6, 7, 2, 3, 4$. *In this case we also serendipitously repair the cycle* $10, 11, 12$, *instead getting the cycle* $8, 9, 5, 12, 10, 11$. *The cycle* $13, 14, 15$ *(like most cycles, most of the time) is uninvolved.*

adding the path $A \setminus \{(x_{i+1}, y_i)\}$. On the other hand, if y_i is a vertex of Q, then x_{i+1} will be the predecessor of y_i on Q. In this case removing (x_{i+1}, y_i) leaves a shorter Q plus a new cycle which has been "split off."

When we construct P_b, we can think of starting with the NPD, and in particular with the Q constructed from P_a, and now extending Q from the b end. As long as no cycles split off are small, and either y_0 or x'_0 is on a large cycle, the new cycle containing a and b, and any other new cycles formed, will be large. We will try to arrange for this to be the case, otherwise declaring the attempt a failure.

In fact we will construct this alternating path from a to b as the conjunction of a path P_a directed out of a to some vertex z and a path P_b directed from some vertex z' into b, plus the edge (z, z'). We will require (z, z') to be a blue edge. In order to find such a pair z, z' whp, we will have to produce *many* candidates for P_a, P_b.

If we fail to remedy a small cycle, then the entire algorithm fails. If we succeed,

we proceed to the next small cycle, until all small cycles are repaired.

Of course the "new" PD of one case becomes the "original" PD of the next one, and the most difficult part of the analysis will be to handle conditioning that might be introduced by this evolving cycle structure. (We will rely on the fact that a PD is induced by a bipartite matching *when the two sets of vertices are put into correspondence by a labeling*, and until that labeling is established, the PD and the matching are in a sense independent.)

The reader will notice that we are using red edges twice, once to find the optimal assignment π and again in Phase 2 to get rid of small cycles. Thus we have to be very careful about conditioning. It would be much simpler to give up another $\log n$ factor by using blue edges for this phase, but we feel that we are getting close to the true upper bound and that the effort is worthwhile. As we will see, the crucial new idea is to condition on the sizes of the cycles in the optimum PD for the assignment problem.

The first detail is the construction of the cheap alternating paths out of vertices a and b. Paths alternating with respect to a PD as described above are—equivalently—alternating with respect to the corresponding bipartite matching. We begin by finding a cheap "alternating tree" (really an alternating directed acyclic graph, or DAG), rooted at a, containing many cheap alternating paths. After doing the same for b, we hope to find a cheap edge between some a-leaf and some b-leaf, and we use the paths determined by these leaves.

To define the trees, recall the definitions (2) and (3) of $N_k(S)$ and $N_k(T)$. For the remainder of this section let K be a suitably large constant. Let $E_K = \{(x,y) \in E(K_{X,Y}) : y \in N_K(x) \text{ or } x \in N_K(y)\}$.

LEMMA 8. *For any fixed K, whp over random matrices C, every set of $s \le s^* = \frac{\ln n}{2 \ln \ln n}$ vertices spans at most s edges from E_K.*

Proof. Since K is large, we know that whp every edge in E_K has length at most $2K\frac{\ln n}{n}$. (Chernoff bounds imply that whp there are at least K edges of length $\le 2K\frac{\ln n}{n}$ leaving and entering *every* vertex.) Starting with the $o(1)$ failure probability for that event, the probability there exists a small set S containing $|S|+1$ edges (even counting loops) is at most

$$\begin{aligned}
&o(1) + \sum_{s=1}^{s^*} \binom{n}{s}\binom{s^2}{s+1}\left(2K\frac{\ln n}{n}\right)^{s+1} \\
&\le o(1) + \sum_{s=1}^{s^*} \left(\frac{ne}{s}\right)^s \left(\frac{s^2 e}{s+1}\right)^{s+1}\left(2K\frac{\ln n}{n}\right)^{s+1} \\
&\le o(1) + \sum_{s=1}^{s^*} \frac{s}{n}(2e^2 K \ln n)^{s+1} \\
&= o(1)
\end{aligned}$$

(that is, the number of ways of choosing s vertices, the number of ways of choosing $s+1$ edges from the s^2 edge slots including loops, times the probability that all these edge slots are realized by edges). ∎

LEMMA 9. *Whp over random matrices C, for all $S \subseteq X$, $T \subseteq Y$, with $|S|, |T| \le n^{3/4}$,*

$$|N_K(S)| \ge (K-2)|S| \text{ and } N_K(T)| \ge (K-2)|T|. \tag{9}$$

Proof. Just as in deriving (6),

$$\begin{aligned}
&\mathbf{Pr}(\exists S \text{ or } T:\ \neg(9)) \\
&\le 2\sum_{s=1}^{n^{3/4}} \binom{n}{s}\binom{n}{(K-2)s}\left(\frac{\binom{(K-2)s}{K}}{\binom{n}{K}}\right)^s \\
&\le 2\sum_{s=1}^{n^{3/4}} \left(\frac{ne}{s}\right)^s \left(\frac{ne}{(K-2)s}\right)^{(K-2)s} \left(\frac{(K-2)s}{n}\right)^{Ks} \\
&= 2\sum_{s=1}^{n^{3/4}} \left(\frac{e^{K-1}(K-2)^2 s}{n}\right)^s \\
&= o(1). \qquad \square
\end{aligned}$$

We say that a cycle C of Π_1 is *small* if $|C| < n_0$; recall that we defined

$$n_0 = \left\lceil \frac{n \ln\ln n}{\ln n} \right\rceil.$$

Detailed analyses of random permutations have been performed by Arratia, Barbour, and Tavaré [3], in which the joint distribution of counts k_i of cycles of length i is approximated by independent Poissons $Z_i \sim \mathrm{Pois}(1/i)$, and by Arratia and Tavaré [4], who provide a tighter bound on the variation distance between the true distribution and the Poisson approximation. From these (or more elementary analyses) we observe first that the expected number of vertices on small cycles is $n_0 - 1$, and so with probability $1 - O(n_0/n)$,

(10) there are less than $2n_0$ vertices on small cycles.

(The distance between the true distribution and independent Poisson estimate dominates the bound; the probability the Poissons exceed their expectation of n_0 by a factor of 2 is much smaller.) Assume from now on that π satisfies (10).

Let the small cycles of Π_1 be $C_1, C_2, \ldots, C_\lambda$. At the start of Phase 2, from each small cycle C we choose an edge (a, b) of C. Let the chosen edges be $(a_i, b_i), i = 1, 2, \ldots, \lambda$. We now describe how we try to remove a C_i without creating any new small cycles. (See Figure 1.)

As before, we use expansion to create many short alternating paths. Let a bijection (matching) ρ_i between X and Y be given, and let one matching edge (a_i, b_i) be specified. Define branching factors

$$r_1 = \lceil K \ln n \rceil \qquad \text{and} \qquad r_t = K,$$

respectively, for a first generation, $t = 1$, and for all subsequent generations, $t \ge 2$. For each i we construct a pair of "trees" (actually DAGs), S_i rooted at a_i and T_i at b_i, which we will use to modify bijection $\rho = \rho_i$. Their depth-t nodes consist of the sets $S_i^{(t)}$ and $T_i^{(t)}$, respectively. The depth-0 node sets are the singletons

$$S_i^{(0)} = \{a_i\} \text{ and } T_i^{(0)} = \{b_i\}.$$

Define

$$s_0 = \frac{\ln n}{12 \ln\ln n},$$

and for $1 \le t \le s_0$ let

$$S_i^{(t)} = \rho^{-1}(N_{r_t}(S_i^{(t-1)})) \text{ and } T_i^{(t)} = \rho(N_{r_t}(T_i^{(t-1)})).$$

For $t > s_0$ let

$$S_i^{(t)} = \rho^{-1}(N_{r_t}(S_i^{(t-1)})) \setminus \left(\bigcup_{i'=1}^{i-1} \bigcup_{u=1}^{\ln\ln n} S_{i'}^{(u)} \cup \bigcup_{i'=1}^{i-1} \bigcup_{u=1}^{\ln\ln n} \rho^{-1}(T_{i'}^{(u)}) \right)$$

$$T_i^{(t)} = \rho(N_{r_t}(T_i^{(t-1)})) \setminus \left(\bigcup_{i'=1}^{i-1} \bigcup_{u=1}^{\ln\ln n} T_{i'}^{(u)} \cup \bigcup_{i'=1}^{i-1} \bigcup_{u=1}^{\ln\ln n} \rho(S_{i'}^{(u)}) \right).$$

It is immediate that $|S_i^{(1)}| = |T_i^{(1)}| = r_1$. For $t \ge 2$ and (as will always be the case) $i < 4\ln n$, it follows from Lemmas 8 and 9 that whp $|S_i^{(t)}| \ge (K-4)|S_i^{(t-1)}|$ and $|T_i^{(t)}| \ge (K-4)|T_i^{(t-1)}|$ as long as both $S_i^{(t-1)}$ and $T_i^{(t-1)}$ are of size at most $n^{3/4}$. Indeed, Lemma 8 implies that whp for all $i' < i$,

$$\left| \bigcup_{t=1}^{s_0} S_i^{(t)} \cap \bigcup_{t=1}^{s_0} S_{i'}^{(t)} \right| \le 2. \tag{11}$$

(Otherwise, if the repeated points are x_1, x_2, x_3, then the paths between $a_i, a_{i'}$ and x_1, x_2, x_3 form a bicyclic graph with at most $6s_0$ vertices, contradicting Lemma 8.) Combining this with Lemma 9 means that for $t \le s_0$, $|S_i^{(t)}| \ge (K-4)|S_i^{(t-1)}|$. For generations $t > s_0$, for each i' the sets subtracted out are of size $O(K^{\ln\ln n})$, and so as long as $i < 4\ln n$, in all, the sets subtracted out are of size $O(K^{\ln\ln n}\ln n)$, much smaller than the size $\Omega((K-4)^{s_0})$ to which the set $S_i^{(t)}$ has by then grown. By throwing away vertices if necessary, we can assume that $|S_i^{(t)}| = (K-4)|S_i^{(t-1)}|$ and $|T_i^{(t)}| = (K-4)|T_i^{(t-1)}|$. Thus if

$$\tau = \lceil 1 + \log_{K-4}(n^{3/4}/\lceil K\ln n\rceil)\rceil,$$

then whp

$$\forall i:\ n^{3/4} \le |S_i^{(\tau)}| = |T_i^{(\tau)}| \le Kn^{3/4}. \tag{12}$$

Each $x \in S_i^{(t)}$ defines a *walk* from a_i to x, of length $2t$, which is alternating with respect to the matching M_ρ; prune it to define a *path* $P[i,x]$. Similarly, each $y \in T_i^{(t)}$ defines a path $Q[i,y]$ from y to b_i, of length at most $2t$, which is alternating with respect to M_ρ.

Suppose that we have removed $C_1, C_2, \ldots, C_{i-1}$ and that the original permutation π has become $\rho = \rho_i$. Assume that we have not already serendipitously removed C_i as well. Let (a_i, b_i) be the chosen edge of C_i.

Each alternating path $P[i,x]$ starts with a "forward" edge which is one of the $K\ln n$ shortest edges leaving a_i (the first branching factor was $r_1 = K\ln n$), has up to $\tau - 1$ other forward edges,[3] each of which is one of the K shortest edges leaving a vertex, and has an additional up to τ "backward" matching edges (edges in M_ρ). A symmetric condition holds for $Q[i,y]$.

[3]Fewer than $\tau - 1$ if the path $P[i,x]$ resulted from nontrivially pruning a $(\tau-1)$-long walk.

It follows from the proof of Lemma 7 that whp each of these paths is such that the total length of its forward edges minus the total length of its backward edges is bounded by $c_4 \frac{\ln n}{n}$.

We now see that if we find $\xi_i \in S_i^{(\tau)}$ and $\eta_i \in T_i^{(\tau)}$ such that (ξ_i, η_i) is red or blue (recall the definition from the start of section 3) then it—together with the edge (a_i, b_i) and the paths $P[i, \xi_i]$ and $Q[i, \eta_i]$—defines an alternating cycle whose action on the current perfect matching increases the matching's cost by at most $(2c_4 + 2c_2)\frac{\ln n}{n}$. We now show that we can whp find at least one such alternating cycle whose action *does not create any new small cycles*. Furthermore, if such a path contains an edge of $C_{i'}$, $i' > i$, then this alternating cycle will also remove the small cycle $C_{i'}$.

Let ϕ be a random permutation of $[n]$ associating the vertices of X to those of Y, and let matrix $\hat{C}$ be defined by $\hat{C}(i,j) = C(i, \phi(j))$. If ψ is the (with probability 1, unique) minimum solution to the assignment problem with matrix $\hat{C}$, then $\pi = \phi\psi$ is the minimum solution to the original problem. We exploit the randomness of ϕ, which produces a random permutation π from ψ. Instead of taking π as given, we assume that ψ is given and π is to be obtained through a random permutation ϕ. We condition on the cycle structure of π. Defining k_i as the number of cycles of length i in π, we assume that (i) $\sum_{i=1}^{n_0} i k_i \le 2n_0$ and that (ii) $\sigma = \sum_{i=1}^{n} k_i \le 2 \ln n$; these conditions hold whp.

How do we sample a random permutation conditioned upon having a cycle structure dictated by $k_1, k_2, \ldots$, i.e., dictated by the multiset $\{k_i \times i : i \in [n]\}$ in which cycle length i appears k_i times? Let Π denote the set of permutations of X with the given cycle structure. Let γ be any fixed permutation with the given cycle structure. (For example, if $t_1 = 0$, $t_{\sigma+1} = n$, and the multisets $\{t_{j+1} - t_j : j \in [\sigma]\}$ and $\{k_i \times i : i \in [n]\}$ coincide, then we may define γ by the following: If $x, y \in C_j$ and $y = x + 1 \bmod t_{j+1} - t_j$, then $\gamma(x) = y$.) Then given a bijection $f : X \to X$ we define a permutation π_f on X by $\pi_f = f^{-1}\gamma f$. Each permutation $\pi \in \Pi$ appears precisely $\prod_{i=1}^{n} k_i! i^{k_i}$ times as π_f. Thus choosing a random mapping f chooses a random π_f from Π. (This is equivalent to randomly choosing $\phi = f^{-1}\gamma f \psi^{-1}$.)

The most natural way to look at this is to think of having oriented cycles on the plane whose vertices are at points $P_1, P_2, \ldots, P_n$ and then randomly labeling these points with X. Then if P' follows P on one of the cycles and P, P' are labeled x, x' by f, then $\pi_f(x) = x'$.

To give a concrete example, Figure 1 included a "canonical" digraph 4-cycle, labeled $1, 2, 3, 4$, arising from a corresponding canonically labeled structure in the bipartite graph, the matching edges (x_1, y_2), (x_2, y_3), (x_3, y_4), (y_4, x_1). In a random labeling dictated by a random permutation f, these matching edges would be labeled $(x_{f(1)}, y_{f(2)})$, $(x_{f(2)}, y_{f(3)})$, $(x_{f(3)}, y_{f(4)})$, $(x_{f(4)}, y_{f(1)})$, and the digraph's 4-cycle would be labeled $f(1)$, $f(2)$, $f(3)$, $f(4)$.

As we proceed through Phase 2 we have to expose parts of f (equivalently ϕ). x is *clean* if $f(x)$ is unexposed (the label x has not yet been used) and *dirty* otherwise. Thus imagine that we have cycles, mostly unlabeled, but with a few vertices labeled. Let us use $\tilde{}$ to denote a partially labeled graph.

We can now add the final layer to our description of how to eliminate the small cycles. We proceed in order through the selected edges $i \in [\lambda]$. At stage i we should have eliminated $C_1, C_2, \ldots, C_{j-1}$ for some j, and have a current perfect matching M_i, defining ρ_i. (Consider M_i to be fully revealed, but the labels on its vertices not revealed except for the selected edges in short cycles; thus all that is revealed of ρ_i is its cycle structure and labels on these few edges.)

We construct the trees S_i and T_i and then seek red or blue edges between the leaves $S_i^{(\tau)}$ of S_i and $T_i^{(\tau)}$ of T_i. We will consider only vertices of $S_i^{(t)}$ and $T_i^{(t)}$ that are clean and whose paths to their respective roots also contain only clean vertices; call these vertices *squeaky clean*. We take the squeaky clean vertices $v \in S_i^{(\tau)}$ in some fixed order. For each such vertex v we look, again in some fixed order, at each squeaky clean vertex $w \in T_i^{(\tau)}$. Each such edge (v, w) is either red or has length uniform in $[c_2 \frac{\ln n}{n}, 1]$. Thus the probability that it is red or blue is at least $c_2 \frac{\ln n}{n}$. This lower bound holds conditionally on the current history—see the comment at the beginning of section 3.1. We assert that

(13) whp there are $\geq n^{3/5}$ squeaky clean vertices in each of $S_i^{(\tau)}$ and $T_i^{(\tau)}$;

this will be shown in the last paragraph in this subsection. Thus if we run through $n^{2/5}$ squeaky clean vertices $v \in S_i^{(\tau)}$, the expected number of red or blue edges to squeaky clean vertices $w \in T_i^{(\tau)}$ is $\geq c_2 \frac{\ln n}{n} \cdot n^{2/5} \cdot n^{3/5}$, and with probability $\geq 1 - e^{-c_2 \ln n} = 1 - n^{-c_2}$, we find at least one red or blue edge. For a given v, if we find no red or blue edge to any w, we move on to the next v. If we find a red or blue edge (v, w), we test it for *acceptability* as described in the following paragraphs. If the edge is acceptable, the cycle can be repaired and we move on to the next cycle i. If the edge is not acceptable, v and w have been dirtied in the course of the testing, and we move on to the next v. Because we find a red or blue edge after exploring about $n^{2/5}$ vertices from $S_i^{(\tau)}$, and $|S_i^{(\tau)}| \geq n^{3/5}$, we can find at least $n^{1/5}$ red or blue edges to test; we will soon see that the failure probability is $\ll n^{-1/5}$, so eventual success is assured whp.

Now consider squeaky clean $\xi_i \in S_i^{(\tau)}, \eta_i \in T_i^{(\tau)}$ such that (ξ_i, η_i) is red or blue. In the bipartite graph, there is a squeaky clean alternating circuit $C = P[i, \xi_i]$, (ξ_i, η_i), $Q[i, \eta_i]$, (b_i, a_i). (C may start life with vertices crossed multiple times, but we can prune it down to a circuit containing (b_i, a_i).) We will define what it means for a cycle to be "acceptable" and show that C is acceptable with probability at least $(\ln n)^{-\alpha}$, where

$$\alpha = 7(\ln K)^{-1} < 1/2.$$

For any $x \in S_i^{(t)}$, consider $P[i, x] = (x_0 = a_i, y_1, x_1, y_2, \ldots, y_t, x_t = x)$, where $y_j = \rho_i(x_j)$ for $j \geq 1$. $P[i, x]$ defines a sequence $M^{(0)}, M^{(1)}, \ldots, M^{(t)}$ of near-perfect matchings (see Figure 2) $M^{(s)} = (M^{(s-1)} \cup \{(x_{s-1}, y_s)\}) \setminus \{(x_s, y_s)\}$. Let $\Gamma^{(0)}, \Gamma^{(1)}, \ldots, \Gamma^{(t)}$ be the associated NPDs. We say that $\Gamma^{(s)}$ is *acceptable* if (i) $|\mathrm{PATH}(\Gamma^{(s)})| \geq n_0$ and (ii) the small cycles of $\Gamma^{(s)}$ are a subset of $\{C_{i+1}, \ldots, C_\lambda\}$. We say that x is acceptable if $\Gamma^{(0)}, \Gamma^{(1)}, \ldots, \Gamma^{(t)}$ are all acceptable.

Going back to $P[i, x = x_t]$, let us estimate the probability that x_t is acceptable, given that it is clean and x_{t-1} is acceptable. Assume that we have revealed $f(x_{t-1})$ and that we have a partially labeled NPD $\tilde{\Gamma}_i^{(t-1)}$. We randomly choose $f(x_t)$ from the unlabeled points and label it with x_t. We then replace the arc $(f(x_t), \cdot)$ of $\tilde{\Gamma}_i^{(t-1)}$ with $(f(x_{t-1}), \cdot)$.[4] When $t = 1$, x_1 is acceptable unless $f(x_1)$ lies in a small cycle; it follows from (10) that given the previous exposures, this has probability at most $p_1 = 3 \frac{\ln \ln n}{\ln n}$. For $t > 1$, x_t will be acceptable if $f(x_t)$ is not within n_0 of an endpoint

[4]When dealing with the path from b_i to η_i we randomly choose $f(x_t)$ and then replace the arc $(f(x_{t-1}), \cdot)$ with $(f(x_t), \cdot)$.

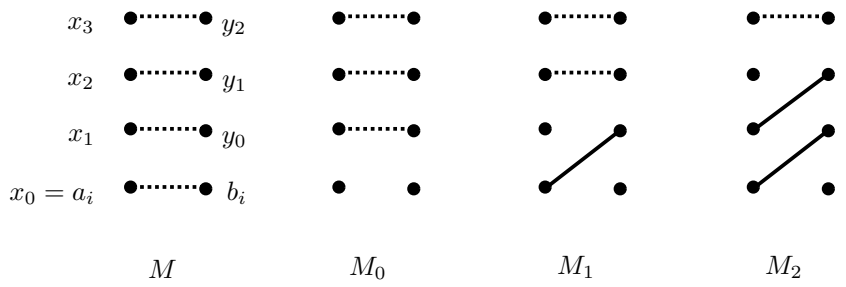


FIG. 2. *For $t = 3$, a perfect matching (dashed edges) and the sequence of near-perfect matchings (dashed and solid edges) defined by an alternating path $(x_0 = a_i, y_1, x_1, y_2, \ldots, x_t = x)$ (the union of all edges shown).*

of PATH($\tilde{\Gamma}_i^{(t-1)}$). We will see that

whp only $O((\ln n)^{2\alpha})$ squeaky clean alternating circuits need to be checked before an acceptable one is found.

Since each path has $O(\ln n)$ points, $|S_i^{(\tau)}|, |T_i^{(\tau)}| = O(n^{3/4})$ (12), and we repeat for $O(\ln n)$ cycles, in all, at most $\tilde{O}(n^{3/4})$ values of f are exposed.[5] So if x_t is clean, it will be unacceptable with probability at most $p_2 = \frac{2n_0}{n-\tilde{O}(n^{3/4})} \le \frac{3\ln\ln n}{\ln n}$ conditional on previous exposures. A similar analysis holds for the paths $Q[i, y]$.

If all vertices on C are clean, then the probability that C is not acceptable is at most $p_1 + 1 - (1-p_2)^{2\tau} \le \frac{3\ln\ln n}{\ln n} + 1 - \big(1 - \frac{3\ln\ln n}{\ln n}\big)^{2\tau} \le 1 - (\ln n)^{-\alpha}$. Thus if we can find $(\ln n)^{2\alpha}$ clean cycles, then one of them will succeed, with probability at least $1 - (1 - (\ln n)^{-\alpha})^{(\ln n)^{2\alpha}} \ge 1 - \exp\{-(\ln n)^{\alpha}\}$. As remarked earlier, we can find far more clean cycles than this—in fact around $n^{1/5}$ of them—as long as there are around $n^{3/5}$ squeaky clean vertices in each of $S_i^{(\tau)}$ and $T_i^{(\tau)}$; this is all that remains to be shown.

Let $A_i^{(t)}$ denote the squeaky clean vertices of $S_i^{(t)}$, $t = 1, 2, \ldots, \tau$. It follows from Lemmas 8 and 9 that $|A_i^{(1)}| \ge K \ln n - 4\lambda - (\ln n)^{2\alpha}$ (λ is the number of small cycles) and that $|A_i^{(t)}| \ge (K-4)|A_i^{(t-1)}| - 4\lambda - (\ln n)^{2\alpha}$ for $2 \le t \le \ln\ln n$. Here we use (11) to argue that for $i' < i$, the first $\ln\ln n$ levels of each $S_{i'}, T_{i'}$ dirty at most 2 vertices of the first $\ln\ln n$ levels of S_i, giving the 4λ term. The $(\ln n)^{2\alpha}$ term comes from vertices dirtied during failed acceptability tests for the current cycle i, one dirtied vertex per level t per failed test. The higher levels of $S_{i'}, T_{i'}$ do not dirty any of the lower levels of S_i, by construction. In general, for $t > \ln\ln n$, Lemma 9 implies that $|A_i^{(t+1)}| \ge (K-4)|A_i^{(t)}| - 4\lambda\tau(\ln n)^{2\alpha}$. Thus, $|A_i^{(\tau)}| \ge n^{3/5}$. A similar argument holds for squeaky clean vertices of $T_i^{(\tau)}$, verifying assertion (13).

3.3. Phase 3. For Phase 3 we use the green edges. We can assert that whp at the end of Phase 2, all cycles are of length at least n_0 and so there are $o(\ln n)$ cycles. Given two cycles C_1, C_2 each of length at least n_0, the probability that we cannot patch them together (delete edges (a_i, b_i) from C_i, $i = 1, 2$ and replace them with red or blue or green edges $(a_1, b_2), (a_2, b_1)$) is at most $(1 - \frac{c_2^2(\ln n)^2}{n^2})^{n_0^2} \le e^{-c_2^2(\ln\ln n)^2}$.

[5] Recall that $\tilde{O}$ suppresses $\ln n$ factors.

Doing this $o(\ln n)$ times increases the cost by at most $o(\frac{(\ln n)^2}{n})$ and so Phase 3 succeeds whp.

This completes the proof of the high-probability upper bound on ATSP − AP.

We now consider the lower bound of Theorem 1.

3.4. Proof of the lower bound. The AP can be expressed as a *linear program*:

(LP)

$$\text{Minimize } \sum_{i,j} C(i,j)z_{i,j} \text{ subject to } \sum_i z_{i,k} = \sum_j z_{k,j} = 1, \forall k, 0 \le z_{i,j} \le 1, \forall i,j.$$

This has the dual linear program:

$$\text{(DLP)} \qquad \text{Maximize } \sum_i u_i + \sum_j v_j \text{ subject to } u_i + v_j \le C(i,j),\ \forall i,j.$$

Remark 10 (condition on an optimal basis for (LP)). We may w.l.o.g. take $u_1 = 0$ in (DLP), whereupon with probability 1 the other dual variables are uniquely determined. Furthermore, the reduced costs of the *nonbasic* variables $\bar{C}(i,j) = C(i,j) - u_i - v_j$ are independently and uniformly distributed, with $\bar{C}(i,j) \in_{\text{unif}} [\max\{0, -u_i - v_j\}, 1 - u_i - v_j]$.[6]

Proof. The $2n-1$ dual variables are unique with probability 1 because they satisfy $2n-1$ linear equations. The only conditions on the nonbasic edge costs are that $C(i,j) \in [0,1]$ (equivalently $\bar{C}(i,j) \in [-u_i - v_j, 1 - u_i - v_j]$) and $\bar{C}(i,j) \ge 0$; intersecting these intervals yields the last claim. ∎

LEMMA 11. *Whp*

$$\max_{i,j} \{|u_i|, |v_j|\} \le c_5 \frac{\ln n}{n}. \tag{14}$$

Proof. Optimal dual values u_i, v_j can be characterized as shortest distances, as follows [1]. Consider a directed bipartite digraph Γ on $X \cup Y$ with "forward" edges (x_i, y_j), $i,j \in [n]$, $j \ne \pi(i)$, of length $C(i,j)$; and "backward" edges (y_j, x_i), $i,j \in [n]$, $j = \pi(i)$, of length $-C(i,\pi(i))$. If $u_1 = 0$, then $-u_i$ is the shortest distance $d(x_1, x_i)$ from x_1 to x_i in Γ, and v_j is the shortest distance from x_1 to y_j.[7]

Lemma 7 implies that $-u_i, v_i \le c_6 \frac{\ln n}{n}$ for $i \in [n]$. Furthermore, using the fact that a cheapest path is also a cheapest walk (derived from the optimal assignment, Γ has no negative-cost cycles), $-u_j = d(x_1, x_j) \le d(x_1, x_i) + d(x_i, x_j) \le -u_i + c_6 \frac{\ln n}{n}$ implies $u_i - u_j \le c_6 \frac{\ln n}{n}$. Immediately, $|u_i| \le c_6 \frac{\ln n}{n}$ and also, with $\bar{u} = \sum u_i/n$, $|\bar{u}| \le c_6 \frac{\ln n}{n}$. Likewise, $v_i - v_j \le c_6 \frac{\ln n}{n}$, from which $|v_i - \bar{v}| \le c_6 \frac{\ln n}{n}$. But we know that whp the optimal assignment cost satisfies $(1-\epsilon)\pi^2/6 < \sum_i u_i + \sum_j v_j < (1+\epsilon)\pi^2/6$ for any fixed positive ϵ [2, 16], so $\bar{v} \in (1.6/n - \bar{u},\ 1.7/n - \bar{u})$, giving $|\bar{v}| \le c_6 \frac{\ln n}{n} + O(1/n)$ and finally $|v_j| \le c_7 \frac{\ln n}{n}$. (While the tight assignment bounds from [2, 16] are elegant, for our purposes older bounds from [8, 13] would suffice.) ∎

[6] Do not be misled by the notation: $-u_i - v_j$ can be (and often is) positive.

[7] It is easy to see this from the graph with edge costs $\bar{C}(i,j) = C(i,j) - u_i - v_j \ge 0$. This graph includes a spanning tree of 0-cost edges, so all distances are 0. The C-cost of any path is almost the same as its $\bar{C}$-cost: Of the two directed edges leading into and out of any intermediate node, one has a u_i (or v_j) added, and the other has the same quantity subtracted. The cancellation fails only at the path's source (but we defined $u_1 = 0$) and at its terminal, resulting in C-distance $-u_i$ or $+v_j$ as claimed.

Having solved (LP) we will have n basic variables $z_{i,j}$, $(i,j) \in I_1$, with value 1 and $n-1$ basic variables $z_{i,j}$, $(i,j) \in I_2$, with value 0. The edges (x_i, y_j), $(i,j) \in I = I_1 \cup I_2$ form a tree T^* in $K_{X,Y}$. We show that with probability at least $c_9 > 0$ there exists $(i,i) \in I_1$ (a loop) such that (x_i, y_i) is a pendant edge in T^*; w.l.o.g. suppose x_i is its leaf. In this case the optimal TSP tour, viewed as a bipartite matching, cannot use the edge (x_i, y_i) (a loop), and must use some other edge $(x_i, y_{i'})$; since x_i is a leaf in T^*, $z_{i,i'}$ is not a basic LP variable. The expected value of the reduced cost of $z_{i,i'}$ is at least $\frac{c_{10}}{n}$, and so $\mathbf{E}(\mathrm{ATSP} - \mathrm{AP}) \geq \frac{c_9 c_{10}}{n}$ and the lower bound follows.

To prove the existence of (i,i) we show that whp the optimal assignment ψ for $\hat{C}$ of section 3 has at least $c_{11} n$ leaves L. After applying the random permutation ϕ, the number of leaves giving rise to loops is, at least, a random variable whose distribution is asymptotically Poisson with density c_{11}; thus

$$\mathbf{Pr}(\exists \text{ at least one leaf-loop}) \geq (1 - o(1))(1 - e^{-c_{11}}).$$

By taking a spanning tree T of $K_{X,Y}$ that contains a perfect matching M, and shrinking the edges of M, we obtain a tree isomorphic to a spanning tree T' of K_n. Each T arises from exactly 2^{n-1} trees T' because we have two choices as to how to configure each non-M edge. (An (i,j) edge in T' can in T be expanded to (x_i, y_j) or to (x_j, y_i).) Let $b(T) = b(T')$ denote the number of branching nodes (degree ≥ 3) of T and T'. A tree T' is ϵ-bushy if $b(T') \leq \epsilon n$. Bohman and Frieze used this concept in [6] and showed that the number of ϵ-bushy trees is at most $n! e^{\theta(\epsilon) n}$, where $\theta(\epsilon) \to 0$ as $\epsilon \to 0$. It follows that the number of ϵ-bushy trees of $K_{X,Y}$ which have a perfect matching is at most $e^{\theta(\epsilon) n} 2^{n-1} n!$. Observe that the number of leaves in T is at least $b(T)$. We complete the proof by showing that, for a sufficiently small constant ϵ,

$$\mathbf{Pr}(T^* \text{ is } \epsilon\text{-bushy}) = o(1). \tag{15}$$

For any tree T with a perfect matching, we can put $u_1 = 0$ and then solve the equations $u_i + v_j = C(i,j)$ for $(x_i, y_j) \in T$ to obtain the associated dual variables. T is optimal if $\bar{C}(i,j) = C(i,j) - u_i - v_j \geq 0$ for all $(x_i, y_j) \notin T$. Let $Z_T = \sum_i u_i + \sum_j v_j$. Now whp the optimal tree T^* satisfies $Z_{T^*} \in [1.6, 1.7]$, because Z_{T^*} is the optimal assignment cost, and it is known both that expectation is in the stated range [2] and that the actual value is concentrated about the expectation [16]. Then if $\mathcal{E}$ denotes the event $\{(14) \text{ and } Z_T \in [1.6, 1.7]\}$, for any tree T, over random matrices $C(i,j)$,

$$\begin{aligned}
&\mathbf{Pr}(Z_T \in [1.6, 1.7] \text{ and } (14) \text{ and } \bar{C}(i,j) \geq 0 \ \forall (i,j) \notin I) \\
&\quad \leq \mathbf{Pr}(\bar{C}(i,j) \geq 0 \ \forall (i,j) \notin T \mid \mathcal{E}) \times \mathbf{Pr}(Z_T \in [1.6, 1.7]) \\
&\quad \leq \frac{1.7^n}{n!} \mathbf{E}\left(\prod_{(x_i,y_j) \notin T} (1 - (u_i + v_j)^+) \mid \mathcal{E} \right) \\
&\quad \leq \mathbf{E}\left(\exp\left\{ - \sum_{(x_i,y_j) \notin T} (u_i + v_j) \right\} \mid \mathcal{E} \right) \frac{1.7^n}{n!} \\
&\quad \leq \mathbf{E}\left(e^{-n Z_T} \exp\left\{ \sum_{(x_i,y_j) \in T} (u_i + v_j) \right\} \mid \mathcal{E} \right) \frac{1.7^n}{n!} \\
&\quad \leq e^{-1.6n} n^{2c_5} \frac{1.7^n}{n!}.
\end{aligned}$$

Explanation. $\frac{1.7^n}{n!}$ bounds the probability that the sum of the lengths of the edges in the perfect matching of T is at most 1.7. The product term is the probability that each nonbasic reduced cost is nonnegative.

Thus

$$\begin{aligned}&\mathbf{Pr}(\exists \text{ an } \epsilon\text{-bushy tree } T : Z_T \in [1.6, 1.7] \text{ and (14) and } \bar{C}(i,j) \geq 0 \ \forall (i,j) \notin I)\\ &\quad \leq n! 2^n e^{\theta(\epsilon)n} \times e^{-1.6n} n^{2c_5} \frac{1.7^n}{n!}\\ &\quad = o(1)\end{aligned}$$

for ϵ sufficiently small. This implies (15). □

Remark 12. The above proof of Theorem 1's lower bound relies on there being a positive probability that the optimal AP solution includes loops. However, loops (unlike other cycles) can easily be excluded from an AP solution by changing the weight matrix C to give weight 1 to each loop. Since this potentially increases the AP cost but leaves the ATSP cost unchanged, it would be interesting to prove the Theorem 1 lower bound for random cost matrices C with diagonal entries 1.

4. An enumerative algorithm. We can now prove Theorem 4, restated here for convenience.

THEOREM 4. *Whp, a random instance of the* ATSP *can be solved exactly in time* $e^{\tilde{O}(\sqrt{n})}$.

Proof. Let I_k denote the interval $[2^{-k} c_1 \frac{(\ln n)^2}{n},\ 2^{-(k-1)} c_1 \frac{(\ln n)^2}{n}]$ for $k \geq 1$. It follows from Remark 10 and Lemma 11 that whp (i) there are $\leq c_1 2^{-(k-1)} n \ln n$ nonbasic variables $z_{i,j}$ whose reduced cost is in $I_k, 1 \leq k \leq k_0 = \frac{1}{2}\log_2 n$, and (ii) there are $\leq 2c_1\sqrt{n}\ln n$ nonbasic variables $z_{i,j}$ whose reduced cost is $\leq c_1 \frac{(\ln n)^2}{n^{3/2}}$.

We can search for an optimal solution to the ATSP by choosing a set of nonbasic variables, setting them to 1 and then re-solving the AP. If we try all sets and choose the best tour we find, then we will clearly solve the problem exactly. However, it follows from Theorem 1 that whp we need only consider sets which contain $\leq 2^k$ variables with reduced costs in I_k and none with reduced cost $\geq c_1 \frac{(\ln n)^2}{n}$. Thus whp we need only check at most

$$\begin{aligned}2^{2c_1\sqrt{n}\ln n} \prod_{k=1}^{k_0} \sum_{t=1}^{2^k} \binom{c_1 2^{-(k-1)} n \ln n}{t} &\leq 2^{2c_1\sqrt{n}\ln n} \prod_{k=1}^{k_0} (2(c_1 2^{-(k-1)} n \ln n)^{2^k})\\ &\leq 2^{2c_1\sqrt{n}\ln n} (2(c_1 2^{-(k_0-1)} n \ln n)^{2^{k_0}})^{k_0}\\ &= e^{\tilde{O}(\sqrt{n})}\end{aligned}$$

sets, using the fact that $(A/x)^x$ is monotone increasing for $x \leq A/e$. □

5. Second-best assignment. We recall and prove Theorem 3, on the gap Δ_1 between the costs of the cheapest and second-cheapest assignments.

THEOREM 3. *Over random cost matrices* C,

$$\frac{1}{n^2}(1 - o(1)) \leq \mathbf{E}(\Delta_1) \leq c_3 \frac{\ln n}{n^2}.$$

Proof. Δ_1 is equal to the minimum nonbasic reduced cost.[8] From Lemma 11 and $\sum_i u_i + \sum_j v_j > 1.6$ whp, it follows that whp there are at least $n_1 = c_7 \frac{n^2}{\ln n}$ pairs i, j such that $u_i + v_j > 0$. Each such pair corresponds to a nonbasic variable $\bar{C}(i, j)$, and it follows from Remark 10 that the minimum reduced cost among this set is at most $\frac{1}{n_1+1}$ in expectation, proving the upper bound.

For the lower bound, again from Remark 10, the $n^2 - 2n + 1$ nonbasic reduced costs $\bar{C}(i, j)$ are independent, with $\bar{C}(i, j) \in_{\text{unif}} [a_{i,j}, b_{i,j}]$, where each $a_{i,j} \geq 0$ and (from Lemma 11) each $b_{i,j} \geq 1 - 2c_5 \ln n/n$. The minimum of this collection satisfies $\mathbf{E}(\min\{\bar{C}(i, j)\}) \geq \frac{1}{n^2-2n}(1 - 2c_5 \ln n/n) = \frac{1}{n^2}(1 - o(1))$. □

REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows Theory, Algorithms and Applications*, Prentice–Hall, Englewood Cliffs, NJ, 1993.

[2] D. Aldous, *Asymptotics in the random assignment problem*, Probab. Theory Related Fields, 93 (1992), pp. 507–534.

[3] R. Arratia, A. D. Barbour, and S. Tavaré, *Poisson process approximations for the Ewens sampling formula*, Ann. Appl. Probab., 2 (1992), pp. 519–535.

[4] R. Arratia and S. Tavaré, *The cycle structure of random permutations*, Ann. Probab., 20 (1992), pp. 1567–1591.

[5] E. Balas and P. Toth, *Branch and bound methods*, in The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, E. L. Lawler, J. K. Lenstra, A. H. Rinnooy Kan, and D. B. Shmoys, eds., Wiley, New York, 1986, pp. 361–400.

[6] T. Bohman and A. M. Frieze, *Avoiding a giant component*, Random Structures Algorithms, 19 (2001), pp. 75–85.

[7] M. E. Dyer and A. M. Frieze, *On patching algorithms for random asymmetric travelling salesman problems*, Math. Programming, 46 (1990), pp. 361–378.

[8] M. E. Dyer, A. M. Frieze, and C. McDiarmid, *On linear programs with random costs*, Math. Programming, 35 (1986), pp. 3–16.

[9] P. Erdős and A. Rényi, *On random matrices*, Publ. Math. Inst. Hungar. Acad. Sci., 8 (1964), pp. 455–461.

[10] A. M. Frieze and G. R. Grimmett, *The shortest path problem for graphs with random arc-lengths*, Discrete Appl. Math., 10 (1985), pp. 57–77.

[11] A. Frieze, R. M. Karp, and B. Reed, *When is the assignment bound tight for the asymmetric traveling-salesman problem?*, SIAM J. Comput., 24 (1995), pp. 484–493.

[12] M. Held and R. M. Karp, *A dynamic programming approach to sequencing problems*, SIAM J. Appl. Math., 10 (1962), pp. 196–210.

[13] R. M. Karp, *An upper bound on the expected cost of an optimal assignment*, in Discrete Algorithms and Complexity: Proceedings of the Japan–U.S. Joint Seminar, D. Johnson et al., eds., Academic Press, New York, 1987, pp. 1–4.

[14] R. M. Karp, *A patching algorithm for the nonsymmetric traveling salesman problem*, SIAM J. Comput., 8 (1979), pp. 561–573.

[15] R. M. Karp and J. M. Steele, *Probabilistic analysis of heuristics*, in The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, eds., Wiley, New York, 1985, pp. 181–206.

[16] M. Talagrand, *Concentration of measure and isoperimetric inequalities in product spaces*, Inst. Hautes Études Sci. Publ. Math., 81 (1995), pp. 73–205.

[8] From linear programming, Δ_1 is at least the minimum nonbasic reduced cost. Also, Δ_1 is no more than this: For the assignment problem, edges corresponding to basic variables form a tree. Adding any nonbasic edge creates an alternating cycle, whose symmetric difference with the optimal matching gives a second matching. The cost increase is the cost of the cycle, which is the sum of its (signed) edge costs. The sum of the signed costs of the edges around a cycle is equal to the same sum of the reduced costs, because each u_i, for example, is added twice, with opposite signs, to the two edges incident on x_i. The cycle in question contains only a single nonbasic edge, so the sum of its reduced edge costs is just the cost of this edge.

SIAM J. COMPUT.
Vol. 36, No. 5, pp. 1453–1471

THE WAKE-UP PROBLEM IN MULTIHOP RADIO NETWORKS*

MAREK CHROBAK†, LESZEK GĄSIENIEC‡, AND DARIUSZ R. KOWALSKI‡

Abstract. We study the problem of waking up a collection of n processors connected by a multihop ad hoc ratio network with unknown topology, no access to a global clock, and no collision detection mechanism available. Each node in the network either wakes up spontaneously or gets activated by receiving a wake-up signal from another node. All active nodes transmit the wake-up signals according to a given protocol $\mathcal{W}$. The running time of $\mathcal{W}$ is the number of steps counted from the first spontaneous wake-up until all nodes become activated. We provide two protocols for this problem. The first one is a deterministic protocol with running time $O(n^{5/3} \log n)$. Our protocol is based on a novel concept of a shift-tolerant selector to which we refer as a *(radio) synchronizer*. The second protocol is randomized, and its expected running time is $O(D \log^2 n)$, where D is the diameter of the network. Subsequently we show how to employ our wake-up protocols to solve two other communication primitives: leader election and clock synchronization.

Key words. radio network, wake-up, gossiping, broadcasting, probabilistic method

AMS subject classifications. 68W15, 68Q25, 68505, 94A99

DOI. 10.1137/S0097539704442726

1. Introduction.

We define an ad hoc multihop radio network as a directed strongly connected graph with n nodes (also referred to as vertices or processors). The nodes of this graph represent computing devices that can transmit and receive radio signals. The directed edges represent transmission ranges of the nodes: if a signal transmitted from u can be heard by a node v, then G contains the corresponding edge (u, v). The execution time is divided into discrete time steps, the same for all processors. We assume that each processor is assigned a unique integer label (or identifier) from a range of size $O(n)$. The processors know the range of the labels, but they do not know the network's topology.

We consider the fundamental problem of waking up the processors of an ad hoc multihop radio network. Initially, all nodes are assumed to be asleep. Each node in the network can wake up spontaneously, or it can be activated by receiving a wake-up signal from another node. Once a node v is activated, it starts executing its wake-up protocol $\mathcal{W}$. This protocol dictates during which time steps v will transmit a wake-up signal. In our model, the network does not have a global clock, so $\mathcal{W}$ depends only on the local clock of v (the number of steps since activation) and possibly additional information received earlier from other nodes. The wake-up signal from v is sent, during the same time step, to all out-neighbors of v. However, an out-neighbor u of v receives this signal only if no collision occurred, that is, if no other in-neighbor of u transmitted at the same time. Furthermore, we do not assume any collision detection

*Received by the editors April 6, 2004; accepted for publication (in revised form) September 5, 2006; published electronically January 26, 2007. A preliminary version of this paper appeared in the Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA'04), New Orleans, LA, 2004, SIAM, Philadelphia, pages 985–993.

http://www.siam.org/journals/sicomp/36-5/44272.html

†Department of Computer Science, University of California, Riverside, CA 92521 (marek@cs.ucr.edu). The first author's research was supported by NSF grants CCR-9988360 and CCR-0208856.

‡Department of Computer Science, University of Liverpool, Liverpool L69 3BX, UK (leszek@csc.liv.ac.uk, darek@csu.liv.ac.uk). The second author's research was supported by the EPSRC grant GR/R84917/01. Part of the third author's work was done when he was a postdoctoral researcher in Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 6123 Saarbruecken, Germany.

capability, so if u does not receive a signal at time t, it has no way to decide whether a collision occurred or none of its neighbors transmitted. The running time of $\mathcal{W}$ is the number of time steps, counting from the first spontaneous wake-up until all nodes become activated.

While other important communication primitives, such as *broadcasting* (one-to-all communication), *gossiping* (all-to-all communication), and *multicast* (some-to-some communication) have been explored extensively in the context of ad hoc multihop radio networks (e.g., see [5, 7, 8, 9, 10, 12, 14, 15, 16, 17, 21, 24, 25, 26, 30]), the wake-up and synchronization problems were studied primarily for *one-hop* networks (i.e., in the presence of a complete graph of connections). For one-hop networks, Gąsieniec, Pelc, and Peleg [20] presented a deterministic protocol with running time $O(n^2 \log n)$. They also gave a probabilistic argument showing that there exists a wake-up protocol with running time $O(n \log^2 n)$. Later, Indyk [21] showed that the same proof can be turned into a constructive argument, with only a slight increase in the running time. In the randomized case, in [20] one can find an algorithm that, for any given $\epsilon > 0$, completes the wake-up process in time $O(n \log(1/\epsilon))$ with probability at least $1 - \epsilon$. This result was subsequently improved by Jurdziński and Stachowiak [23], who gave a protocol with running time $O(\log n \log(1/\epsilon))$.

Note that broadcasting can be reduced to wake-up, in the following sense. Given a wake-up protocol $\mathcal{W}$, we can use it to perform broadcasting by waking up only the node s that wants to initiate the broadcast, and then using the transmissions of $\mathcal{W}$ to transmit the broadcast message instead of the wake-up signal. Therefore the wake-up problem can be thought of as a generalization of broadcasting.

Our results. We study the wake-up problem in ad hoc multihop radio networks when the network has an arbitrary topology—a more general setting than the one considered in [20, 23]. Further, we assume that this topology is not known to the algorithm. We provide two protocols for this problem.

First, we give a probabilistic construction of a deterministic wake-up protocol with running time $O(n^{5/3} \log n)$. Our construction works in several stages. In the first stage, we construct certain combinatorial structures called *(radio) synchronizers*. Later, we take a random protocol $\mathcal{P}$ that consists of a synchronizer followed by a random sequence of transmissions, where each transmission is executed with probability $1/n$. We prove that, with very high probability, this random protocol completes the wake-up task in time $O(n^{5/3} \log n)$ in so-called *path graphs*. This implies, via the probabilistic method, that there exists a deterministic protocol $\mathcal{D}$ for path graphs with the same running time. Finally, we prove that $\mathcal{D}$ works, in fact, for arbitrary strongly connected graphs.

As our second result, we give a randomized wake-up protocol $\mathcal{R}$ whose expected running time is $O(D \log^2 n)$, where D is the diameter of the network. (Unlike in the deterministic case, this randomized protocol does not use node labels.)

We subsequently study two other communication primitives: leader election and clock synchronization. In the *leader election* problem, we want to designate one node, typically the one with minimum label, as the leader. All nodes have to be informed about the leader identity (label). In the *clock synchronization* problem, we require that, after the completion of the protocol, all nodes agree on some integer value (typically, the local time of one selected node) as the current global time. Assuming that all node labels are integers drawn from an $O(n)$-size range, we show that both tasks, leader election and clock synchronization, can be achieved deterministically in time $O(n^{5/3} \log^2 n)$, and in expected time $O(D \log^3 n)$ with randomization.

Other related work. Recall that in our model all nodes are identified by unique labels of size $O(n)$. A comprehensive discussion of the wake-up problem in one-hop networks with unbounded labels can be found in [20, 23].

Zheng, Hou, and Sha [31] study a similar wake-up problem in ad hoc symmetric wireless networks, with the main focus on energy efficiency. They also provide an extensive discussion of existing wake-up mechanisms and related practical and theoretical issues.

The problem of waking up a collection of processing units is not specific to radio networks. In fact, one can formulate analogous problems for other distributed computing models. The general idea is that we have a number of agents that are initially in arbitrary unpredictable states (including clock values), and the goal is to achieve some degree of synchronization sufficient to perform communication or parallel computation, while minimizing resources like time, memory, etc. One such variant of the wake-up problem in the asynchronous shared memory model in the presence of faults was introduced by Fischer et al. [19]. On the more practical side, challenges similar to those in the wake-up problem arise, for example, after crashes or malicious attacks in distributed systems.

An unrelated, although similar in spirit, offline problem called the *freeze-tag problem* was studied by Arkin et al. [2]. Here, the goal is to design an optimal wake-up schedule for a swarm of robots.

The leader election problem has been, of course, widely studied in the area of distributed computing; see, e.g., [28]. In the context of radio networks, leader election has been studied mainly for one-hop networks. For example, Jurdziński, Kutyłowski, and Zatopiański [22] investigated the energy-efficient randomized algorithms for this problem. (See [22] for more references to work on this case.) To our knowledge, the only work on leader election in multihop radio networks is that by Bar-Yehuda, Goldreich, and Itai [6], who proposed an efficient algorithm for undirected networks.

Finally, Awerbuch et al. [4] considered synchronization-like problems in an asynchronous message-passing distributed system. We also point out that the term "synchronizer" was used in other contexts in distributed computing (see [3]). However, our definition of synchronizers is different than those used in the setting of asynchronous distributed message-passing systems.

Comments about the model. Concluding the introduction, we offer a few general comments about our model in relation to past and contemporary network technologies.

In addition to the initial lack of synchronization, discussed above, any wake-up protocol in our model needs to overcome three other obstacles: lack of information about the topology, unidirectionality of links, and channel contention (transmission collisions.)

In today's wireless ad hoc networks, it would be unrealistic to assume that the nodes have accurate information about the network's topology. Our model of a completely unknown topology, although rather extreme in this respect, allows us to focus on the fundamental principles of communication in such networks, and, to our knowledge, it has been studied only from the theoretical perspective.

Networks with shared communication channels have been widely used since 1970s, starting with Abramson's pioneering work [1] on radio-based Aloha. (The version called Slotted Aloha allows communication to take place only at discrete time steps, thus bearing some resemblance to our model.) The ideas behind Aloha were adapted to Ethernet local area networks [29], in which communication takes place over a

common bus (wire). The use of shared channels is, in fact, still common in today's wireless networks.

Reliable communications over a random-access shared channel typically employ some control mechanism that detects collisions and subsequently retransmits the lost messages. Unlike in our model, protocols used in practice are based on symmetric connections, where collision detection can be implemented through a feedback mechanism that allows the sending node to detect whether its transmission was successful. (In Aloha, this is accomplished with message acknowledgements and a timeout mechanism, while in Ethernet by sensing the signal level on the shared channel.) To deal with potential instability due to retransmission of lost messages, appropriate backoff protocols are employed. The exponential backoff mechanism used in Ethernet decreases the frequency of retransmissions by doubling the expected waiting time after each collision. (The backoff mechanism was not specified in Aloha.) We find it somewhat intriguing that our randomized protocol in section 5 does exactly the opposite, as it exponentially *increases* the transmission frequency over time.

2. Radio networks. A *radio network* is modeled as a directed strongly connected graph G with n *nodes* (also called *vertices* or *processors*). The nodes of G represent transmitting/receiving devices, and directed edges represent their ranges: if a node v is within the range of a node u, then G contains an edge (u, v).

If there is an edge from u to v, then we say that v is an *out-neighbor* of u and that u is an *in-neighbor* of v. The set of all in-neighbors of v is denoted by $N_G(v)$, or simply $N(v)$ if G is understood from context.

To simplify presentation, throughout the paper we assume, without loss of generality, that n is a power of 2. Further, we often use other functions of n, say $n^{1/3}$, $n^{5/3}$, etc., as if their values were integer. All the calculations can be easily formalized by using the appropriately rounded values instead, say $\lfloor n^{1/3} \rfloor$, $\lceil n^{1/3} \rceil$, etc., although we choose not to do it in order to avoid clutter.

Each node v is assigned a unique *label* ℓ_v, also called an *identifier*. The labels are integer numbers from an interval $\{\ell_{\min}, \ldots, \ell_{\max}\}$, where $\ell_{\max} - \ell_{\min} = O(n)$. Initially, each node knows only its label and the bounds $\ell_{\min}$, $\ell_{\max}$. In the paper we assume that $\ell_{\min} = 0$, since otherwise a node v instead of label ℓ_v can use an auxiliary label $\ell_v - \ell_{\min}$ for execution. In the construction of synchronizers and wake-up protocols, we further simplify the notation by assuming, without loss of generality, that the label set is $V = \{0, 1, \ldots, n-1\}$, so that we can identify nodes by consecutive integers. (Note that we cannot make this assumption in the leader election and synchronization protocols, because then node 0 could be always selected as the leader, making both problems trivial.)

The execution time is divided into discrete time steps. For the sake of clarity, we make a distinction between the terms "time" and "time step." A *time* t is the point t on the time axis. A *time step* t is the unit time interval $(t, t+1]$. Any actions (activations, transmissions, etc.) take place during time steps. Properties of the network (for example, the set of active nodes) are observed at time points. If some action takes place at time step t, its effect will be reflected by the change of the state of the network from time t to $t+1$.

Since our focus is on communication, we assume that processors have unlimited computing power and can perform arbitrary computations within one time step. However, only one transmission or message receipt is allowed in one time step. Each processor has its own local clock whose initial value, at the time of activation, is 0. All local clocks run at the same speed.

By a *wake-up schedule* we mean any vector $\omega = (\omega_x)_{x \in V}$, where ω_x denotes the time step in which x wakes up spontaneously. For any set $X \subseteq V$, by ω_X we denote the earliest wake-up time step in X; that is, $\omega_X = \min_{x \in X} \omega_x$. For our convenience, we will be assuming that $\omega_V = \min_{x \in V} \omega_x = 0$. A node v with $\omega_v = 0$ is referred to as a *start node*. Note that there may be several start nodes present. A node x is called *active* at time t if it either awakened spontaneously or was activated by another active node u by receiving its wake-up signal in some time step $t' \leq t - 1$.

A *wake-up protocol* $\mathcal{W}$ is a function that, for each label ℓ and for each $\tau = 1, 2, 3, \ldots$, given all past messages received by the node v with label $\ell_v = \ell$, specifies whether v will transmit the wake-up signal in time step τ (since its activation.)

A message M transmitted in step t from a node v is sent instantly to all its out-neighbors. However, an out-neighbor u of v receives M in time step t only if no collision occurred, i.e., if the other in-neighbors of u do not transmit in step t at all. Further, collisions cannot be distinguished from the background noise. If u does not receive any message in step t, it knows that either none of its in-neighbors transmitted in step t, or that at least two did, but it does not know which of these two events occurred.

Given a network G and a wake-up schedule ω, by a *wake-up network* we will mean the pair $\langle G, \omega \rangle$. The wake-up network, together with a (deterministic) wake-up protocol $\mathcal{W}$, fully determines the state of the network at each time. Let $acttime_v(\mathcal{W}, G, \omega)$ denote the activation time step of node v, that is, either ω_v or the time step when v received a wake-up message, whichever comes first. Note that, by the definition of active nodes, v is active for the first time at time $acttime_v(\mathcal{W}, G, \omega)+1$. By $ActSet_t(\mathcal{W}, G, \omega)$ we denote the set of all active nodes at time t. We will often simplify this notation and omit the arguments that are well understood from context, writing $acttime_v(\omega)$, $ActSet_t$, etc.

The running time of a wake-up protocol $\mathcal{W}$ is the smallest T such that, for any wake-up network $\langle G, \omega \rangle$ with vertex set V, all nodes are activated by time T; that is, $ActSet_T(\mathcal{W}, G, \omega) = V$. Note that according to this definition some nodes may continue transmissions after time T. However, without loss of generality we can assume that each node stops transmitting after T steps from its activation (any further transmissions are redundant.) Then the total termination time—from the first wake-up until all nodes complete their transmissions—is at most $2T$.

Protocols for leader election and synchronization, as well as their running times, are defined in an analogous manner. We remark here, however, that in our wake-up and leader election protocols, nodes transmit only wake-up signals to their neighbors; no other messages are used. For clock synchronization, messages include numerical values representing the global time.

3. Synchronizers. The notion of *selectors* was introduced in the context of time efficient distributed communication protocols in synchronized radio networks [14], and later studied by DeBonis, Gąsieniec, and Vaccaro [18] as a tool in weakly adaptive combinatorial group testing. In particular, an (n, k, m)-selector $\mathcal{T}$ is defined in [18] as follows: $\mathcal{T} = \{\mathcal{T}^x\}_{x \in V}$, where each $\mathcal{T}^x$ is a 0-1 sequence of length m, and for each subset $X \subseteq V$ of size k there exists a position $1 \leq t \leq m$ such that exactly one sequence $\mathcal{T}^x$ has 1 on position t. In radio network applications, $\mathcal{T}$ represents a transmission protocol for the network, where a node x transmits at time t if the tth bit of $\mathcal{T}^x$ is 1. The main intuition behind the condition on $\mathcal{T}$ is that if a set $X = N(v)$ of k nodes follows this protocol, then within at most m steps a successful (that is, collision-free) transmission from X to v will occur.

The selectors as defined in [14] are not sufficient for the wake-up problem, due to the lack of clock synchronization. Intuitively, as the nodes wake up at different times and start transmitting, their transmission sequences may be shifted with respect to each other. To deal with this difficulty, we now introduce a new type of selector-like structures called *radio synchronizers* (or *synchronizers*, for short), that tolerate arbitrary shifts.

Formally, let $\mathcal{S} = \{\mathcal{S}^x\}_{x\in V}$, where each $\mathcal{S}^x = S_1^x S_2^x \dots S_m^x$ is a 0-1 sequence of length m. We say that $\mathcal{S}$ is a (n,k,m)-*synchronizer* if it satisfies the following property:

($*$) For any nonempty set $X \subseteq V$ of cardinality at most k, and for any wake-up schedule ω, there exists t, where $\omega_X < t \le \omega_X + m$, such that

$$\sum_{x\in X} S^x_{t-\omega_x} = 1.$$

We assume here that $S_i^x = 0$, for $i \le 0$.

As explained earlier, we will interpret $\mathcal{S}$ as a transmission protocol, where $S_i^x = 1$ indicates that node x transmits in step $\omega_x + i$. Thus the condition ($*$) states that, in at most m steps after the first node in X wakes-up, there will be a time step when exactly one node in X transmits.

LEMMA 1. *Let $C \ge 31$ be an integer constant. For each n and $k \le n$, there exists an (n,k,m)-synchronizer with $m = Ck^2 \log n$.*

Proof. We use here a probabilistic argument. Without loss of generality, we can assume that $n \ge k \ge 2$. For all $x \in V$ and $i = 1, \dots, m$, independently, assign $S_i^x = 1$ with probability $1/k$, and 0 otherwise. We show that, with very high probability, $\mathcal{S} = \{\mathcal{S}^x\}_{x\in V}$ is an (n,k,m)-synchronizer.

Initially, we fix a set of nodes X with $1 \le |X| \le k$ and a wake-up schedule ω. Let $z \in X$ be the node with the earliest wake-up time in X, which is $\omega_z = \omega_X$. Fix a time $t \in (\omega_X, \omega_X + m]$, and let $X' = \{x \in X : \omega_x < t\}$ be the set of nodes in X that are "awakened" at time t. The probability that $S^x_{t-\omega_x} = 1$ for exactly one $x \in X$ is not smaller than the probability that z is the only one with $S^z_{t-\omega_z} = 1$. Using this inequality, the independence of the random variables S_i^x, for $x \in X$, and some routine calculations, we get

$$\begin{aligned}
\Pr\left[\sum_{x\in X} S^x_{t-\omega_x} = 1\right] &\ge \Pr\left[S^z_{t-\omega_z} = 1 \;\&\; \sum_{x\in X'-\{z\}} S^x_{t-\omega_x} = 0\right] \\
&= \frac{1}{k}\left(1-\frac{1}{k}\right)^{|X'|-1} \\
&\ge \frac{1}{k-1}\left(1-\frac{1}{k}\right)^{k} \\
&\ge \frac{1}{4k},
\end{aligned}$$

where the last inequality holds because $(1-1/k)^k \ge \frac{1}{4}$ for $k \ge 2$.

For different times t, the events "$\sum_{x\in X} S^x_{t-\omega_x} \ne 1$" are independent. Therefore, using the bound above, we get that the probability that no such time step exists is

$$\begin{aligned}
\Pr\left[\sum_{x\in X} S^x_{t-\omega_x} \ne 1 \;\; \forall t \in (\omega_X, \omega_X + m]\right] &\le \left(1-\frac{1}{4k}\right)^m \\
&\le \beta^{m/k}
\end{aligned}$$

for $\beta = e^{-1/4} < 1$.

The probability that $\mathcal{S}$ violates the definition of the (n,k,m)-synchronizer for a fixed X can be estimated by multiplying the above probability by the number of choices of ω. It is sufficient to count only those wake-up schedules ω that satisfy $\omega_X \le \omega_x \le \omega_X + m$ for $x \in X$, and $\omega_x = \omega_X$ for $x \notin X$. This is because, given an arbitrary ω, we can define another wake-up schedule ω' as $\omega'_x = \min\{\omega_x, \omega_X + m\}$ for $x \in X$, and $\omega'_x = \omega_X$ for $x \notin X$. Then $\mathcal{S}$ satisfies $(*)$ for X and ω iff $\mathcal{S}$ satisfies $(*)$ for X and ω'. The number of such relevant wake-up schedules is not greater than $(m+1)^{|X|} \le (2m)^k$, so the probability that $\mathcal{S}$ violates $(*)$ for X is at most $(2m)^k \beta^{m/k}$.

Now, to estimate the probability that $\mathcal{S}$ is not an (n,k,m)-synchronizer, we can multiply the above probability by the number of ways in which X can be chosen, which is at most $\sum_{j=1}^{k} \binom{n}{j} \le \sum_{j=1}^{k} n^j \le n^{2k}$. Thus

$$\begin{aligned}\Pr[\,\mathcal{S} \text{ is not a } (n,k,m)\text{-synchronizer}\,] &\le n^{2k}(2m)^k \beta^{m/k} \\ &\le n^{2k}(2Ck^2 \log n)^k \beta^{Ck \log n} \\ &\le \left[n^{6+\log C + C \log \beta}\right]^k \\ &< 1,\end{aligned}$$

as long as $C \ge 31$. Since this probability is smaller than 1, there exists an (n,k,m)-synchronizer $\mathcal{S}$ with $m = Ck^2 \log n$. ∎

4. A deterministic wake-up protocol. In this section we propose a wake-up protocol based on synchronizers and random transmission patterns. We also show later (using the probabilistic method) the existence of a deterministic protocol that completes the wake-up task in n-vertex directed graphs in time $O(n^{5/3} \log n)$. We proceed in four steps:

(1) We define a random protocol $\mathcal{P}$ and a class of graphs that we call *path graphs*. We also define a slightly different communication model for those graphs, which we call the *restricted model*.

(2) We show that, with probability exponentially close to 1, $\mathcal{P}$ will complete the wake-up task in time $O(n^{5/3} \log n)$, according to the restricted model, for an arbitrary n-vertex path graph and a wake-up schedule.

(3) Using the bound on the failure probability in (2), we show that there exists a *deterministic* protocol $\mathcal{D}$ for path graphs with running time $O(n^{5/3} \log n)$, in the restricted model.

(4) Finally, we prove that (3) implies that our protocol $\mathcal{D}$ works in fact also in arbitrary directed graphs, with running time $O(n^{5/3} \log n)$.

We point out here that applying the probabilistic method to path graphs, rather than directly to arbitrary graphs, is crucial, since the number of path graphs is substantially smaller than the number of arbitrary directed graphs—roughly n^n instead of $2^{n(n-1)}$.

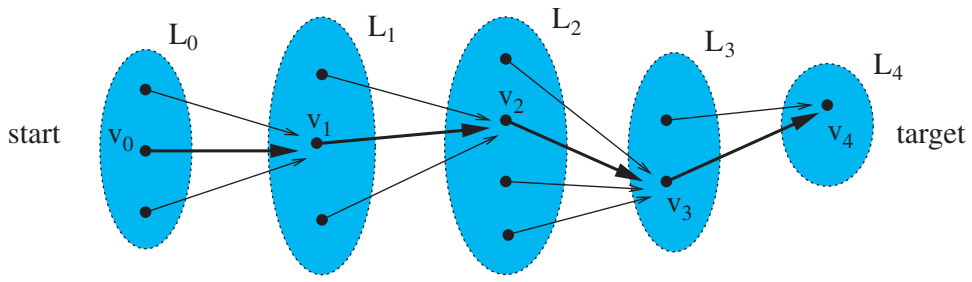
Random protocol. We now define our *random protocol* $\mathcal{P}$ (that is, a probability distribution on deterministic protocols.) Let $\mathcal{S} = \{\mathcal{S}^x\}_{x \in V}$ be an (n,k,m)-synchronizer with $k = n^{1/3}$ and $m = Cn^{2/3} \log n$, where $C = 31$. The existence of $\mathcal{S}$ has been established in Lemma 1. For each $x \in V$, we define a random 0-1 sequence $\mathcal{P}^x$ as

$$\mathcal{P}^x = \mathcal{S}^x \mathcal{U}^x,$$

where $\mathcal{U}^x$ is a random 0-1 sequence of length $O(n^{5/3} \log n)$ in which each bit is chosen, independently, to be 1 with probability $1/n$ and 0 with probability $1 - 1/n$. (The exact length of $\mathcal{U}^x$ will be determined later in this section.) Then $\mathcal{P} = \{\mathcal{P}^x\}_{x \in V}$ is a random

protocol, where the 1's in $\mathcal{P}^x$ indicate the times when node x transmits the wake-up signal. We refer to the sequence $\mathcal{S}^x$ as the *S-stage* and $\mathcal{U}^x$ as the *U-stage* of $\mathcal{P}^x$. We emphasize that for any given node x, its S-stage transmissions are fixed; only the transmissions in its U-stage vary randomly.

Path graphs and restricted executions. A directed graph H is called a *path graph* if it has the following structure: the nodes of H can be partitioned into sets L_i, $i = 0, \ldots, D$, called *layers*, each with a distinguished node $v_i \in L_i$. The edges of H are of the form (v, v_{i+1}), where $0 \le i < D$ and $v \in L_i$. We refer to D as the *depth* of H. The path $\pi = v_0, v_1, \ldots, v_D$ is called the *main path* of H. Node v_0 is the *start node* and v_D is the *target node* (see the figure below). We assume that $L_D = \{v_D\}$. If H is a path graph and μ is a wake-up schedule, we refer to the pair $\langle H, \mu \rangle$ as a *path wake-up network*.



In path wake-up networks, we consider only wake-up schedules μ with v_0 as the start node, that is, $\mu_{v_0} = 0$ and $\mu_v \ge 0$ for $v \ne v_0$. The wake-up task is also different: our objective is to wake up only the target node v_D, and not necessarily all nodes.

We will consider a *restricted communication model* for path graphs, which differs slightly from the standard model described in section 2. Let H be a path graph and μ a wake-up schedule. Assume that the sets of active nodes are already defined for times $1, \ldots, t$. (As in a standard radio wake-up execution, if a node becomes active, it remains active through the end of the execution.) We denote by h_t the greatest index i of an active layer L_i (that is, a layer with at least one active node) at time t. We refer to L_{h_t} as the *head layer* at time t. A nonactive node v becomes activated in step t either if v wakes up spontaneously, or if v is on the main in the layer right after the head layer and is activated by some in-neighbor from the active layer. More formally,

(Act1) $\mu_v = t$, or

(Act2) $v = v_{h_t+1}$ and exactly one node in L_{h_t} transmits in step t.

Thus the difference between the standard and restricted models in path graphs is that the nodes v_i, for $i \le h_t$, cannot be activated by their in-neighbors in H even if exactly one of these in-neighbors transmits. (Nodes v_i, for $i > h_t + 1$, cannot be activated by their in-neighbors in either model, by the choice of h_t.) Given a protocol $\mathcal{P}$, we refer to the execution of $\mathcal{P}$ on $\langle H, \mu \rangle$ under the restricted model as a *restricted execution*. By $acttime^*_u(\mathcal{D}, H, \mu)$ and $ActSet^*_t(\mathcal{D}, H, \mu)$, respectively, we denote the activation time step of u and the set of active notes at time t in the restricted execution of $\mathcal{D}$ on $\langle H, \mu \rangle$ (with the asterisk indicating a restricted execution).

Recall that $m = Cn^{2/3} \log n$ is the length of our synchronizer, where $C = 31$ is the constant from Lemma 1.

LEMMA 2. *Let $B \geq 15$ be an integer constant. Consider the restricted execution of protocol $\mathcal{P}$ on a path wake-up network $\langle H, \mu \rangle$ with vertices from V. The probability that $\mathcal{P}$ does not activate the target node of H in time $T = (B + C)n^{5/3} \log n$ is at most $n^{-\frac{1}{16}Bn}$.*

Proof. Let $L_0, \ldots, L_D$ be the layers of H and $\pi = v_0, \ldots, v_D$ be the main path. The activation time step of a layer L_i is a random variable that is equal to t either if $t = \mu_{L_i}$, or if $h_t = i - 1$ and v_i receives the wake-up signal from L_{i-1} in time step t, whichever comes first.

Whenever the head index h_t increases, we call it an *advance*. Recall that v_0 is assumed to wake up at time step 0. We want to bound the probability of failure, i.e., that v_D is still not awakened at time $T = (B + C)n^{5/3} \log n$ or, equivalently, that $h_T < D$.

Given some fixed execution of $\mathcal{P}$ (that is, with fixed random choices), we group all time steps t in this execution into two categories:

S-steps: time steps t when at least one node in a head layer L_{h_t} executes its S-stage, and

U-steps: time steps t when all active nodes in L_{h_t} execute their U-stage.

Suppose that layer L_i was activated in this execution at time step t'. If the number of active nodes in L_i at time $t' + m$ is at most $n^{1/3}$, then we say that L_i is *fast*. Otherwise, we call L_i *slow*. (We emphasize that the notions of fast and slow layers are relative to a particular execution of $\mathcal{P}$; the same layer can be slow or fast, depending on the outcomes of random choices before t'.)

We now make two important observations. First, if L_i is fast, then, since $\mathcal{S}$ is an (n, k, m)-synchronizer with $k = n^{1/3}$, an advance to L_{i+1} will occur during the time period $[t' + 1, t' + m]$ with probability 1 (unless some nodes in layers after L_{i+1} wake up spontaneously, in which case we advance even further). The second observation, following from the previous one, is that U-steps occur only when the head layer is slow and thus has size at least $n^{1/3}$.

Now we look at the execution of $\mathcal{P}$ on $\langle H, \mu \rangle$ globally. The total number of S-steps is at most $nm = Cn^{5/3} \log n$. During these S-steps, $\mathcal{P}$ will complete advances from all fast layers in its execution with probability 1.

Since each slow layer has size at least $n^{1/3}$, the number of slow layers in any execution cannot exceed $n^{2/3}$. The total number of U-steps is at least $J \stackrel{\text{def}}{=} T - Cn^{5/3} \log n = Bn^{5/3} \log n$. Let X_i be a random variable such that $X_i = 1$ if there is an advance in the ith U-step, and 0 otherwise. Thus, it is sufficient to prove that

$$\Pr\left[\sum_{i=1}^{J} X_i < n^{2/3}\right] \leq n^{-\frac{1}{16}Bn}. \tag{1}$$

Consider one U-step, say the ith one, for some fixed execution history Ψ (that is, all random choices up to this point). At this step, the head layer has $x \geq n^{1/3}$ active nodes, and all these nodes transmit with probability $1/n$. The probability of an advance is the probability that exactly one of them transmits, so

$$\Pr[\, X_i = 1 \mid \Psi \,] \;=\; x \cdot \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{x-1} \;\geq\; \tfrac{1}{4} n^{-2/3} \;\stackrel{\text{def}}{=}\; p. \tag{2}$$

The last inequality holds because it is true for $x = n^{1/3}$ and $x(1 - 1/n)^{x-1}$ is a nondecreasing function for $x < n$. (Recall that $n \geq 2$.) Therefore $\Pr[\, X_i = 1 \,] \geq p$.

The idea behind the proof of (1) is this: we can think of the X_i's as a sequence of Bernoulli trials in which the probability of a success (an advance) is p. Then the

probability that in this process the number of advances in J U-steps is less than $n^{2/3}$ can be bounded by $n^{-\Omega(n)}$ using an appropriate tail inequality.

The flaw in the above argument is that the advance events (variables X_i) are not fully independent, and thus this is not truly a Bernoulli process. To make the argument more rigorous, we introduce independent random variables $Y_i \in \{0,1\}$, where $\Pr[\,Y_i = 1\,] = p$. For these variables, the Chernoff bound (see [27], for example), states that $\Pr[\,\sum_{i=1}^{J} Y_i < K\,] \le \exp\left[-\frac{1}{2}Jp(1-K/(Jp))^2\right]$, for any $0 \le K < Jp$. For $K = n^{2/3}$ and $B \ge 15$ we have

$$\left(1 - \frac{n^{2/3}}{Jp}\right)^2 \;=\; \left(1 - \frac{4}{Bn^{1/3}\log n}\right)^2 \;\ge\; \left(1 - \frac{4}{B}\right)^2 \ge \frac{1}{2}\,.$$

Therefore

$$\begin{aligned}\Pr\left[\sum_{i=1}^{J} Y_i < n^{2/3}\right] &\le e^{-\frac{1}{2}Jp(1-n^{2/3}/(Jp))^2} \\ &\le e^{-\frac{1}{16}Bn\log n} \\ &\le n^{-\frac{1}{16}Bn}.\end{aligned}$$

To complete the proof it is now sufficient to show that

$$\Pr\left[\sum_{i=1}^{J} X_i < K\right] \le \Pr\left[\sum_{i=1}^{J} Y_i < K\right] \tag{3}$$

for each K. To show (3), we introduce new random variables $Z_i \in \{0,1\}$, $i = 1,\ldots,J$, where, for each history Ψ (the random choices up to before the ith U-step), we define

$$\begin{aligned}\Pr[\,Z_i = 1 \mid \Psi \;\&\; Y_i = 1\,] &= 0, \\ \Pr[\,Z_i = 1 \mid \Psi \;\&\; Y_i = 0\,] &= \frac{\Pr[\,X_i = 1|\Psi\,] - p}{1-p}.\end{aligned}$$

Then processes $\{X_i\}$ and $\{Y_i + Z_i\}$ are stochastically equivalent. More precisely, for $i = 1,\ldots,J$ and any history Ψ,

$$\begin{aligned}\Pr[\,Y_i + Z_i = 1 \mid \Psi\,] &= \Pr[\,Y_i = 1 \,\&\, Z_i = 0 \mid \Psi\,] + \Pr[\,Y_i = 0 \,\&\, Z_i = 1 \mid \Psi\,] \\ &= p + (1-p)\cdot\frac{\Pr[\,X_i = 1 \mid \Psi\,] - p}{1-p} \\ &= \Pr[\,X_i = 1 \mid \Psi\,],\end{aligned}$$

and thus $\Pr[\,Y_i + Z_i = 1\,] = \Pr[\,X_i = 1\,]$ as well. Therefore

$$\Pr\left[\sum_{i=1}^{J} X_i < K\right] \;=\; \Pr\left[\sum_{i=1}^{J}(Y_i + Z_i) < K\right] \;\le\; \Pr\left[\sum_{i=1}^{M} Y_i < K\right],$$

completing the proof of (3). This also completes the proof of (1) and the whole lemma. □

LEMMA 3. *There exists a deterministic protocol $\mathcal{D}$ that, under the restricted model, for any path wake-up network $\langle H, \mu\rangle$ with at most n nodes, activates the target node of H in time $O(n^{5/3}\log n)$.*

Proof. Let $T = (B + C)n^{5/3} \log n$ be the running time from Lemma 2, where $C = 31$ and $B \geq 15$. Each path graph has out-degree 1, so the number of (labeled) n-vertex path graphs is at most n^n. We need to be concerned only with wake-up schedules μ that satisfy $0 \leq \mu_v \leq T + 1 \leq 2T$ for all v and $\mu_{v_0} = 0$ (where v_0 is the start node). There are at most $(2T)^n$ such wake-up schedules. Thus, by Lemma 2, the probability that $\mathcal{P}$ does not complete the wake-up during a restricted execution on some n-vertex path graph and some wake-up schedule is at most

$$\begin{aligned} n^{-\frac{1}{16}Bn} \cdot n^n (2T)^n &\leq \left[n^{-\frac{1}{16}B} \cdot n \cdot 2(B + C)n^{5/3} \log n \right]^n \\ &\leq \left[n^{-\frac{1}{16}B + \log(B+C) + 5} \right]^n \\ &< 1, \end{aligned}$$

as long as $B \geq 15$ and $-\frac{1}{16}B + \log(B + C) + 5 < 0$. Any $B \geq 207$ satisfies these conditions. This implies that there exists a deterministic protocol $\mathcal{D}$ that, for any n-vertex path graph and any wake-up schedule, completes the wake-up task (under the restricted model) in time $O(n^{5/3} \log n)$. ∎

THEOREM 1. *There exists a deterministic protocol that completes the wake-up process in each n-vertex strongly connected directed graph in time $O(n^{5/3} \log n)$.*

Proof. We prove that protocol $\mathcal{D}$ from Lemma 3 satisfies the theorem. Let $T = O(n^{5/3} \log n)$ be the running time of protocol $\mathcal{D}$ from Lemma 3, in the restricted model, on any path wake-up network $\langle H, \mu \rangle$, where the vertex set of H is a subset of V. We show that $\mathcal{D}$ completes the wake-up in *any* graph with vertex set V under any wake-up schedule in time $T = O(n^{5/3} \log n)$.

Fix a wake-up network $\langle G, \omega \rangle$, and denote by v_0 the start node, where $\omega_{v_0} = 0$ and $\omega_u \geq 0$ for all $u \in G$. Let v be some arbitrary but fixed target node $v \in G$, and let $\pi = v_0, v_1, \ldots, v_d = v$ be the shortest path from v_0 to v. We define a subgraph H_v of G as follows: the vertices of H_v are all nodes $v_0, \ldots, v_d$ and all in-neighbors of nodes $v_1, \ldots, v_d$. The edges of H_v are all edges (u, v_i) in G such that $u \in H_v$ and $(u, v_j) \notin G$ for $j > i$. Clearly, H_v is a path graph with main path $\pi = v_0, v_1, \ldots, v_d = v$. As usual, we denote the layers of H_v by $L_0, \ldots, L_d$. (See Figure 1.)

Define a wake-up schedule μ for nodes $u \in H_v$ by $\mu_u = acttime_u(\mathcal{D}, G, \omega)$, the activation time of u in G under protocol $\mathcal{D}$ and wake-up schedule ω. We compare the two following executions of $\mathcal{D}$:

(i) the execution of $\mathcal{D}$ on G with wake-up schedule ω, and

(ii) the restricted execution of $\mathcal{D}$ on H_v with wake-up schedule μ.

The difficulty we need to address is that the nodes v_i may have different neighborhoods in G and in H_v. All in-neighbors of v_i in H_v are in L_{i-1}, while in G, v_i may also have in-neighbors in layers $L_i, \ldots, L_d$. Nevertheless, we claim that the following invariant holds at each time $t = 0, 1, \ldots, T$:

$$\forall\, u \in H_v : \quad u \in ActSet_t(\mathcal{D}, G, \omega) \text{ iff } u \in ActSet_t^*(\mathcal{D}, H_v, \mu). \tag{4}$$

If (4) holds, then the activation times of each node (in particular, the target node) in both executions are the same. Since v was chosen arbitrarily, the theorem follows from Lemma 3. Thus, to complete the proof of the theorem, it is sufficient to prove (4).

The proof of invariant (4) is by induction on times $t = 0, \ldots, T$. For $t = 0$, (4) holds by the very definition of μ. Suppose that (4) is satisfied up to time t. We show that it is also satisfied at time $t + 1$.

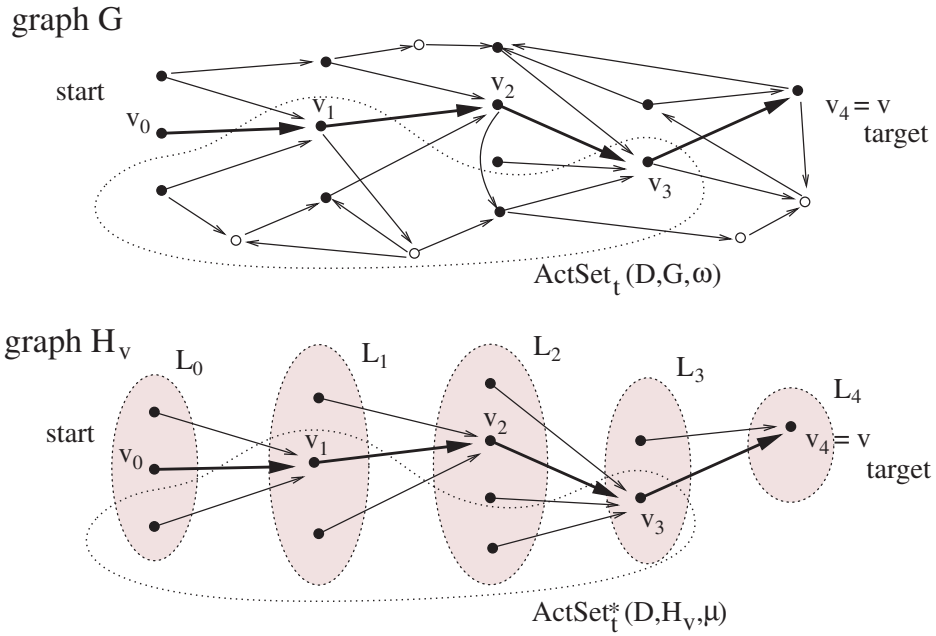


FIG. 1. *An illustration of the construction of H_v. The main path in G is shown with thick arrows. Vertices that belong to H_v are represented with filled circles.*

Note first that for each node $u \in H_v - \pi$, its activation times in both executions are the same, $acttime^*_u(\mathcal{D}, H_v, \mu) = acttime_u(\mathcal{D}, G, \omega) = \mu_u$. Thus (4) holds for nodes $u \in H - \pi$. The implication ($\Rightarrow$) is also immediate from the definition of μ. It is thus sufficient to prove the only implication ($\Leftarrow$) for $u \in \pi$.

It follows directly from invariant (4) applied to times $1, \ldots, t$ that the activation time steps (up to step $t-1$) of active nodes in H_v are the same in both executions of $\mathcal{D}$, the execution on $\langle G, \omega \rangle$, and the restricted execution on $\langle H_v, \mu \rangle$. Consequently, any $u \in ActSet^*_t(\mathcal{D}, H_v, \mu)$ performs the same action in step t in both executions. (This is because $\mathcal{D}^u$ does not depend on whether u woke up spontaneously or was activated by another node.)

As before, let h_t be the head layer of $\langle H_v, \mu \rangle$ at time t. For any node v_i, where $i \neq h_t + 1$, it follows directly from the definition of H_v and the rules of the restricted model that v_i is activated in step t of the execution of $\mathcal{D}$ on $\langle G, \omega \rangle$ iff it is activated in step t of the restricted execution of $\mathcal{D}$ on $\langle H_v, \mu \rangle$.

Consider v_i, for $i = h_t + 1$, and suppose that v_i is activated in H_v at time step $t \leq \mu_{v_i}$. It is sufficient to prove that $t < \mu_{v_i}$ is not possible. Toward contradiction, suppose $t < \mu_{v_i}$, that is, v_i was activated at step t using rule (Act2) by a wake-up signal from some node in L_{i-1}. Then, by the definitions of H_v and h_t, and also by invariant (4), we have $N_G(v_i) \cap ActSet_t(\mathcal{D}, G, \omega) = N_{H_v}(v_i) \cap ActSet^*_t(\mathcal{D}, H_v, \mu)$, and all nodes in this set are in the same state in both executions. It follows that v_i would have received the same message in G in step t as well—a contradiction with the assumption that $\mu_i > t$. This implies that (4) holds at time $t+1$, completing the proof. □

We emphasize that the above proof is not valid for arbitrary protocols, as we strongly use the fact that in our protocol $\mathcal{D}$ the active nodes do not use any information about how they have been activated (spontaneously or by a wake-up signal).

Recall that in this section we made a simplifying assumption that the node labels are consecutive integers $1, 2, \ldots, n$. Theorem 1 remains valid, however, if the node labels are chosen from a range $\ell_{\min}, \ldots, \ell_{\max}$, where $\ell_{\max} - \ell_{\min} \le An$, for some constant A, for in this case we can simply use a protocol for auxiliary labels $0, 1, \ldots, An$ (that is, a node with label ℓ uses label $\ell - \ell_{\min}$ to identify itself). This will not affect the asymptotic performance of the protocol.

5. A randomized wake-up protocol. In [23], the authors presented a randomized wake-up protocol for complete graphs. We now give a randomized Monte-Carlo wake-up protocol $\mathcal{R}$ for general multihop radio networks (even without node labels.) $\mathcal{R}$ receives on input a parameter $\epsilon > 0$, which is the desired bound on the probability of failure.

PROTOCOL $\mathcal{R}$. The pseudocode of the protocol for an arbitrary node v is

for $s \leftarrow 0, 1, \ldots, \log n - 1$ **do**
  repeat $b = 22 \log(n/\epsilon)$ **times** *// stage s starts*
    TRANSMIT with probability $2^s/n$

We start with the following technical lemma from [23].

LEMMA 4 (see [23]). *Suppose we have k numbers, $p_1, p_2, \ldots, p_k \in (0, \frac{1}{2}]$, such that $\frac{1}{2} \le \sum_{i=1}^{k} p_i \le 2$. Then*

$$\sum_{i=1}^{k} p_i \prod_{\substack{j=1 \\ j \neq i}}^{k} (1 - p_j) \ge \frac{1}{32}.$$

THEOREM 2. *Let $\epsilon > 0$ and let $\langle G, \omega \rangle$ be any wake-up network with n nodes and diameter D. With probability at least $1 - \epsilon$, $\mathcal{R}$ completes wake-up in $\langle G, \omega \rangle$ in time $O(D \log n \log(n/\epsilon))$.*

Proof. We divide the execution of $\mathcal{R}$ into $\log n$ stages, where stage s is simply the s-th iteration of the **for** loop. Recall that $b = 22 \log(n/\epsilon)$ is the number of iterations of the inner **repeat** loop. By $B = b \log n$ we denote the total execution time of $\mathcal{R}$ (for a single node).

Fix a wake-up network $\langle G, \omega \rangle$, and consider an execution of $\mathcal{R}$ on $\langle G, \omega \rangle$. Let v_0 be the start node (the one that wakes up first). For any node v, let $t_v = \mathit{acttime}_v$ and $\bar{t}_v = \min_{u \in N_v} \mathit{acttime}_u$. Thus $t_v - \bar{t}_v$ is the time (a random variable) it takes for v to become activated after the time when the first of its in-neighbors gets activated.

Call a node v an *obstruction* during an execution of $\mathcal{R}$ if $\bar{t}_v < \infty$ but $t_v > \bar{t}_v + B$. If there is no obstruction, then, clearly, the execution of $\mathcal{R}$ completes after at most $DB = O(D \log n \log(n/\epsilon))$ steps. Thus it remains to prove that the probability that there exists an obstruction during an execution of $\mathcal{R}$ is at most ϵ.

Consider some fixed node v, and fix some value of $\bar{t}_v$. To simplify notation, we can in fact assume that $\bar{t}_v = 0$. If $\omega_v < B$, then v is not an obstruction with probability 1, and we are done, so we can assume that $\omega_v \ge B$. We now need to estimate the probability that v will receive a wake-up signal from N_v in at most B steps.

For $u \in N_v$, let $p_{u,t}$ be the probability that u transmits at time step t (that is, $p_{u,t} = 2^s/n$ if u is in stage s at time t), and let $\pi_t = \sum_{u \in N_v} p_{u,t}$. We have $\pi_0 = 0$, $\pi_{B-b+1} \ge \frac{1}{2}$, and π_t is nondecreasing. By the definition of $\mathcal{R}$, for any $t = 1, \ldots, B - b$, we have $\pi_{t+b} \le 2\pi_t + 1$, since in b steps the active nodes will double their probability of transmission, and each other node contributes at most $1/n$ to π_{t+b}. Choose the largest t' for which $\pi_{t'-1} < \frac{1}{2}$. Then $\pi_{t'} \ge \frac{1}{2}$ and $\pi_{t'+b-1} \le 2\pi_{t'-1} + 1 \le 2$. We

conclude that

$$\frac{1}{2} \;\le\; \pi_t \;\le\; 2 \quad \text{for } t = t', t'+1, \ldots, t'+b-1. \tag{5}$$

Thus, by Lemma 4, for each time $t = t', t'+1, \ldots, t'+b-1$, the probability that v receives a wake-up signal from N_v at time t is

$$\sum_{u \in N_v} p_{u,t} \prod_{z \in N_v \setminus \{u\}} (1 - p_{z,t}) \ge \frac{1}{32}.$$

Then the probability that v will not receive a wake-up signal in steps $1, 2, \ldots, B$ is at most $(\frac{31}{32})^b$. We conclude that the probability that v is an obstruction is bounded by $(\frac{31}{32})^b$ as well. Therefore the probability that there exists an obstruction node during the execution of $\mathcal{R}$ is at most

$$n\left(\frac{31}{32}\right)^b \;=\; n\left(\frac{31}{32}\right)^{22\log(n/\epsilon)} \;\le\; n\left(\frac{1}{2}\right)^{\log(n/\epsilon)} \;\le\; \epsilon,$$

completing the proof. □

We can modify $\mathcal{R}$ to obtain a Las Vegas protocol with low expected running time. Let $\mathcal{R}_0$ be $\mathcal{R}$ with $\epsilon = n^{-3}$, so that the running time of $\mathcal{R}_0$ is $T_0 = O(D \log^2 n)$. Let $T = O(n^{5/3} \log n)$ be the running time of the deterministic protocol $\mathcal{D}$ from the previous section. In this Las Vegas protocol $\mathcal{R}'$, each node executes the following steps: (1) run $\mathcal{R}_0$, and then suspend itself for time T_0, (2) repeat n times: run $\mathcal{D}$ and then suspend itself for time T. At least one run of $\mathcal{D}$ will complete without interference from nodes that wake up spontaneously and execute $\mathcal{R}_0$. Thus $\mathcal{R}'$ will wake up all nodes in time T_0 with probability at least $1 - \epsilon$, and in time $O(n^{8/3} \log n)$ with probability 1. By the choice of ϵ, its expected running time will be $O(D \log^2 n)$.

THEOREM 3. *Let $\langle G, \omega \rangle$ be any wake-up network with n nodes. Protocol $\mathcal{R}'$ described above completes wake-up in $\langle G, \omega \rangle$ in expected time $O(D \log^2 n)$.*

6. Leader election and clock synchronization. In the leader election problem, we want to designate one node as *the leader*, and to announce its identity to all nodes in the network. In the clock synchronization problem, upon the completion of the protocol, all nodes must agree on a common global time. In this section we show that any wake-up protocol $\mathcal{W}$ (deterministic or randomized) can be transformed into a leader election protocol or a clock synchronization protocol with only a logarithmic overhead.

Let $\mathcal{W}$ be a wake-up protocol. For any node v, let $\mathcal{W}^v$ be the instance of $\mathcal{W}$ executed by node v. (Formally, the execution of $\mathcal{W}$ depends on the label, but, for simplicity of notation, we write $\mathcal{W}^v$ instead of $\mathcal{W}^{\ell_v}$.) Fix n, and by $w = w(n)$ denote the running time of $\mathcal{W}$ if all node labels are drawn from the range $\ell_{\min}, \ldots, \ell_{\max}$, where $\ell_{\max} - \ell_{\min} \le An$, for some constant A. We can assume, without loss of generality, that in protocol $\mathcal{W}$ each node transmits only for at most w time steps since activation. Note that our wake-up protocols described earlier in the paper satisfy this requirement.

Recall that, without loss of generality, we assume that $\ell_{\min} = 0$. We can then think of the labels as binary strings of length $\log(An)$, and by $\ell_v[i, \ldots, j]$ we denote the string consisting of the bits of ℓ_v on positions $i, i+1, \ldots, j$, counting from left to right (that is, highest bits first).

6.1. Leader election. The leader election protocol consists of two stages: the *wake-up stage* and the *election stage*. We first describe both stages informally, and then give a pseudocode for the complete protocol.

Wake-up stage. In this stage all nodes become activated. Each node v, once it wakes up, starts executing its wake-up protocol $\mathcal{W}^v$. After completing this protocol, v suspends itself for w time steps. The purpose of this suspension period is to ensure that the wake-up stage does not overlap with the election stage that follows.

Election stage. In this stage, all nodes in the network select the leader to be the node $\hat{v}$ that possesses the smallest label. The selection process is performed in $\log(An)$ rounds, where in round r, for $r = 0, 1, \ldots, \log(An) - 1$, the ith highest bit of $\ell_{\hat{v}}$ is announced. At the beginning of round r all nodes know the top r bits of $\ell_{\hat{v}}$. Let $min\_label = \ell_{\hat{v}}[0, \ldots, r-1]$ be the string consisting of these bits. Nodes v for which $\ell_v[0, \ldots, r-1] = min\_label$ are potential candidates for the leader. If such a candidate node v also has a 0 on the rth bit, it informs other nodes about its existence by initiating a wake-up process. If no wake-up process is initiated in round r, by the end of the round all nodes conclude that the ith most significant bit in $\ell_{\hat{v}}$ is 1.

The difficulty that we need to overcome is that, due to lack of synchronization, the division into consecutive rounds must be based on the local clocks. This means that very likely there will be some time overlaps (of size at most w—the maximum offset between the activation times of any two nodes) between neighboring rounds of different nodes. In order to avoid simultaneous transmissions of signals belonging to two different rounds, we will pad each round with two wait periods of length w, one at the beginning and one at the end. More specifically, every round is split into four subrounds, each of length w. If a node v is a candidate for the leader and initiates the transmission process by itself, this is always done in the beginning of the second subround. On the other hand, v can be activated to start its wake-up process by another node u at any time during the first, second, or third subround.

Leader election protocol $\mathcal{E}$. Below, we give a pseudocode for the protocol $\mathcal{E}^v$ executed by a node v. We use small capitals font, START, EXECUTE, etc., for commands related to timing. Variable *clock* measures the local time of a node. The scopes of compound statements are indicated by indentation. The chosen leader $\hat{v}$ is the node whose local variable *am_leader* has value *true* after the protocol is complete.

```
SLEEPUNTIL wake_up_signal (spontaneous or external)
START clock ← 0
EXECUTE 𝒲^v (w steps)
WAITUNTIL clock = 2w
min_label ← [ ]                          // empty string
for r ← 0, 1, ..., log(An) − 1 do
    bit ← 1                              // round r starts, local time is clock = (4r + 2)w
    if ℓ_v[0, ..., r] = min_label ∘ 0 then
        bit ← 0
        WAITUNTIL wake_up_signal or clock = (4r + 3)w
    else
        WAITUNTIL wake_up_signal or clock = (4r + 5)w
        if wake_up_signal then bit ← 0
    if bit = 0 then EXECUTE W^v (w steps)
    min_label ← min_label ∘ bit
    WAITUNTIL clock = (4r + 6)w
am_leader ← (ℓ_v = min_label)
```

One detail that we should clarify is that, when we simulate $\mathcal{W}$ in protocol $\mathcal{E}$ during the election stage, all nodes "pretend" to be asleep during the wait periods,

and nodes that initiate wake-up "pretend" to be awakened spontaneously. During the simulation of a wake-up process, each node uses an auxiliary clock to simulate a local clock used by $\mathcal{W}$.

To justify correctness, it is sufficient to show that there will be no interference between wake-up processes from different rounds. The crucial property used in the argument is that the (global) activation times (in the wake-up phase) of any two nodes differ by at most w. This implies, in particular, that two rounds of different nodes can overlap only if they are consecutive, and if so, they can overlap on at most w time steps. More specifically, the only possible overlap is when the fourth subround of a node in round r overlaps the first subround of another node that is in round $r+1$. (In our argument, to avoid dealing separately with the wake-up phase, we can treat the wake-up phase as round -1 of the election phase.)

If a wake-up process in round r is initiated at a (global) time t, then all nodes will be activated by time $t+w$, and they will have completed their transmissions at time $t+2w$. Since wake-ups are initiated at the beginning of the second subround, and the local times are shifted by at most w, it follows that, in each round r, each node will be activated no later than at the end of its third subround and thus complete its transmissions by the end of its round r.

Fix a node v that is first to execute its transmission from round $r+1$. This means that v initiates a wake-up process, and it starts transmitting in the first step of its second subround. But, again, using the fact that the offset between different rounds is at most w, by that time all other nodes completed their transmissions from round r.

Thus we obtain the following result.

THEOREM 4. (a) *Suppose that $\mathcal{W}$ is a deterministic wake-up protocol with running time $w(n)$. Then $\mathcal{W}$ can be converted into a leader election protocol $\mathcal{E}$ with running time $O(w(n)\log n)$.*

(b) *Suppose that $\mathcal{W}$ is a randomized wake-up protocol that completes the wake-up task in time $w_\epsilon(n)$ with probability at least $1-\epsilon$. Then $\mathcal{W}$ can be converted into a randomized leader election protocol $\mathcal{E}$ that completes the election task in time $O(w_\epsilon(n)\log n)$ with probability at least $1-O(\epsilon\log n)$.*

6.2. Clock cynchronization. It is not difficult to extend the leader election protocol $\mathcal{E}$ from the previous subsection to perform clock synchronization. We first run $\mathcal{E}$. Once the leader $\hat{v}$ has been elected, $\hat{v}$ waits for w steps and then broadcasts its local time to all other nodes using some broadcasting protocol $\mathcal{B}$ (see, for example [14]). When the nodes perform $\mathcal{B}$, they increment their global time counter at each step, to make sure that the correct value is transmitted. (A similar idea appeared in [30].) As explained in the introduction, broadcasting can be reduced to wake-up, so we can assume that the running time of $\mathcal{B}$ is at most $w(n)$.

Thus we obtain the following result.

THEOREM 5. (a) *Suppose that $\mathcal{W}$ is a deterministic wake-up protocol with running time $w(n)$. Then $\mathcal{W}$ can be converted into a deterministic clock synchronization protocol $\mathcal{C}$ with running time $O(w(n)\log n)$.*

(b) *Suppose that $\mathcal{W}$ is a randomized wake-up protocol that completes the wake-up task in time $w_\epsilon(n)$ with probability at least $1-\epsilon$. Then $\mathcal{W}$ can be converted into a randomized clock synchronization protocol $\mathcal{C}$ that completes the election task in time $O(w_\epsilon(n)\log n)$ with probability at least $1-O(\epsilon\log n)$.*

6.3. New protocols. Recall that our deterministic wake-up protocol $\mathcal{D}$ runs in time $O(n^{5/3}\log n)$, our Monte Carlo protocol $\mathcal{R}$ in time $O(D\log n\log(n/\epsilon))$ and with failure probability ϵ, and our Las Vegas protocol $\mathcal{R}'$ runs in expected time $O(D\log^2 n)$.

For broadcasting, we can use protocols from [14] or [24, 17], and any randomized broadcasting protocol in [5, 25, 17], or, alternatively, we can adapt wake-up protocols from our paper. Summarizing the results in this section, we obtain the following corollary.

COROLLARY 1. *There exist protocols for leader election and clock synchronization in radio networks with the following performance in n-node networks:*

(a) *a deterministic protocol with running time* $O(n^{5/3}\log^2 n)$,

(b) *a Monte Carlo protocol with running time* $O(D\log n\log(n/\epsilon))$ *and failure probability at most* ϵ *for any* $\epsilon > 0$,

(c) *a Las Vegas protocol with expected running time* $O(D\log^3 n)$.

Results in Corollary 1(a) and (b) are straightforward from Theorems 1, 2, 4, 5. (To obtain part (b), choose the error probability of $\mathcal{W}$ in Theorem 5(b) to be $\epsilon' = A\epsilon/\log n$, for some sufficiently small constant $A > 0$.)

The randomized Las Vegas protocols in Corollary 1(c) are obtained in the similar way as was the algorithm Las Vegas for the wake-up problem: by taking the corresponding Monte Carlo protocol with sufficiently small failure probability, say $\epsilon = n^{-3}$, followed by n repetitions of a deterministic protocol. These algorithms succeed in time $O(D\log^3 n)$ with probability at least $1-\epsilon\log n$, and in time $O(n^{8/3}\log n)$ with probability 1, so their expected running time will be $O(D\log^3 n)$.

7. Final comments. In this paper, we initiated the study of wake-up protocols in a general model of multihop radio networks without collision detection, where the topology of the network is unknown.

In our model we assume that all nodes know n (the number of nodes) and that the labels are integers not greater than $O(n)$ (except for the randomized wake-up protocol that does not use labels at all). Some of these assumptions can be relaxed. For example, it is not necessary that n be known, for otherwise the protocols can employ the standard doubling technique: the computation is divided into phases, where in phase $i = 1, 2, \ldots$ we run the protocol for $n = 2^i$, with appropriate waiting periods to take into account lack of initial synchronization. (See the discussion in [14].) These modified algorithms still work correctly, in fact, when the labels are unique but unbounded, although then the running time depends on the maximum label value.

Since the lower bounds for radio broadcasting (see, e.g., [8, 15, 16, 26]) apply to the wake-up problem, no randomized wake-up protocol can be faster than $\Omega(D\log(n/D))$. Thus our randomized protocol is within a polylogarithmic factor from the optimum. However, in the deterministic case, our results still leave a substantial gap between the lower bound of $\Omega(n\log n)$ and the upper bound $O(n^{5/3}\log n)$ achieved by our protocol $\mathcal{D}$.

Subsequent to the submission of this paper, new work has appeared where this gap has been significantly reduced. First, in [13], Chlebus and Kowalski improved the upper bound to $O(n^{3/2}\log n)$. More recently, Chlebus et al. [11] designed a deterministic protocol with running time $\Omega(n\log^2 n)$—within only an $O(\log n)$-factor from the optimum.

The optimum complexity of the wake-up problem (as well as a seemingly simpler broadcasting problem), in both deterministic and randomized cases, remains unknown.

Acknowledgments. We wish to express our gratitude to the anonymous referees whose numerous and insightful comments helped us improve presentation and correct some inaccuracies in the earlier version of this paper. We also would like to thank Mart Molle for useful discussions on network protocols.

REFERENCES

[1] N. Abramson, *The ALOHA System*, in Computer Networks, N. Abramson and F. Kuo, eds., Prentice–Hall, Englewood Cliffs, NJ, 1973, pp. 501–518.

[2] E. Arkin, M. Bender, D. Ge, S. He, and J. Mitchell, *Improved approximation algorithms for the freeze-tag problem*, in Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'03), San Diego, CA, 2003, ACM, New York, pp. 295–303.

[3] B. Awerbuch, *Complexity of network synchronization*, J. ACM, 32 (1985), pp. 804–823.

[4] B. Awerbuch, S. Kutten, Y. Mansour, B. Patt-Shamir, and G. Varghese, *Time optimal self-stabilizing synchronization*, in Proceedings of the 25th ACM Symposium on Theory of Computing (STOC'93), San Diego, CA, 1993, ACM, New York, pp. 6526–6561.

[5] R. Bar-Yehuda, O. Goldreich, and A. Itai, *On the time complexity of broadcast in radio networks: An exponential gap between determinism and randomization*, J. Comput. System Sci., 45 (1992), pp. 104–126.

[6] R. Bar-Yehuda, O. Goldreich, and A. Itai, *Efficient emulation of single-hop radio network with collision detection on multi-hop radio network with no collision detection*, Distrib. Comput., 5 (1991), pp. 67–71.

[7] R. Bar-Yehuda, A. Israeli, and A. Itai, *Multiple communication in multihop radio networks*, SIAM J. Comput., 22 (1993), pp. 875–887.

[8] D. Bruschi and M. Del Pinto, *Lower bounds for the broadcast problem in mobile radio networks*, Distrib. Comput., 10 (1997), pp. 129–135.

[9] I. Chlamtac and O. Weinstein, *The wave expansion approach to broadcasting in multihop radio networks*, IEEE Trans. Commun., 39 (1991), pp. 426–433.

[10] B. S. Chlebus, L. Gąsieniec, A. Gibbons, A. Pelc, and W. Rytter, *Deterministic broadcasting in unknown radio networks*, Distrib. Comput., 15 (2002), pp. 27–38.

[11] B. S. Chlebus, L. Gąsieniec, D. R. Kowalski, and T. Radzik, *On the wake-up problem in radio networks*, in Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05), Lisbon, Portugal, 2005, Lecture Notes in Comput. Sci. 3580, Springer, New York, 2005, pp. 347–359.

[12] B. S. Chlebus, L. Gąsieniec, A. Östlin, and J. M. Robson, *Deterministic radio broadcasting*, in Proceedings of the 27th International Colloquium on Automata, Languages and Programming (ICALP'00), Lecture Notes in Comput. Sci. 1853, Springer, New York, 2000, pp. 717–728.

[13] B. S. Chlebus and D. R. Kowalski, *A better wake-up in radio networks*, in Proceedings of the 23rd Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC'04), St. John's, NF, 2004, ACM, New York, pp. 266–274.

[14] M. Chrobak, L. Gąsieniec, and W. Rytter, *Fast broadcasting and gossiping in radio networks*, J. Algorithms, 46 (2003), pp. 1–20.

[15] A. E. F. Clementi, A. Monti, and R. Silvestri, *Selective families, superimposed codes, and broadcasting on unknown radio networks*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'01), Washington, DC, 2001, SIAM, Philadelphia, pp. 709–718.

[16] A. Clementi, A. Monti, and R. Silvestri, *Distributed broadcast in radio networks of unknown topology*, Theoret. Comput. Sci., 302 (2003), pp. 337–364.

[17] A. Czumaj and W. Rytter, *Broadcasting algorithms in radio networks with unknown topology*, J. Algorithms, 60 (2006), pp. 115–143.

[18] A. DeBonis, L. Gąsieniec, and U. Vaccaro, *Optimal two-stage algorithms for group testing problems*, SIAM J. Comput., 34 (2005), pp. 1253–1270.

[19] M. J. Fischer, S. Moran, S. Rudich, and G. Taubenfeld, *The wakeup problem*, SIAM J. Comput., 25 (1996), pp. 1332–1357.

[20] L. Gąsieniec, A. Pelc, and D. Peleg, *The wakeup problem in synchronous broadcast systems*, SIAM J. Discrete Math., 14 (2001), pp. 207–222.

[21] P. Indyk, *Explicit constructions of selectors and related combinatorial structures, with applications*, in Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'02), San Francisco, CA, 2002, SIAM, Philadelphia, pp. 697–704.

[22] T. Jurdziński, M. Kutyłowski, and J. Zatopiański, *Efficient algorithms for leader election in radio networks*, in Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC'02), Monterey, CA, 2002, ACM, New York, pp. 51–57.

[23] T. Jurdziński and G. Stachowiak, *Probabilistic algorithms for the wakeup problem in single-hop radio networks*, Theory Comput. Syst., 38 (2005), pp. 347–367.

[24] D. R. Kowalski and A. Pelc, *Faster deterministic broadcasting in ad hoc radio networks*, SIAM J. Discrete Math., 18 (2004), pp. 332–346.

[25] D. R.Kowalski and A. Pelc, *Broadcasting in undirected ad hoc radio networks*, Distrib. Comput., 18 (2005), pp. 43–57.
[26] E. Kushilevitz and Y. Mansour, *An* $\Omega(D \log(N/D))$ *lower bound for broadcast in radio networks*, SIAM J. Comput., 27 (1998), pp. 702–712.
[27] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.
[28] N. Lynch, *Distributed Algorithms*, Morgan-Kauffman, San Francisco, 1997.
[29] R. M. Metcalfe and D. R. Boggs, *Ethernet: Distributed packet switching for local computer networks*, ACM Communications, 19 (1976), pp. 395–404.
[30] D. Peleg, *Deterministic Radio Broadcast with No Topological Knowledge*, unpublished manuscript, 2000.
[31] R. Zheng, J. C. Hou, and L. Sha, *Asynchronous wakeup in ad hoc networks*, in Proceedings of the 4th ACM Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC'03), Annapolis, MD, 2003, ACM, New York, pp. 35–45.

SIAM J. COMPUT.
Vol. 36, No. 5, pp. 1472–1493

QUANTUM AND CLASSICAL STRONG DIRECT PRODUCT THEOREMS AND OPTIMAL TIME-SPACE TRADEOFFS*

HARTMUT KLAUCK†, ROBERT ŠPALEK‡, AND RONALD DE WOLF‡

Abstract. A strong direct product theorem says that if we want to compute k independent instances of a function, using less than k times the resources needed for one instance, then our overall success probability will be exponentially small in k. We establish such theorems for the classical as well as quantum query complexity of the OR-function. This implies slightly weaker direct product results for all total functions. We prove a similar result for quantum communication protocols computing k instances of the disjointness function. Our direct product theorems imply a time-space tradeoff $T^2S = \Omega(N^3)$ for sorting N items on a quantum computer, which is optimal up to polylog factors. They also give several tight time-space and communication-space tradeoffs for the problems of Boolean matrix-vector multiplication and matrix multiplication.

Key words. complexity theory, quantum computing, lower bounds, decision trees, communication complexity

AMS subject classifications. 03D15, 68Q10, 81P68, 68Q17, 68Q05

DOI. 10.1137/05063235X

1. Introduction.

1.1. Direct product theorems.

For every reasonable model of computation one can ask the following fundamental question:

> How do the resources that we need for computing k independent instances of f scale with the resources needed for one instance and with k?

Here the notion of "resource" needs to be specified. It could refer to time, space, queries, or communication. Similarly we need to define what we mean by "computing f," for instance, whether we allow the algorithm some probability of error and whether this probability of error is average-case or worst-case.

In this paper we consider two kinds of resources, queries and communication, and allow our algorithms some error probability. An algorithm is given k inputs $x^1, \ldots, x^k$ and has to output the vector of k answers $f(x^1), \ldots, f(x^k)$. The issue is how the algorithm can optimally distribute its resources among the k instances it needs to compute. We focus on the relation between the total amount T of resources available and the best-achievable success probability σ (which could be average-case or worst-case). Intuitively, if every algorithm with t resources must have some constant error probability when computing *one* instance of f, then for computing k instances we expect a constant error on each instance and hence an exponentially small success

*Received by the editors May 25, 2005; accepted for publication (in revised form) May 11, 2006; published electronically February 5, 2007. A preliminary version of this paper appeared in *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, 2004, pp. 12–21.
http://www.siam.org/journals/sicomp/36-5/63235.html

†Institut for Informatik, Goethe Universität Frankfurt, Frankfurt 60054, Germany (klauck@thi.informatik.uni-frankfurt.de). The work of this author was supported by Canada's NSERC and MITACS, and by DFG grant KL 1470/1.

‡CWI, Kruislaan 413, Amsterdam 1098 SJ, The Netherlands (sr@cwi.nl, rdewolf@cwi.nl). The work of the second author was partially supported by the European Commission under projects RESQ, IST-2001-37559 and QAP, IST-2005-015848. The work of the third author was partially supported by a Veni grant from the Netherlands Organization for Scientific Research (NWO), and by the European Commission under projects RESQ, IST-2001-37559 and QAP, IST-2005-015848.

probability for the k-vector as a whole. Such a statement is known as a *weak* direct product theorem:

If $T \approx t$, then $\sigma = 2^{-\Omega(k)}$.

Here "$T \approx t$" informally means that T is not much smaller than t. However, even if we give our algorithm roughly kt resources, on average it still has only t resources available per instance. So even here we expect a constant error per instance and an exponentially small success probability overall. Such a statement is known as a *strong* direct product theorem:

If $T \approx kt$, then $\sigma = 2^{-\Omega(k)}$.

Strong direct product theorems (SDPTs), though intuitively very plausible, are generally hard to prove and sometimes not even true. Shaltiel [Sha01] exhibits a general class of examples where SDPTs fail. This applies, for instance, to query complexity, communication complexity, and circuit complexity. In his examples, success probability is taken under the uniform probability distribution on inputs. The function is chosen such that for most inputs, most of the k instances can be computed quickly and without any error probability. This leaves enough resources to solve the few hard instances with high success probability. Hence for his functions, with $T \approx tk$, one can achieve average success probability close to 1.

Accordingly, we can establish direct product theorems only in special cases. Examples are the SDPT for "decision forests" by Nisan, Rudich, and Saks [NRS94], the direct product theorem for "forests" of communication protocols by Parnafes, Raz, and Wigderson [PRW97], and Shaltiel's SDPT for "fair" decision trees and his discrepancy bound for communication complexity [Sha01]. Shaltiel's result for discrepancy was recently strengthened to an SDPT for the "corruption" measure under product distributions on the inputs by Beame et al. [BPSW05]. There also has been recent progress on the related issue of *direct sum* results; see, e.g., [CSWY01, BJKS02b, BJKS02a] and the references therein. A direct sum theorem states that computing k instances with overall error ε requires roughly k times as many resources as computing one instance with error ε. Clearly, SDPTs always imply direct sum results, since they state the same resource lower bounds even for algorithms whose overall error is allowed to be exponentially close to 1, rather than at most ε.

In the quantum case, much less work has been done. Aaronson [Aar04, Theorem 10] established a direct product result for the unordered search problem that lies between the weak and the strong theorems: Every T-query quantum algorithm for searching k marked items among $N = kn$ input bits will have success probability $\sigma \leq \mathrm{O}\big(T^2/N\big)^k$. In particular, if $T \ll \sqrt{N} = \sqrt{kn}$, then $\sigma = 2^{-\Omega(k)}$.

Our main contributions in this paper are SDPTs for the OR-function in various settings. First consider the case of classical randomized algorithms. Let OR_n denote the n-bit OR-function, and let $f^{(k)}$ denote k independent instances of a function f. Any randomized algorithm with fewer than, say, $n/2$ queries will have a constant error probability when computing OR_n. Hence we expect an exponentially small success probability when computing $\mathrm{OR}_n^{(k)}$ using $\ll kn$ queries. We prove the following in section 3:

> *SDPT for classical query complexity.* Every randomized algorithm that computes $\mathrm{OR}_n^{(k)}$ using $T \leq \alpha kn$ queries has worst-case success probability $\sigma = 2^{-\Omega(k)}$ (for $\alpha > 0$ a sufficiently small constant).

For simplicity we have stated this result with σ being *worst-case* success probability, but the statement is also valid for the *average* success probability under a hard k-fold product distribution that is implicit in our proof.

This statement for OR actually implies a somewhat weaker strong product theorem for all *total* functions f, via the notion of *block sensitivity* $bs(f)$. Using techniques of Nisan and Szegedy [NS94], we can embed $\mathrm{OR}_{bs(f)}$ in f (with the promise that the weight of the OR's input is 0 or 1), while on the other hand we know that the classical bounded-error query complexity $R_2(f)$ is upper bounded by $bs(f)^3$ [BBC+01]. This implies the following:

> Every randomized algorithm that computes $f^{(k)}$ using $T \leq \alpha k R_2(f)^{1/3}$ queries has worst-case success probability $\sigma = 2^{-\Omega(k)}$ (for $\alpha > 0$ a sufficiently small constant).

This theorem falls short of being a true SDPT in having $R_2^{1/3}(f)$ instead of $R_2(f)$ in the resource bound. However, the other two main aspects of an SDPT remain valid: the linear dependence of the resources on k and the exponential decay of the success probability.

Next we turn our attention to *quantum* algorithms. Buhrman et al. [BNRW05] actually proved that roughly k times the resources for one instance suffices to compute $f^{(k)}$ with success probability *close to* 1 rather than exponentially small: $Q_2(f^{(k)}) = \mathrm{O}(kQ_2(f))$, where $Q_2(f)$ denotes the quantum bounded-error query complexity of f (such a result is not known to hold in the classical world). For instance, $Q_2(\mathrm{OR}_n) = \Theta(\sqrt{n})$ by Grover's search algorithm; thus $\mathrm{O}(k\sqrt{n})$ quantum queries suffice to compute $\mathrm{OR}_n^{(k)}$ with high success probability. In section 4 we show that if we make the number of queries slightly smaller, the best-achievable success probability suddenly becomes exponentially small:

> *SDPT for quantum query complexity.* Every quantum algorithm that computes $\mathrm{OR}_n^{(k)}$ using $T \leq \alpha k\sqrt{n}$ queries has worst-case success probability $\sigma = 2^{-\Omega(k)}$ (for $\alpha > 0$ a sufficiently small constant).

Our proof uses the polynomial method [BBC+01] and is completely different from the classical proof. The polynomial method was also used by Aaronson [Aar04] in his proof of a weaker quantum direct product theorem for the search problem, mentioned above. Our proof takes its starting point from his proof, analyzing the degree of a single-variate polynomial that is 0 on $\{0, \ldots, k-1\}$, at least σ on k, and between 0 and 1 on $\{0, \ldots, kn\}$. The difference between his proof and ours is that we partially factor this polynomial, which gives us some nice extra properties over Aaronson's approach of differentiating the polynomial, and that we use a strong result of Coppersmith and Rivlin [CR92]. In both cases (different) extremal properties of Chebyshev polynomials finish the proofs.

Again, using block sensitivity we can obtain a weaker result for all total functions:

> Every quantum algorithm that computes $f^{(k)}$ using $T \leq \alpha k Q_2(f)^{1/6}$ queries has worst-case success probability $\sigma = 2^{-\Omega(k)}$.

The third and last setting where we establish an SDPT is quantum communication complexity. Suppose Alice has an n-bit input x and Bob has an n-bit input y. These x and y represent sets, and $\mathrm{DISJ}_n(x, y) = 1$ if and only if those sets are disjoint. Note that DISJ_n is the negation of $\mathrm{OR}_n(x \wedge y)$, where $x \wedge y$ is the n-bit string obtained by bitwise AND-ing x and y. In many ways, DISJ_n has the same central role in communication complexity as OR_n has in query complexity. In particular, it is "co-NP complete" [BFS86]. The communication complexity of DISJ_n has been well studied: It takes $\Theta(n)$ bits of communication in the classical world [KS92, Raz92] and $\Theta(\sqrt{n})$ in the quantum world [BCW98, HW02, AA03, Raz03]. For the case where Alice and Bob want to compute k instances of disjointness, we establish the following SDPT in section 5:

SDPT for quantum communication complexity. Every quantum protocol that computes $\mathrm{DISJ}_n^{(k)}$ using $T \leq \alpha k\sqrt{n}$ qubits of communication has worst-case success probability $\sigma = 2^{-\Omega(k)}$ (for $\alpha > 0$ a sufficiently small constant).

Our proof uses Razborov's lower bound technique [Raz03] to translate the quantum protocol to a polynomial, at which point the polynomial results established for the quantum query SDPT take over. We can obtain similar results for other symmetric predicates. The same bound was obtained independently by Beame et al. [BPSW05, Corollary 9] for classical protocols under a specific input distribution, as a corollary of their SDPT for corruption.[1] We conjecture that the optimal result in the classical case has a communication bound of αkn rather than $\alpha k\sqrt{n}$, but cannot prove this.

One may also consider algorithms that compute the *parity* of the k outcomes instead of the vector of k outcomes. This issue has been well studied, particularly in circuit complexity, and generally goes under the name of *XOR lemmas* [Yao82, GNW95]. In this paper we focus mostly on the vector version but can prove similar strong bounds for the parity version. In particular, we state a classical strong XOR lemma in section 3.3 and can get similar strong XOR lemmas for the quantum case using the technique of Cleve et al. [CDNT98, section 3]. They show how the ability to compute the parity of any subset of k bits with probability $1/2 + \varepsilon$ suffices to compute the full k-vector with probability $4\varepsilon^2$. Hence our strong quantum direct product theorems imply strong quantum XOR lemmas.

1.2. Time-space and communication-space tradeoffs. Apart from answering a fundamental question about the computational models of (quantum) query complexity and communication complexity, our direct product theorems also imply a number of new and optimal time-space tradeoffs.

First, we consider the tradeoff between the time T and space S that a quantum circuit needs for *sorting* N numbers. Classically, it is well known that $TS = \Omega\big(N^2\big)$ and that this tradeoff is achievable [Bea91]. In the quantum case, Klauck [Kla03] constructed a bounded-error quantum algorithm that runs in time $T = \mathrm{O}((N \log N)^{3/2}/\sqrt{S})$ for all $(\log N)^3 \leq S \leq N/\log N$. He also claimed a lower bound $TS = \Omega\big(N^{3/2}\big)$, which would be close to optimal for small S but not for large S. Unfortunately there is an error in the proof presented in [Kla03] (Lemma 5 appears to be wrong). Here we use our SDPT to prove the tradeoff $T^2S = \Omega\big(N^3\big)$. This is tight up to polylogarithmic factors.

Secondly, we consider time-space and communication-space tradeoffs for the problems of *Boolean matrix-vector product* and *Boolean matrix product*. In the first problem there are an $N \times N$ matrix A and a vector b of dimension N, and the goal is to compute the vector $c = Ab$, where $c_i = \vee_{j=1}^{N} (A[i,j] \wedge b_j)$. In the setting of time-space tradeoffs, the matrix A is fixed and the input is the vector b. In the problem of matrix multiplication, two matrices have to be multiplied with the same type of Boolean product, and both are inputs.

Time-space tradeoffs for Boolean matrix-vector multiplication have been analyzed in an average-case scenario by Abrahamson [Abr90], whose results give a worst-case lower bound of $TS = \Omega\big(N^{3/2}\big)$ for classical algorithms. He conjectured that a worst-case lower bound of $TS = \Omega\big(N^2\big)$ holds. Using our classical direct product result we are able to confirm this; i.e., there is a matrix A, such that computing Ab requires

[1] We proved our result in February 2004 and published it on the quant-ph preprint server in the same month (http://www.arxiv.org/abs/quant-ph/0402123), while they proved theirs in the summer of 2004, unaware of our paper (personal communication with Paul Beame).

$TS = \Omega(N^2)$. We also show a lower bound of $T^2S = \Omega(N^3)$ for this problem in the quantum case. Both bounds are tight (the second within a logarithmic factor) if T is taken to be the number of queries to the inputs. We also get a lower bound of $T^2S = \Omega(N^5)$ for the problem of multiplying two matrices in the quantum case. This bound is close to optimal for small S; it is open whether it is close to optimal for large S.

Research on communication-space tradeoffs in the classical setting has been initiated by Lam, Tiwari, and Tompa [LTT92] in a restricted setting and by Beame, Tompa, and Yan [BTY94] in a general model of space-bounded communication complexity. In the setting of communication-space tradeoffs, players Alice and Bob are modeled as space-bounded circuits, and we are interested in the communication cost when given particular space bounds. For the problem of computing the matrix-vector product Alice receives the matrix A (now an input) and Bob receives the vector b. Beame, Tompa, and Yan gave tight lower bounds, e.g., for the matrix-vector product and matrix product over GF(2), but stated the complexity of Boolean matrix-vector multiplication as an open problem. Using our direct product result for quantum communication complexity, we are able to show that any quantum protocol for this problem satisfies $C^2S = \Omega(N^3)$. This is tight within a polylogarithmic factor. We also get a lower bound of $C^2S = \Omega(N^5)$ for computing the product of two matrices, which again is tight.

Note that no classical lower bounds for these problems were known previously and that finding better classical lower bounds than these remains open. The ability to show good quantum bounds comes from the deep relation between quantum protocols and polynomials implicit in Razborov's lower bound technique [Raz03].

2. Preliminaries.

2.1. Quantum query algorithms. We assume familiarity with quantum computing [NC00] and sketch the model of quantum query complexity, referring to [BW02] for more details, including details on the close relation between query complexity and degrees of multivariate polynomials. Suppose we want to compute some function f. For input $x \in \{0,1\}^N$, a *query* gives us access to the input bits. It corresponds to the unitary transformation

$$O : |i, b, z\rangle \mapsto |i, b \oplus x_i, z\rangle.$$

Here $i \in [N] = \{1, \ldots, N\}$ and $b \in \{0,1\}$; the z-part corresponds to the workspace, which is not affected by the query. We assume the input can be accessed only via such queries. A T-query quantum algorithm has the form $A = U_T O U_{T-1} \cdots O U_1 O U_0$, where the U_k are fixed unitary transformations, independent of x. This A depends on x via the T applications of O. The algorithm starts in initial S-qubit state $|0\rangle$, and its *output* is the result of measuring a dedicated part of the final state $A|0\rangle$. For a Boolean function f, the output of A is obtained by observing the leftmost qubit of the final superposition $A|0\rangle$, and its *acceptance probability* on input x is its probability of outputting 1.

One of the most interesting quantum query algorithms is Grover's search algorithm [Gro96, BBHT98]. It can find an index of a 1-bit in an n-bit input in expected number of $O(\sqrt{n/(|x|+1)})$ queries, where $|x|$ is the Hamming weight (number of 1's) in the input. If we know that $|x| \leq 1$, we can solve the search problem exactly using $\lceil \frac{\pi}{4}\sqrt{n} \rceil$ queries [BHMT02].

For investigating time-space tradeoffs we use the circuit model. A circuit accesses its input via an oracle such as a query algorithm. Time corresponds to the number

of gates in the circuit. We will, however, usually consider the number of queries to the input, which is obviously a lower bound on time. A quantum circuit uses space S if it works with S qubits only. We require that the outputs are made at predefined gates in the circuit, by writing their value to some extra qubits that may not be used later on. Similar definitions are made for classical circuits.

2.2. Communicating quantum circuits. In the model of quantum communication complexity, two players Alice and Bob compute a function f on distributed inputs x and y. The complexity measure of interest in this setting is the amount of communication. The players follow some predefined protocol that consists of local unitary operations and the exchange of qubits. The communication cost of a protocol is the maximal number of qubits exchanged for any input. In the standard model of communication complexity, Alice and Bob are computationally unbounded entities, but we are also interested in what happens if they have bounded memory, i.e., they work with a bounded number of qubits. To this end we model Alice and Bob as communicating quantum circuits, following Yao [Yao93].

A pair of communicating quantum circuits is actually a single quantum circuit partitioned into two parts. The allowed operations are local unitary operations and access to the inputs that are given by oracles. Alice's part of the circuit may use oracle gates to read single bits from her input, and Bob's part of the circuit may do so for his input. The communication C between the two parties is simply the number of wires carrying qubits that cross between the two parts of the circuit. A pair of communicating quantum circuits uses space S if the whole circuit works on S qubits.

In the problems we consider, the number of outputs is much larger than the memory of the players. Therefore we use the following output convention: The player who computes the value of an output sends this value to the other player at a predetermined point in the protocol. In order to make the models as general as possible, we furthermore allow the players to do local measurements and to throw qubits away as well as pick up some fresh qubits. The space requirement demands only that at any given time no more than S qubits are in use in the whole circuit.

A final comment regarding upper bounds: Buhrman, Cleve, and Wigderson [BCW98] showed how to run a query algorithm in a distributed fashion with small overhead in the communication. In particular, if there is a T-query quantum algorithm computing N-bit function f, then there is a pair of communicating quantum circuits with $O(T \log N)$ communication that computes $f(x \wedge y)$ with the same success probability. We refer to the book of Kushilevitz and Nisan [KN97] for more on communication complexity in general and to the surveys [Kla00, Buh00, Wol02] for more on its quantum variety.

3. Strong direct product theorem for classical queries. In this section we prove an SDPT for classical randomized algorithms computing k independent instances of OR_n. By Yao's principle, it is sufficient to prove it for deterministic algorithms under a fixed hard input distribution.

3.1. Nonadaptive algorithms. We first establish an SDPT for nonadaptive algorithms. We call an algorithm *nonadaptive* if, for each of the k input blocks, the maximum number of queries in that block is fixed before the first query. Note that this definition is nonstandard in fixing only the *number* of queries in each block rather than fixing all queried *indices* in advance. Let $\mathrm{Suc}_{t,\mu}(f)$ be the success probability of the best algorithm for f under μ that queries at most t input bits.

LEMMA 1. *Let $f : \{0,1\}^n \to \{0,1\}$, and let μ be an input distribution. Every nonadaptive deterministic algorithm for $f^{(k)}$ under μ^k with $T \le kt$ queries has success probability $\sigma \le \mathrm{Suc}_{t,\mu}(f)^k$.*

Proof. The proof has two steps. First, we prove by induction that nonadaptive algorithms for $f^{(k)}$ under general product distribution $\mu_1 \times \cdots \times \mu_k$ that spend t_i queries in the ith input x^i have success probability $\le \prod_{i=1}^k \mathrm{Suc}_{t_i,\mu_i}(f)$. Second, we argue that, when $\mu_i = \mu$, the value is maximal for $t_i = t$.

Following [Sha01, Lemma 7], we prove the first part by induction on $T = t_1 + \cdots + t_k$. If $T = 0$, then the algorithm has to guess k independent random variables $x^i \sim \mu_i$. The probability of success is equal to the product of the individual success probabilities, i.e., $\prod_{i=1}^k \mathrm{Suc}_{0,\mu_i}(f)$.

For the induction step $T \Rightarrow T+1$, pick some $t_i \neq 0$ and consider two input distributions $\mu'_{i,0}$ and $\mu'_{i,1}$ obtained from μ_i by fixing the queried bit x^i_j (jth bit in the ith input). By the induction hypothesis, for each value $b \in \{0,1\}$, there is an optimal nonadaptive algorithm A_b that achieves the success probability $\mathrm{Suc}_{t_i-1,\mu'_{i,b}}(f) \cdot \prod_{j\neq i} \mathrm{Suc}_{t_j,\mu_j}(f)$. We construct a new algorithm A that calls A_b as a subroutine after it has queried x^i_j with b as an outcome. A is optimal and has success probability

$$\left(\sum_{b=0}^{1} \Pr_{\mu_i}[x^i_j = b] \cdot \mathrm{Suc}_{t_i-1,\mu'_{i,b}}(f)\right) \cdot \prod_{j\neq i} \mathrm{Suc}_{t_j,\mu_j}(f) = \prod_{i=1}^{k} \mathrm{Suc}_{t_i,\mu_i}(f).$$

Since we are dealing with nonadaptive algorithms here, symmetry reasons imply that if all k instances x^i are independent and identically distributed, then the optimal distribution of queries $t_1 + \cdots + t_k = kt$ is uniform, i.e., $t_i = t$. (Note that counterexamples to the analogous property for the general nonadaptive case, like those given by Shaltiel [Sha01], do not apply here.) In such a case, the algorithm achieves the success probability $\mathrm{Suc}_{t,\mu}(f)^k$. □

3.2. Adaptive algorithms. In this section we prove a similar statement also for adaptive algorithms.

Remark. The SDPT is not always true for adaptive algorithms. Following [Sha01], define $h(x) = x_1 \vee (x_2 \oplus \cdots \oplus x_n)$. Clearly $\mathrm{Suc}_{\frac{2}{3}n,\mu}(h) = 3/4$ for μ uniform. By a Chernoff bound, $\mathrm{Suc}_{\frac{2}{3}nk,\mu^k}(h^{(k)}) = 1 - 2^{-\Omega(k)}$, because approximately half of the blocks can be solved using just 1 query and the unused queries can be used to answer exactly also the other half of the blocks.

However, the SDPT is valid for $\mathrm{OR}_n^{(k)}$ under ν^k, where $\nu(0^n) = 1/2$ and $\nu(e_i) = 1/2n$ for e_i an n-bit string that contains a 1 only at the ith position. It is simple to prove that $\mathrm{Suc}_{\alpha n,\nu}(\mathrm{OR}_n) = \frac{\alpha+1}{2}$. Nonadaptive algorithms for $\mathrm{OR}_n^{(k)}$ under ν^k with αkn queries thus have $\sigma \le (\frac{\alpha+1}{2})^k = 2^{-\log(\frac{2}{\alpha+1})k}$. We can achieve any $\gamma < 1$ by choosing α sufficiently small. We prove that adaptive algorithms cannot be much better. Without loss of generality, we assume the following:

1. The adaptive algorithm is deterministic. By Yao's principle [Yao77], if there exists a randomized algorithm with success probability σ under some input distribution, then there exists a deterministic algorithm with success probability σ under that distribution.
2. Whenever the algorithm finds a 1 in some input block, it stops querying that block.
3. The algorithm spends the same number of queries in all blocks where it does not find a 1. This is optimal due to the symmetry between the blocks (we

omit the straightforward calculation that justifies this). It implies that the algorithm spends at least as many queries in each "empty" input block as in each "nonempty" block.

LEMMA 2. *If there is an adaptive T-query algorithm A computing $\mathrm{OR}_n^{(k)}$ under ν^k with success probability σ, then there is a nonadaptive $3T$-query algorithm A' computing $\mathrm{OR}_n^{(k)}$ with success probability $\sigma - 2^{-\Omega(k)}$.*

Proof. Let Z be the number of empty blocks. $\mathrm{E}[Z] = k/2$, and, by a Chernoff bound, $\delta = \Pr[Z < k/3] = 2^{-\Omega(k)}$. If $Z \geq k/3$, then A spends at most $3T/k$ queries in each empty block. Define nonadaptive A' that spends $3T/k$ queries in *each* block. Then A' queries all the positions that A queries and maybe some more. Compare the overall success probabilities of A and A':

$$\begin{aligned}\sigma_A &= \Pr[Z < k/3] \cdot \Pr[A \text{ succeeds} \mid Z < k/3] \\ &\quad + \Pr[Z \geq k/3] \cdot \Pr[A \text{ succeeds} \mid Z \geq k/3] \\ &\leq \delta \cdot 1 + \Pr[Z \geq k/3] \cdot \Pr[A' \text{ succeeds} \mid Z \geq k/3] \\ &\leq \delta + \sigma_{A'}.\end{aligned}$$

We conclude that $\sigma_{A'} \geq \sigma_A - \delta$. (*Remark.* By replacing the $k/3$-bound on Z by a βk-bound for some $\beta > 0$, we can obtain arbitrary $\gamma < 1$ in the exponent $\delta = 2^{-\gamma k}$, while the number of queries of A' becomes T/β.) □

Combining the two lemmas establishes the following theorem.

THEOREM 3 (SDPT for OR). *For every $0 < \gamma < 1$, there exists an $\alpha > 0$ such that every randomized algorithm for $\mathrm{OR}_n^{(k)}$ with $T \leq \alpha kn$ queries has success probability $\sigma \leq 2^{-\gamma k}$.*

3.3. A bound for the parity instead of the vector of results. Here we give an SDPT for the *parity* of k independent instances of OR_n. The parity is a Boolean variable; hence we can always guess it with probability at least $\frac{1}{2}$. However, we prove that the advantage (instead of the success probability) of our guess must be exponentially small.

Let X be a random bit with $\Pr[X = 1] = p$. We define the *advantage of X* by $\mathrm{Adv}(X) = |2p - 1|$. Note that a uniformly distributed random bit has advantage 0 and a bit known with certainty has advantage 1. It is well known that if $X_1, \ldots, X_k$ are independent random bits, then $\mathrm{Adv}(X_1 \oplus \cdots \oplus X_k) = \prod_{i=1}^k \mathrm{Adv}(X_i)$. Compare this with the fact that the probability of correctly guessing the complete vector $(X_1, \ldots, X_k)$ is the product of the individual probabilities.

We have proved a lower bound for the computation of $\mathrm{OR}_n^{(k)}$ (vector of ORs). By the same technique, replacing the success probability by the advantage in all claims and proofs, we can also prove a lower bound for the computation of $\mathrm{OR}_n^{\oplus k}$ (parity of ORs).

THEOREM 4 (SDPT for parity of ORs). *For every $0 < \gamma < 1$, there exists an $\alpha > 0$ such that every randomized algorithm for $\mathrm{OR}_n^{\oplus k}$ with $T \leq \alpha kn$ queries has advantage $\tau \leq 2^{-\gamma k}$.*

3.4. A bound for all functions. Here we show that the SDPT for OR actually implies a weaker direct product theorem for all functions. In this weaker version, the success probability of computing k instances still goes down exponentially with k, but we need to start from a polynomially smaller bound on the overall number of queries.

DEFINITION 5. *For $x \in \{0,1\}^n$ and $S \subseteq [n]$, we use x^S to denote the n-bit string obtained from x by flipping the bits in S. Consider a (possibly partial) function*

$f : \mathcal{D} \to Z$, with $\mathcal{D} \subseteq \{0,1\}^n$. The block sensitivity *$bs_x(f)$ of $x \in \mathcal{D}$ is the maximal b for which there are disjoint sets $S_1, \ldots, S_b$ such that $f(x) \neq f(x^{S_i})$. The* block sensitivity *of f is $\max_{x \in \mathcal{D}} bs_x(f)$.*

Block sensitivity is closely related to deterministic and bounded-error classical query complexity as shown in the following theorem.

THEOREM 6 (see [Nis91, BBC+01]). *$R_2(f) = \Omega(bs(f))$ for all f, and $D(f) \leq bs(f)^3$ for all total Boolean f.*

Nisan and Szegedy [NS94] showed how to embed a $bs(f)$-bit OR-function (with the promise that the input has weight ≤ 1) into f. Combined with our SDPT for OR, this implies the following direct product theorem for all functions in terms of their block sensitivity.

THEOREM 7. *For every $0 < \gamma < 1$, there exists an $\alpha > 0$ such that for every f, every classical algorithm for $f^{(k)}$ with $T \leq \alpha k bs(f)$ queries has success probability $\sigma \leq 2^{-\gamma k}$.*

This is optimal whenever $R_2(f) = \Theta(bs(f))$, which is the case for most functions. For total functions, the gap between $R_2(f)$ and $bs(f)$ is not more than cubic; hence, we have the following corollary.

COROLLARY 8. *For every $0 < \gamma < 1$, there exists an $\alpha > 0$ such that for every total Boolean f, every classical algorithm for $f^{(k)}$ with $T \leq \alpha k R_2(f)^{1/3}$ queries has success probability $\sigma \leq 2^{-\gamma k}$.*

4. Strong direct product theorem for quantum queries. In this section we prove an SDPT for quantum algorithms computing k independent instances of OR. Our proof relies on the polynomial method of [BBC+01].

4.1. Bounds on polynomials. We use three results about polynomials, also used in [BCWZ99]. The first is by Coppersmith and Rivlin [CR92, p. 980] and gives a general bound for polynomials bounded by 1 at integer points.

THEOREM 9 (see Coppersmith and Rivlin [CR92]). *Every polynomial p of degree $d \leq n$ that has absolute value*

$$|p(i)| \leq 1 \text{ for all integers } i \in [0, n]$$

satisfies

$$|p(x)| < ae^{bd^2/n} \text{ for all real } x \in [0, n],$$

where $a, b > 0$ are universal constants (no explicit values for a and b are given in [CR92]*).*

The other two results concern the Chebyshev polynomials T_d, defined by (see, e.g., [Riv90]):

$$T_d(x) = \frac{1}{2}\left(\left(x + \sqrt{x^2-1}\right)^d + \left(x - \sqrt{x^2-1}\right)^d\right).$$

T_d has degree d, and its absolute value $|T_d(x)|$ is bounded by 1 if $x \in [-1, 1]$. On the interval $[1, \infty)$, T_d exceeds all other polynomials with those two properties ([Riv90, p. 108] and [Pat92, Fact 2]).

THEOREM 10. *If q is a polynomial of degree d such that $|q(x)| \leq 1$ for all $x \in [-1, 1]$ then $|q(x)| \leq |T_d(x)|$ for all $x \geq 1$.*

Paturi [Pat92, before Fact 2] proved the following lemma.

LEMMA 11 (see Paturi [Pat92]). *$T_d(1 + \mu) \leq e^{2d\sqrt{2\mu+\mu^2}}$ for all $\mu \geq 0$.*

Proof. For $x = 1+\mu$, $T_d(x) \le (x+\sqrt{x^2-1})^d = (1+\mu+\sqrt{2\mu+\mu^2})^d \le (1+2\sqrt{2\mu+\mu^2})^d \le e^{2d\sqrt{2\mu+\mu^2}}$ (using that $1+z \le e^z$ for all real z). □

The following key lemma is the basis for all our direct product theorems.

LEMMA 12. *Suppose p is a degree-D polynomial such that for some $\delta \ge 0$*

$-\delta \le p(i) \le \delta$ *for all* $i \in \{0,\dots,k-1\}$,

$p(k) = \sigma$,

$p(i) \in [-\delta, 1+\delta]$ *for all* $i \in \{0,\dots,N\}$.

Then for every integer $1 \le C < N-k$ and $\mu = 2C/(N-k-C)$ we have

$$\sigma \le a\left(1+\delta+\frac{\delta(2N)^k}{(k-1)!}\right) \cdot \exp\left(\frac{b(D-k)^2}{(N-k-C)} + 2(D-k)\sqrt{2\mu+\mu^2} - k\ln(C/k)\right) + \delta k 2^{k-1},$$

where a, b are the constants given by Theorem 9.

Before establishing this gruesome bound, let us reassure the reader by noting that we will apply this lemma with δ either 0 or negligibly small, $D = \alpha\sqrt{kN}$ for sufficiently small α, and $C = ke^{\gamma+1}$, giving

$$\sigma \le \exp\Big((b\alpha^2 + 4\alpha e^{\gamma/2+1/2} - 1 - \gamma)k\Big) \le e^{-\gamma k} \le 2^{-\gamma k}.$$

Proof of Lemma 12. Divide p with remainder by $\prod_{j=0}^{k-1}(x-j)$ to obtain

$$p(x) = q(x)\prod_{j=0}^{k-1}(x-j) + r(x),$$

where $d = \deg(q) = D-k$ and $\deg(r) \le k-1$. We know that $r(x) = p(x) \in [-\delta,\delta]$ for all $x \in \{0,\dots,k-1\}$. Decompose r as a linear combination of polynomials e_i, where $e_i(i) = 1$ and $e_i(x) = 0$ for $x \in \{0,\dots,k-1\} - \{i\}$:

$$r(x) = \sum_{i=0}^{k-1} p(i)e_i(x) = \sum_{i=0}^{k-1} p(i) \prod_{\substack{j=0\\ j\ne i}}^{k-1} \frac{x-j}{i-j}.$$

We bound the values of r for all real $x \in [0,N]$ by

$$|r(x)| \le \sum_{i=0}^{k-1} \frac{|p(i)|}{i!(k-1-i)!} \prod_{\substack{j=0\\ j\ne i}}^{k-1} |x-j|$$

$$\le \frac{\delta}{(k-1)!}\sum_{i=0}^{k-1}\binom{k-1}{i}N^k \le \frac{\delta(2N)^k}{(k-1)!},$$

$$|r(k)| \le \delta k 2^{k-1}.$$

This implies the following about the values of the polynomial q:

$$|q(k)| \ge (\sigma - \delta k 2^{k-1})/k!$$

$$|q(i)| \le \frac{(i-k)!}{i!}\left(1+\delta+\frac{\delta(2N)^k}{(k-1)!}\right) \quad \text{for } i \in \{k,\dots,N\}.$$

In particular,

$$|q(i)| \le C^{-k}\left(1+\delta+\frac{\delta(2N)^k}{(k-1)!}\right) = A \quad \text{for } i \in \{k+C,\dots,N\}.$$

Theorem 9 implies that there are constants $a, b > 0$ such that

$$|q(x)| \le A \cdot ae^{bd^2/(N-k-C)} = B \quad \text{for all real } x \in [k+C, N].$$

We now divide q by B to normalize it and rescale the interval $[k+C, N]$ to $[1, -1]$ to get a degree-d polynomial t satisfying

$$|t(x)| \le 1 \quad \text{for all } x \in [-1,1],$$
$$t(1+\mu) = q(k)/B \quad \text{for } \mu = 2C/(N-k-C).$$

Since t cannot grow faster than the degree-d Chebyshev polynomial, we get

$$t(1+\mu) \le T_d(1+\mu) \le e^{2d\sqrt{2\mu+\mu^2}}.$$

Combining our upper and lower bounds on $t(1+\mu)$, we obtain

$$\frac{(\sigma-\delta k 2^{k-1})/k!}{C^{-k}\left(1+\delta+(\delta(2N)^k/(k-1)!)\right)ae^{bd^2/(N-k-C)}} \le e^{2d\sqrt{2\mu+\mu^2}}.$$

Rearranging gives the bound. ∎

4.2. Consequences for quantum algorithms. The previous result about polynomials implies a strong tradeoff between queries and success probability for quantum algorithms that have to find k 1's in an N-bit input. A *k-threshold algorithm with success probability σ* is an algorithm on N-bit input x that outputs 0 with certainty if $|x| < k$, and outputs 1 with probability at least σ if $|x| = k$.

THEOREM 13. *For every $\gamma > 0$, there exists an $\alpha > 0$ such that every quantum k-threshold algorithm with $T \le \alpha\sqrt{kN}$ queries has success probability $\sigma \le 2^{-\gamma k}$.*

Proof. Fix $\gamma > 0$ and consider a T-query k-threshold algorithm. By [BBC+01], its acceptance probability is an N-variate polynomial of degree $D \le 2T \le 2\alpha\sqrt{kN}$ and can be symmetrized to a single-variate polynomial p with the properties

$p(i) = 0$ if $i \in \{0,\dots,k-1\}$,
$p(k) \ge \sigma$,
$p(i) \in [0,1]$ for all $i \in \{0,\dots,N\}$,

Choosing $\alpha > 0$ sufficiently small and $\delta = 0$, the result follows from Lemma 12. ∎

This implies an SDPT for k instances of the n-bit search problem. For each such instance, the goal is to find the index of a 1-bit among the n input bits of the instance (or to report that none exists).

THEOREM 14 (SQDPT for Search). *For every $\gamma > 0$, there exists an $\alpha > 0$ such that every quantum algorithm for* $\text{Search}_n^{(k)}$ *with $T \le \alpha k\sqrt{n}$ queries has success probability $\sigma \le 2^{-\gamma k}$.*

Proof. Set $N = kn$, and fix a $\gamma > 0$ and a T-query algorithm A for $\text{Search}_n^{(k)}$ with success probability σ. Now consider the following algorithm that acts on an N-bit input x:

1. Apply a random permutation π to x.
2. Run A on $\pi(x)$.

3. Query each of the k positions that A outputs, and return 1 if and only if at least $k/2$ of those bits are 1.

This uses $T + k$ queries. We will show that it is a $k/2$-threshold algorithm. First, if $|x| < k/2$, it always outputs 0. Second, consider the case $|x| = k/2$. The probability that π puts all $k/2$ 1's in distinct n-bit blocks is

$$\frac{N}{N} \cdot \frac{N-n}{N-1} \cdots \frac{N-\frac{k}{2}n}{N-\frac{k}{2}} \geq \left(\frac{N-\frac{k}{2}n}{N}\right)^{k/2} = 2^{-k/2}.$$

Hence our algorithm outputs 1 with probability at least $\sigma 2^{-k/2}$. Choosing α sufficiently small, the previous theorem implies $\sigma 2^{-k/2} \leq 2^{-(\gamma+1/2)k}$; hence $\sigma \leq 2^{-\gamma k}$. □

Our bounds are quite precise for $\alpha \ll 1$. We can choose $\gamma = 2\ln(1/\alpha) - \mathrm{O}(1)$ and ignore some lower-order terms to get roughly $\sigma \leq \alpha^{2k}$. On the other hand, it is known that Grover's search algorithm with $\alpha\sqrt{n}$ queries on an n-bit input has success probability roughly α^2 [BBHT98]. Doing such a search on all k instances gives overall success probability α^{2k}.

THEOREM 15 (SQDPT for OR). *There exist $\alpha, \gamma > 0$ such that every quantum algorithm for $\mathrm{OR}_n^{(k)}$ with $T \leq \alpha k\sqrt{n}$ queries has success probability $\sigma \leq 2^{-\gamma k}$.*

Proof. An algorithm A for $\mathrm{OR}_n^{(k)}$ with success probability σ can be used to build an algorithm A' for $\mathrm{Search}_n^{(k)}$ with slightly worse success probability:

1. Run A on the original input and remember which blocks contain a 1.
2. Run simultaneously (at most k) binary searches on the nonzero blocks. Iterate this $s = 2\log(1/\alpha)$ times. Each iteration is computed by running A on the parts of the blocks that are known to contain a 1, halving the remaining instance size each time.
3. Run the exact version of Grover's algorithm on each of the remaining parts of the instances to look for a 1 there (each remaining part has size $n/2^s$).

This new algorithm A' uses $(s+1)T + \frac{\pi}{4}k\sqrt{n/2^s} = \mathrm{O}(\alpha \log(1/\alpha)k\sqrt{n})$ queries. With probability at least σ^{s+1}, A succeeds in all iterations, in which case A' solves $\mathrm{Search}_n^{(k)}$. By the previous theorem, for every $\gamma' > 0$ of our choice we can choose $\alpha > 0$ such that

$$\sigma^{s+1} \leq 2^{-\gamma' k},$$

which implies the theorem with $\gamma = \gamma'/(s+1)$. □

Choosing our parameters carefully, we can actually show that for every $\gamma < 1$ there is an $\alpha > 0$ such that $\alpha k\sqrt{n}$ queries give success probability $\sigma \leq 2^{-\gamma k}$. Clearly, $\sigma = 2^{-k}$ is achievable without any queries by random guessing.

4.3. A bound for all functions. As in section 3.4, we can extend the SDPT for OR to a slightly weaker theorem for all total functions. Block sensitivity is closely related to bounded-error quantum query complexity as shown in the following theorem.

THEOREM 16 (see [BBC+01]). *$Q_2(f) = \Omega(\sqrt{bs(f)})$ for all f, and $D(f) \leq bs(f)^3$ for all total Boolean f.*

By embedding an OR of size $bs(f)$ in f, we obtain the following theorem.

THEOREM 17. *There exist $\alpha, \gamma > 0$ such that for every f, every quantum algorithm for $f^{(k)}$ with $T \leq \alpha k\sqrt{bs(f)}$ queries has success probability $\sigma \leq 2^{-\gamma k}$.*

This is close to optimal whenever $Q_2(f) = \Theta(\sqrt{bs(f)})$. For total functions, the gap between $Q_2(f)$ and $\sqrt{bs(f)}$ is no more than a 6th power; hence the following corollary holds.

COROLLARY 18. *There exist $\alpha, \gamma > 0$ such that for every total Boolean f, every quantum algorithm for $f^{(k)}$ with $T \leq \alpha k Q_2(f)^{1/6}$ queries has success probability $\sigma \leq 2^{-\gamma k}$.*

5. Strong direct product theorem for quantum communication. In this section we establish an SDPT for quantum communication complexity, specifically for protocols that compute k independent instances of the disjointness problem. Our proof relies crucially on the beautiful technique that Razborov introduced to establish a lower bound on the quantum communication complexity of (one instance of) disjointness [Raz03]. It allows us to translate a quantum communication protocol to a single-variate polynomial that represents, roughly speaking, the protocol's acceptance probability as a function of the size of the intersection of x and y. Once we have this polynomial, the results from section 4.1 suffice to establish an SDPT.

5.1. Razborov's technique. Razborov's technique relies on the following linear algebraic notions. The *operator norm* $\| A \|$ of a matrix A is its largest singular value σ_1. The *trace inner product* (also known as Hilbert–Schmidt inner product) between A and B is $\langle A, B \rangle = \mathrm{Tr}(A^* B)$. The *trace norm* is $\| A \|_{tr} = \max\{|\langle A, B \rangle| : \| B \| = 1\} = \sum_i \sigma_i$, the sum of all singular values of A. The *Frobenius norm* is $\| A \|_F = \sqrt{\sum_{ij} |A_{ij}|^2} = \sqrt{\sum_i \sigma_i^2}$. The following lemma is implicit in Razborov's paper.

LEMMA 19. *Consider a Q-qubit quantum communication protocol on N-bit inputs x and y, with acceptance probabilities denoted by $P(x, y)$. Define*

$$P(i) = \mathrm{E}_{|x|=|y|=N/4, |x \wedge y|=i|}[P(x, y)],$$

where the expectation is taken uniformly over all x, y that each have weight $N/4$ and that have intersection i. For every $d \leq N/4$ there exists a degree-d polynomial q such that $|P(i) - q(i)| \leq 2^{-d/4+2Q}$ for all $i \in \{0, \ldots, N/8\}$.

Proof. We only consider the $\mathcal{N} = \binom{N}{N/4}$ strings of weight $N/4$. Let P denote the $\mathcal{N} \times \mathcal{N}$ matrix of the acceptance probabilities on these inputs. We know from Yao and Kremer [Yao93, Kre95] that we can decompose P as a matrix product $P = AB$, where A is an $\mathcal{N} \times 2^{2Q-2}$ matrix with each entry at most 1 in absolute value, and similarly for B. Note that $\| A \|_F, \| B \|_F \leq \sqrt{\mathcal{N} 2^{2Q-2}}$. Using Hölder's inequality we have

$$\| P \|_{tr} \leq \| A \|_F \cdot \| B \|_F \leq \mathcal{N} 2^{2Q-2}.$$

Let μ_i denote the $\mathcal{N} \times \mathcal{N}$ matrix corresponding to the uniform probability distribution on $\{(x, y) : |x \wedge y| = i\}$. These "combinatorial matrices" have been well studied [Knu03]. Note that $\langle P, \mu_i \rangle$ is the expected acceptance probability $P(i)$ of the protocol under that distribution. One can show that the different μ_i commute; thus they have the same eigenspaces $E_0, \ldots, E_{N/4}$ and can be simultaneously diagonalized by some orthogonal matrix U. For $t \in \{0, \ldots, N/4\}$, let $(UPU^T)_t$ denote the block of UPU^T corresponding to E_t, and let $a_t = \mathrm{Tr}((UPU^T)_t)$ be its trace. Then we have

$$\sum_{t=0}^{N/4} |a_t| \leq \sum_{j=1}^{\mathcal{N}} \left|(UPU^T)_{jj}\right| \leq \| UPU^T \|_{tr} = \| P \|_{tr} \leq \mathcal{N} 2^{2Q-2},$$

where the second inequality is a property of the trace norm.

Let λ_{it} be the eigenvalue of μ_i in eigenspace E_t. It is known [Raz03, section 5.3] that λ_{it} is a degree-t polynomial in i, and that $|\lambda_{it}| \leq 2^{-t/4}/\mathcal{N}$ for $i \leq N/8$

(the factor 1/4 in the exponent is implicit in Razborov's paper). Consider the high-degree polynomial p defined by

$$p(i) = \sum_{t=0}^{N/4} a_t \lambda_{it}.$$

This satisfies

$$p(i) = \sum_{t=0}^{N/4} \mathrm{Tr}((UPU^T)_t)\lambda_{it} = \langle UPU^T, U\mu_i U^T\rangle = \langle P, \mu_i\rangle = P(i).$$

Let q be the degree-d polynomial obtained by removing the high-degree parts of p:

$$q(i) = \sum_{t=0}^{d} a_t \lambda_{it}.$$

Then P and q are close on all integers i between 0 and $N/8$:

$$|P(i) - q(i)| = |p(i) - q(i)| = \left| \sum_{t=d+1}^{N/4} a_t \lambda_{it} \right| \leq \frac{2^{-d/4}}{\mathcal{N}} \sum_{t=0}^{N/4} |a_t| \leq 2^{-d/4+2Q}. \qquad \square$$

5.2. Consequences for quantum protocols. Combining Razborov's technique with our polynomial bounds, we can prove the following theorem.

THEOREM 20 (SQDPT for disjointness). *There exist $\alpha, \gamma > 0$ such that every quantum protocol for $\mathrm{DISJ}_n^{(k)}$ with $Q \leq \alpha k\sqrt{n}$ qubits of communication has success probability $p \leq 2^{-\gamma k}$.*

Proof sketch. By doing the same trick with $s = 2\log(1/\alpha)$ rounds of binary search as for Theorem 15, we can tweak a protocol for $\mathrm{DISJ}_n^{(k)}$ to a protocol that satisfies, with $P(i)$ defined as in Lemma 19, $N = kn$ and $\sigma = p^{s+1}$:

$P(i) = 0$ if $i \in \{0, \ldots, k-1\}$,
$P(k) \geq \sigma$,
$P(i) \in [0,1]$ for all $i \in \{0, \ldots, N\}$

(a subtlety: instead of exact Grover we use an exact version of the $\mathrm{O}(\sqrt{n})$-qubit disjointness protocol of [AA03]; the [BCW98]-protocol would lose a $\log n$-factor). Lemma 19, using $d = 12Q$, then gives a degree-d polynomial q that differs from P by at most $\delta \leq 2^{-Q}$ on all $i \in \{0, \ldots, N/8\}$. This δ is sufficiently small to apply Lemma 12, which in turn upper bounds σ and hence p. $\square$

This technique also gives SDPTs for symmetric predicates other than DISJ_n. As mentioned in the introduction, the same bound was obtained independently by Beame et al. [BPSW05, Corollary 9] for classical protocols.

6. Time-space tradeoff for quantum sorting. We will now use our SDPT to get near-optimal time-space tradeoffs for quantum circuits for sorting. In our model, the numbers $a_1, \ldots, a_N$ that we want to sort can be accessed by means of queries, and the number of queries lower bounds the actual time taken by the circuit. The circuit has N output gates and in the course of its computation outputs the N numbers in sorted (say, descending) order, with success probability at least 2/3.

THEOREM 21. *Every bounded-error quantum circuit for sorting N numbers that uses T queries and S qubits of workspace satisfies $T^2 S = \Omega(N^3)$.*

Proof. We "slice" the circuit along the time-axis into $L = T/\alpha\sqrt{SN}$ slices, each containing $T/L = \alpha\sqrt{SN}$ queries. Each such slice has a number of output gates. Consider any slice. Suppose it contains output gates $i, i+1, \ldots, i+k-1$ for $i \leq N/2$, so that it is supposed to output the ith up to $(i+k-1)$th largest elements of its input. We want to show that $k = \mathrm{O}(S)$. If $k \leq S$, then we are done, so assume $k > S$. We can use the slice as a k-threshold algorithm on $N/2$ bits, as follows. For an $N/2$-bit input x, construct a sorting input by taking $i-1$ copies of the number 2, the $N/2$ bits in x, and $N/2 - i + 1$ copies of the number 0, and append their position behind the numbers.

Consider the behavior of the sorting circuit on this input. The first part of the circuit has to output the $i-1$ largest numbers, which all start with 2. We condition on the event that the circuit succeeds in this. It then passes on an S-qubit state (possibly mixed) as the starting state of the particular slice we are considering. This slice then outputs the k largest numbers in x with probability at least $2/3$. Now, consider an algorithm that runs just this slice, starting with the completely mixed state on S-qubits, and that outputs 1 if it finds k numbers starting with 1, and outputs 0 otherwise. If $|x| < k$, this new algorithm always outputs 0 (note that it can verify finding a 1 since its position is appended), but if $|x| = k$, then it outputs 1 with probability at least $\sigma \geq \frac{2}{3} \cdot 2^{-S}$, because the completely mixed state has "overlap" 2^{-S} with the "good" S-qubit state that would have been the starting state of the slice in the run of the sorting circuit. On the other hand, the slice has only $\alpha\sqrt{SN} < \alpha\sqrt{kN}$ queries, so by choosing α sufficiently small, Theorem 13 implies $\sigma \leq 2^{-\Omega(k)}$. Combining our upper and lower bounds on σ gives $k = \mathrm{O}(S)$. Thus we need $L = \Omega(N/S)$ slices, so $T = L\alpha\sqrt{SN} = \Omega(N^{3/2}/\sqrt{S})$. □

As mentioned, our tradeoff is achievable up to polylog factors [Kla03]. Interestingly, the near-optimal algorithm uses only a polylogarithmic number of qubits and otherwise just classical memory. For simplicity we have shown the lower bound for the case when the outputs have to be made in their natural ordering only, but we can show, using a slightly different proof, the same lower bound for any ordering of the outputs that does not depend on the input.

7. Time-space tradeoffs for Boolean matrix products. First we show a lower bound on the time-space tradeoff for Boolean matrix-vector multiplication on *classical* machines.

THEOREM 22. *There is an $N \times N$ matrix A such that every classical bounded-error circuit that computes the Boolean matrix-vector product Ab with T queries and space $S = \mathrm{o}(N/\log N)$ satisfies $TS = \Omega(N^2)$.*

The bound is tight if T measures queries to the input.

Proof. Fix $k = \mathrm{O}(S)$ large enough. First we have to find a hard matrix A. We pick A randomly by setting $N/(2k)$ random positions in each row to 1. We want to show that with positive probability for all sets of k rows $A[i_1], \ldots, A[i_k]$ many of the rows $A[i_j]$ contain at least $N/(6k)$ 1's that are not 1's in any of the $k-1$ other rows.

This probability can be bounded as follows. We will treat the rows as subsets of $\{1, \ldots, N\}$. A row $A[j]$ is called *bad* with respect to $k-1$ other rows $A[i_1], \ldots, A[i_{k-1}]$ if $|A[j] - \cup_\ell A[i_\ell]| \leq N/(6k)$. For fixed $i_1, \ldots, i_{k-1}$, the probability that some $A[j]$ is bad with respect to the $k-1$ other rows is at most $e^{-\Omega(N/k)}$ by the Chernoff bound and the fact that k rows can together contain at most $N/2$ elements. Since $k = \mathrm{o}(N/\log N)$ we may assume this probability is at most $1/N^{10}$.

Now fix any set $I = \{i_1, \ldots, i_k\}$. The probability that for $j \in I$ it holds that $A[j]$ is bad with respect to the other rows is at most $1/N^{10}$, and this also holds if we

condition on the event that some other rows are bad, since this condition makes it only less probable that another row is also bad. So for any fixed $J \subset I$ of size $k/2$ the probability that all rows in J are bad is at most N^{-5k}, and the probability that there exists such J is at most

$$\binom{k}{k/2} N^{-5k}.$$

Furthermore, the probability that there is a set I of k rows for which $k/2$ are bad is at most

$$\binom{N}{k}\binom{k}{k/2} N^{-5k} < 1.$$

So there is an A as required and we may fix such an A.

Now suppose we are given a circuit with space S that computes the Boolean product between the rows of A and b in some order. We again proceed by "slicing" the circuit into $L = T/\alpha N$ slices, each containing $T/L = \alpha N$ queries. Each such slice has a number of output gates. Consider any slice. Suppose it contains output gates $i_1 < \cdots < i_k \leq N/2$; thus it is supposed to output $\vee_{\ell=1}^{N} (A[i_j, \ell] \wedge b_\ell)$ for all i_j with $1 \leq j \leq k$.

Such a slice starts on a classical value of the "memory" of the circuit, which is in general a probability distribution on S bits (if the circuit is randomized). We replace this probability distribution by the uniform distribution on the possible values of S bits. If the original circuit succeeds in computing the function correctly with probability at least $1/2$, then so does the circuit slice with its outputs, and replacing the initial value of the memory by a uniformly random value decreases the success probability to no less than $(1/2) \cdot 1/2^S$.

If we now show that any classical circuit with αN queries that produces the outputs $i_1, \ldots, i_k$ can succeed only with exponentially small probability in k, we get that $k = \mathrm{O}(S)$, and hence $(T/\alpha N) \cdot \mathrm{O}(S) \geq N$, which gives the claimed lower bound for the time-space tradeoff.

Each set of k outputs corresponds to k rows of A, which contain $N/(2k)$ 1's each. Thanks to the construction of A, there are $k/2$ rows among these, such that $N/(6k)$ of the 1's in each such row are in a position where none of the other contains a 1. So we get $k/2$ sets of $N/(6k)$ positions that are unique to each of the $k/2$ rows. The inputs for b will be restricted to contain 1's only at these positions, and so the algorithm naturally has to solve $k/2$ independent OR problems on $n = N/(6k)$ bits each. By Theorem 3, this is only possible with αN queries if the success probability is exponentially small in k. □

An absolutely analogous construction can be done in the quantum case. Using circuit slices of length $\alpha\sqrt{NS}$, we can prove the following theorem.

THEOREM 23. *There is an $N \times N$ matrix A such that every quantum bounded-error circuit that computes the Boolean matrix-vector product Ab with T queries and space $S = \mathrm{o}(N/\log N)$ satisfies $T^2 S = \Omega\big(N^3\big)$.*

Note that this is tight within a logarithmic factor (needed to improve the success probability of Grover's search).

THEOREM 24. *Every classical bounded-error circuit that computes the $N \times N$ Boolean matrix product AB with T queries and space S satisfies $TS = \Omega\big(N^3\big)$.*

While this is near-optimal for small S, it is probably not tight for large S, a likely tight tradeoff being $T^2 S = \Omega\big(N^6\big)$. It is also no improvement compared to Abrahamson's average-case bounds [Abr90].

Proof. Suppose that $S = o(N)$; otherwise the bound is trivial, since time N^2 is always needed. We can proceed in a manner similar to that of the proof of Theorem 22. We slice the circuit so that each slice has only αN queries. Suppose a slice makes k outputs. We are going to restrict the inputs to get a direct product problem with k instances of size N/k each; hence a slice with αN queries has exponentially small success probability in k and thus can produce only $\mathrm{O}(S)$ outputs. Since the overall number of outputs is N^2, we get the tradeoff $TS = \Omega\big(N^3\big)$.

Suppose a circuit slice makes k outputs, where an output labeled (i, j) needs to produce the vector product of the ith row $A[i]$ of A and the jth column $B[j]$ of B. We may partition the set $\{1, \ldots, N\}$ into k mutually disjoint subsets $U(i, j)$ of size N/k, each associated to an output (i, j).

Assume that there are ℓ outputs $(i, j_1), \ldots, (i, j_\ell)$ involving $A[i]$. Each such output is associated to a subset $U(i, j_t)$, and we set $A[i]$ to zero on all positions that are not in any of these subsets, and to 1 on all positions that are in one of these subsets. When there are ℓ outputs $(i_1, j), \ldots, (i_\ell, j)$ involving $B[j]$, we set $B[j]$ to zero on all positions that are not in any of the corresponding subsets, and allow the inputs to be arbitrary on the other positions.

If the circuit computes on these restricted inputs, it actually has to compute k instances of OR of size $n = N/k$ in B, for it is true that $A[i]$ and $B[j]$ contain a single block of size N/k in which $A[i]$ contains only 1's, and $B[j]$ "free" input bits, if and only if (i, j) is one of the k outputs. Hence the SDPT is applicable. □

The application to the quantum case is analogous.

THEOREM 25. *Every quantum bounded-error circuit that computes the $N \times N$ Boolean matrix product AB with T queries and space S satisfies $T^2S = \Omega\big(N^5\big)$.*

If $S = \mathrm{O}(\log N)$, then N^2 applications of Grover can compute AB with $T = \mathrm{O}\big(N^{2.5} \log N\big)$. Hence our tradeoff is near-optimal for small S. We do not know whether it is optimal for large S.

8. Quantum communication-space tradeoffs for matrix products. In this section we use the strong direct product result for quantum communication (Theorem 20) to prove communication-space tradeoffs. We later show that these are close to optimal.

THEOREM 26. *Every quantum bounded-error protocol in which Alice and Bob have bounded space S and that computes the N-dimensional Boolean matrix-vector product satisfies $C^2S = \Omega\big(N^3\big)$.*

Proof. In a protocol, Alice receives a matrix A, and Bob receives a vector b as inputs. Given a circuit that multiplies these with communication C and space S, we again proceed to slice it. This time, however, a slice contains a limited amount of communication. Recall that in communicating quantum circuits the communication corresponds to wires carrying qubits that cross between Alice's and Bob's circuits. Hence we may cut the circuit after $\alpha\sqrt{NS}$ qubits have been communicated and so on. Overall there are $C/\alpha\sqrt{NS}$ circuit slices. Each starts with an initial state that may be replaced by the completely mixed state at the cost of decreasing the success probability to $(1/2)\cdot 1/2^S$. We want to employ the direct product theorem for quantum communication complexity to show that a protocol with the given communication has success probability at most exponentially small in the number of outputs it produces and thus that a slice can produce at most $\mathrm{O}(S)$ outputs. Combining these bounds with the fact that N outputs have to be produced gives the tradeoff.

To use the direct product theorem we restrict the inputs in the following way: Suppose a protocol makes k outputs. We partition the vector b into k blocks of size

N/k, and each block is assigned to one of the k rows of A for which an output is made. This row is made to contain zeros outside of the positions belonging to its block, and hence we arrive at a problem where disjointness has to be computed on k instances of size N/k. With communication $\alpha\sqrt{kN}$, the success probability must be exponentially small in k due to Theorem 20. Hence $k = \mathrm{O}(S)$ is an upper bound on the number of outputs produced. □

THEOREM 27. *Every quantum bounded-error protocol in which Alice and Bob have bounded space S and that computes the N-dimensional Boolean matrix product satisfies $C^2S = \Omega(N^5)$.*

Proof. The proof uses the same slicing approach as in the other tradeoff results. Note that we can assume that $S = \mathrm{o}(N)$, since otherwise the bound is trivial. Each slice contains communication $\alpha\sqrt{NS}$, and as before a direct product result showing that k outputs can be computed only with success probability exponentially small in k leads to the conclusion that a slice can compute only $\mathrm{O}(S)$ outputs. Therefore $(C/\alpha\sqrt{NS}) \cdot \mathrm{O}(S) \geq N^2$, and we are done.

Consider a protocol with $\alpha\sqrt{NS}$ qubits of communication. We partition the universe $\{1, \ldots, N\}$ of the disjointness problems to be computed into k mutually disjoint subsets $U(i,j)$ of size N/k, each associated to an output (i,j), which in turn corresponds to a row/column pair $A[i]$, $B[j]$ in the input matrices A and B. Assume that there are ℓ outputs $(i,j_1), \ldots, (i,j_\ell)$ involving $A[i]$. Each output is associated to a subset of the universe $U(i,j_t)$, and we set $A[i]$ to zero on all positions that are not in one of these subsets. Then we proceed analogously with the columns of B.

If the protocol computes on these restricted inputs, it has to solve k instances of disjointness of size $n = N/k$ each, since $A[i]$ and $B[j]$ contain a single block of size N/k in which both are not set to 0 if and only if (i,j) is one of the k outputs. Hence Theorem 20 is applicable. □

We now want to show that these tradeoffs are not too far from optimal.

THEOREM 28. *There is a quantum bounded-error protocol with space S that computes the Boolean product between an $N \times N$ matrix and an N-dimensional vector within communication $C = \mathrm{O}((N^{3/2} \log^2 N)/\sqrt{S})$.*

There is a quantum bounded-error protocol with space S that computes the Boolean product between two $N \times N$ matrices within communication $C = \mathrm{O}((N^{5/2} \log^2 N)/\sqrt{S})$.

Proof. We begin by showing a protocol for the following scenario: Alice gets S N-bit vectors $x_1, \ldots, x_S$, Bob gets an N-bit vector y, and they want to compute the S Boolean inner products between these vectors. The protocol uses space $\mathrm{O}(S)$.

In the following, we interpret Boolean vectors as sets. The main idea is that Alice can use the union z of the x_i and then Alice and Bob can find an element in the intersection of z and y using the protocol for the disjointness problem described in [BCW98]. Alice then marks all x_i that contain this element and removes them from z.

A problem with this approach is that Alice cannot store z explicitly, since it might contain many more than S elements. Alice may, however, store in an array of length S the indices of those sets x_i for which an element in the intersection of x_i and y has already been found. This array and the input given as an oracle work as an implicit representation of z.

Now suppose at some point during the protocol that the intersection of z and y has size k. Then Alice and Bob can find one element in this intersection within $\mathrm{O}(\sqrt{N/k})$ rounds of communication, in each of which $\mathrm{O}(\log N)$ qubits are exchanged. Furthermore, in $\mathrm{O}(\sqrt{Nk})$ rounds all elements in the intersection can be found. So if

$k \leq S$, then all elements are found within communication $O(\sqrt{NS}\log N)$, and the problem can be solved completely. On the other hand, if $k \geq S$, finding one element costs $O(\sqrt{N/S}\log N)$, but finding such an element removes at least one x_i from z, and hence this has to be done at most S times, giving the same overall communication bound.

It is not hard to see that this process can be implemented with space $O(S)$. The protocol from [BCW98] is a distributed Grover's search that uses only space $O(\log N)$. Bob can work as in this protocol. For each search, Alice has to start with a superposition over all indices in z. This superposition can be computed from her oracle and her array. In each step she has to apply the Grover iteration. This can also be implemented from these two resources.

To get a protocol for the matrix-vector product, the above procedure is repeated N/S times; hence the communication is $O((N/S)\cdot\sqrt{NS}\log^2 N)$, where one logarithmic factor stems from improving success probability to $1/\mathrm{poly}(N)$.

For the product of two matrices, the matrix-vector protocol may be repeated N times. □

These near-optimal protocols use only $O(\log N)$ qubits, and apart from that S bits of classical memory.

9. Open problems. We mention some open problems. The first is to determine tight time-space tradeoffs for the Boolean matrix product on both classical and quantum computers. Second, regarding communication-space tradeoffs for the Boolean matrix-vector and matrix product, we did not prove any classical bounds that were better than our quantum bounds. Klauck [Kla04] recently proved classical tradeoffs $CS^2 = \Omega(N^3)$ and $CS^2 = \Omega(N^2)$ for the Boolean matrix product and matrix-vector product, respectively, by means of a *weak* direct product theorem for disjointness. A classical *strong* direct product theorem for disjointness (with communication bound αkn instead of our current $\alpha k\sqrt{n}$) would imply optimal tradeoffs, but we do not know how to prove this at the moment. Third, we would like to know whether an SDPT holds in the query and communication setting for all Boolean functions if we consider worst-case error probability (Shaltiel [Sha01] disproved this for *average-case* error probability). Finally, it would be interesting to get any lower bounds on time-space or communication-space tradeoffs for decision problems in the quantum case, for example, for element distinctness [BDH+01, Amb04] or the verification of matrix multiplication [BŠ06].

Acknowledgments. Many thanks to Scott Aaronson for email discussions about the evolving results in his work [Aar04] that motivated some of our proofs, Harry Buhrman for useful discussions, Paul Beame for communication about [BPSW05], and the anonymous referees for comments that improved the presentation of the paper.

REFERENCES

[AA03] S. Aaronson and A. Ambainis, *Quantum search of spatial regions*, in Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, 2003, pp. 200–209; available online from http://www.arxiv.org/abs/quant-ph/0303041.

[Aar04] S. Aaronson, *Limitations of quantum advice and one-way communication*, in Proceedings of the 19th Annual IEEE Conference on Computational Complexity, 2004, pp. 320–332; available online from http://www.arxiv.org/abs/quant-ph/0402095.

[Abr90] K. Abrahamson, *A time-space tradeoff for Boolean matrix multiplication*, in Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science, 1990, pp. 412–419; available online from http://www.arxiv.org/abs/quant-ph/0303041.

[Amb04] A. Ambainis, *Quantum walk algorithm for element distinctness*, in Proceedings of the 45st Annual IEEE Symposium on Foundations of Computer Science, 2004, pp. 22–31; available online from http://www.arxiv.org/abs/quant-ph/0311001.

[BBC+01] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf, *Quantum lower bounds by polynomials*, J. ACM, 48 (2001), pp. 778–797.

[BBHT98] M. Boyer, G. Brassard, P. Høyer, and A. Tapp, *Tight bounds on quantum searching*, Fortschr. Phys., 46 (1998), pp. 493–505.

[BCW98] H. Buhrman, R. Cleve, and A. Wigderson, *Quantum versus classical communication and computation*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, 1998, pp. 63–68; available online from http://www.arxiv.org/abs/quant-ph/9802040.

[BCWZ99] H. Buhrman, R. Cleve, R. de Wolf, and Ch. Zalka, *Bounds for small-error and zero-error quantum algorithms*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, 1999, pp. 358–368; available online from http://www.arxiv.org/abs/cs.CC/9904019.

[BDH+01] H. Buhrman, Ch. Dürr, M. Heiligman, P. Høyer, F. Magniez, M. Santha, and R. de Wolf, *Quantum algorithms for element distinctness*, in Proceedings of the 16th Annual IEEE Conference on Computational Complexity, 2001, pp. 131–137; available online from http://www.arxiv.org/abs/quant-ph/0007016.

[Bea91] P. Beame, *A general sequential time-space tradeoff for finding unique elements*, SIAM J. Comput., 20 (1991), pp. 270–277.

[BFS86] L. Babai, P. Frankl, and J. Simon, *Complexity classes in communication complexity theory*, in Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 337–347.

[BHMT02] G. Brassard, P. Høyer, M. Mosca, and A. Tapp, *Quantum amplitude amplification and estimation*, in Quantum Computation and Quantum Information: A Millennium Volume, Contemp. Math. 305, AMS, Providence, RI, 2002, pp. 53–74; available online from http://www.arxiv.org/abs/quant-ph/0005055.

[BJKS02a] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar, *An information statistics approach to data stream and communication complexity*, in Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002, pp. 209–218.

[BJKS02b] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar, *Information theory methods in communication complexity*, in Proceedings of the 17th Annual IEEE Conference on Computational Complexity, 2002, pp. 93–102.

[BNRW05] H. Buhrman, I. Newman, H. Röhrig, and R. de Wolf, *Robust polynomials and quantum algorithms*, in Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS 2005), Lecture Notes in Comput. Sci. 3404, Springer, Berlin, 2005, pp. 593–604; available online from http://www.arxiv.org/abs/quant-ph/0309220.

[BPSW05] P. Beame, T. Pitassi, N. Segerlind, and A. Wigderson, *A strong direct product lemma for corruption and the multiparty NOF communication complexity of disjointness*, in Proceedings of the 20th Annual IEEE Conference on Computational Complexity, 2005, pp. 52–66.

[BŠ06] H. Buhrman and R. Špalek, *Quantum verification of matrix products*, in Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, Miami, 2006; available online from http://www.arxiv.org/abs/quant-ph/0409035.

[BTY94] P. Beame, M. Tompa, and P. Yan, *Communication-space tradeoffs for unrestricted protocols*, SIAM J. Comput., 23 (1994), pp. 652–661.

[Buh00] H. Buhrman, *Quantum computing and communication complexity*, Bull. Eur. Assoc. Theor. Comput. Sci., 70 (2000), pp. 131–141.

[BW02] H. Buhrman and R. de Wolf, *Complexity measures and decision tree complexity: A survey*, Theoret. Comput. Sci., 288 (2002), pp. 21–43.

[CDNT98] R. Cleve, W. van Dam, M. Nielsen, and A. Tapp, *Quantum entanglement and the communication complexity of the inner product function*, in Quantum Computing and Quantum Communications, Lecture Notes in Comput. Sci. 1509, Springer, Berlin, 1998, pp. 61–74; available online from http://www.arxiv.org/abs/quant-ph/9708019.

[CR92] D. Coppersmith and T. J. Rivlin, *The growth of polynomials bounded at equally spaced points*, SIAM J. Math. Anal., 23 (1992), pp. 970–983.

[CSWY01] A. Chakrabarti, Y. Shi, A. Wirth, and A. Yao, *Informational complexity and the direct sum problem for simultaneous message complexity*, in Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science, 2001, pp. 270–278.

[GNW95] O. Goldreich, N. Nisan, and A. Wigderson, *On Yao's XOR Lemma*, Technical report TR–95–050, ECCC, 1995; available online at http://www.eccc.uni-trier.de/eccc/.

[Gro96] L. K. Grover, *A fast quantum mechanical algorithm for database search*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing, 1996, pp. 212–219; available online from http://www.arxiv.org/abs/quant-ph/9605043.

[HW02] P. Høyer and R. de Wolf, *Improved quantum communication complexity bounds for disjointness and equality*, in Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2002), Lecture Notes in Comput. Sci. 2285, Springer, Berlin, 2002, pp. 299–310; available online from http://www.arxiv.org/abs/quant-ph/0109068.

[Kla00] H. Klauck, *Quantum communication complexity*, in Proceedings of Workshop on Boolean Functions and Applications at the 27th Annual International Colloquium on Automata, Languages and Programming, 2000, pp. 241–252; available online from http://www.arxiv.org/abs/quant-ph/0005032.

[Kla03] H. Klauck, *Quantum time-space tradeoffs for sorting*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, 2003, pp. 69–76; available online from http://www.arxiv.org/abs/quant-ph/0211174.

[Kla04] H. Klauck, *Quantum and classical communication-space tradeoffs from rectangle bounds*, in FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Lecture Notes in Comput. Sci. 3328, Springer, Berlin, 2004, pp. 384–395.

[KN97] E. Kushilevitz and N. Nisan, *Communication Complexity*, Cambridge University Press, Cambridge, UK, 1997.

[Knu03] D. E. Knuth, *Combinatorial matrices*, in Selected Papers on Discrete Mathematics, CSLI Lecture Notes 106, Stanford University, Stanford, CA, 2003, pp. 177–188.

[Kre95] I. Kremer, *Quantum Communication*, Master's thesis, Computer Science Department, The Hebrew University of Jerusalem, Jerusalem, 1995.

[KS92] B. Kalyanasundaram and G. Schnitger, *The probabilistic communication complexity of set intersection*, SIAM J. Discrete Math., 5 (1992), pp. 545–557.

[LTT92] T. W. Lam, P. Tiwari, and M. Tompa, *Trade-offs between communication and space*, J. Comput. System Sci., 45 (1992), pp. 296–315.

[NC00] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, UK, 2000.

[Nis91] N. Nisan, *CREW PRAMs and decision trees*, SIAM J. Comput., 20 (1991), pp. 999–1007.

[NRS94] N. Nisan, S. Rudich, and M. Saks, *Products and help bits in decision trees*, in Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science, 1994, pp. 318–329.

[NS94] N. Nisan and M. Szegedy, *On the degree of Boolean functions as real polynomials*, Comput. Complexity, 4 (1994), pp. 301–313.

[Pat92] R. Paturi, *On the degree of polynomials that approximate symmetric Boolean functions*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, 1992, pp. 468–474.

[PRW97] I. Parnafes, R. Raz, and A. Wigderson, *Direct product results and the GCD problem, in old and new communication models*, in Proceedings of 29th ACM Symposium on Theory of Computing, 1997, pp. 363–372.

[Raz92] A. Razborov, *On the distributional complexity of disjointness*, Theoret. Comput. Sci., 106 (1992), pp. 385–390.

[Raz03] A. Razborov, *Quantum communication complexity of symmetric predicates*, Izv. Ross. Akad. Nauk. Ser. Mat., 67 (2003), pp. 159–176; available online from http://www.arxiv.org/abs/quant-ph/0204025. %pagebreak

[Riv90] T. J. Rivlin, *Chebyshev Polynomials: From Approximation Theory to Algebra and Number Theory*, 2nd ed., Wiley-Interscience, New York, 1990.

[Sha01] R. Shaltiel, *Towards proving strong direct product theorems*, in Proceedings of the 16th Annual IEEE Conference on Computational Complexity, 2001, pp. 107–119.

[Wol02] R. de Wolf, *Quantum communication and complexity*, Theoret. Comput. Sci., 287 (2002), pp. 337–353.

[Yao77] A. C-C. Yao, *Probabilistic computations: Toward a unified measure of complexity*, in Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science, 1977, pp. 222–227.

[Yao82] A. C-C. Yao, *Theory and applications of trapdoor functions*, in Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science, 1982, pp. 80–91.

[Yao93] A. C-C. Yao, *Quantum circuit complexity*, in Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science, 1993, pp. 352–360.

SIAM J. COMPUT.
Vol. 36, No. 5, pp. 1494–1511

INTEGRALITY RATIO FOR GROUP STEINER TREES AND DIRECTED STEINER TREES*

ERAN HALPERIN†, GUY KORTSARZ‡, ROBERT KRAUTHGAMER§, ARAVIND SRINIVASAN¶, AND NAN WANG‖

Abstract. The natural relaxation for the group Steiner tree problem, as well as for its generalization, the directed Steiner tree problem, is a flow-based linear programming relaxation. We prove new lower bounds on the integrality ratio of this relaxation. For the group Steiner tree problem, we show that the integrality ratio is $\Omega(\log^2 k)$, where k denotes the number of groups; this holds even for input graphs that are *hierarchically well-separated trees*, introduced by Bartal [in *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, 1996, pp. 184–193], in which case this lower bound is tight. This also applies for the directed Steiner tree problem. In terms of the number n of vertices, our results for the directed Steiner problem imply an $\Omega(\frac{\log^2 n}{(\log\log n)^2})$ integrality ratio. For both problems, these are the first lower bounds on the integrality ratio that are superlogarithmic in the input size. This exhibits, for the first time, a relaxation of a natural optimization problem whose integrality ratio is known to be superlogarithmic but subpolynomial. Our results and techniques have been used by Halperin and Krauthgamer [in *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, 2003, pp. 585–594] to show comparable inapproximability results, assuming that NP has no quasi-polynomial Las Vegas algorithms. We also show algorithmically that the integrality ratio for the group Steiner tree problem is much better for certain families of instances, which helps pinpoint the types of instances (parametrized by optimal solutions to their flow-based relaxations) that appear to be most difficult to approximate.

Key words. group Steiner tree, directed Steiner tree, flow-based relaxation, linear programming relaxation, integrality ratio, approximation algorithms

AMS subject classifications. 05C05, 05C80, 60C05, 68W25, 90C05, 90C10, 90C27

DOI. 10.1137/S0097539704445718

1. Introduction.

GROUP-STEINER-TREE is a network design problem that generalizes both SET-COVER and STEINER-TREE. DIRECTED-STEINER-TREE is a further generalization of GROUP-STEINER-TREE. The natural relaxation for these two problems is a flow-based linear programming (LP) relaxation. We show a polylogarithmic (about log squared) lower bound on the integrality ratio of this relaxation.

*Received by the editors August 18, 2004; accepted for publication (in revised form) July 17, 2006; published electronically February 5, 2007. A preliminary version of this work appeared in *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2003, pp. 275–284.
http://www.siam.org/journals/sicomp/36-5/44571.html

†International Computer Science Institute, 1947 Center St., Suite 600, Berkeley, CA 94704 (heran@icsi.berkeley.edu). This work was done while this author was also affiliated with the Computer Science Division of the University of California at Berkeley, supported in part by NSF grants CCR-9820951 and CCR-0121555 and DARPA cooperative agreement F30602-00-2-0601.

‡Department of Computer Sciences, Rutgers University, Camden, NJ 08102 (guyk@camden.rutgers.edu).

§IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120 (robi@almaden.ibm.com). This work was done while this author was with the International Computer Science Institute and with the Computer Science Division of the University of California at Berkeley, supported in part by NSF grants CCR-9820951 and CCR-0121555 and DARPA cooperative agreement F30602-00-2-0601.

¶Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742 (srin@cs.umd.edu). This author's work was supported in part by NSF awards CCR-0208005 and CNS-0426683.

‖Morgan Stanley, Inc., 1220 Avenue of the Americas, New York, NY 10020. This work was done while this author was at the Department of Computer Science, University of Maryland, College Park, MD 20742 (nwang@cs.umd.edu). This author's work was supported in part by NSF awards CCR-0208005 and CNS-0426683.

For both problems, these are the first such lower bounds that are superlogarithmic in the input size, and our bounds are, in fact, nearly tight in the important special case of input graphs which are tree networks. Let n be the number of vertices and k the number of groups. Our probabilistic approach and our analysis have been used in [HK03] to show that for every fixed $\epsilon > 0$, Group-Steiner-Tree and Directed-Steiner-Tree admit no efficient $\Omega(\log^{2-\epsilon} k)$ and $\Omega(\log^{2-\epsilon} n)$ approximations, respectively, unless NP has quasi-polynomial time Las Vegas algorithms. We also present improved approximation algorithms for certain families of instances of Group-Steiner-Tree, shedding light on the type of instances that appear to be most difficult for the flow-based relaxation.

1.1. The group Steiner tree problem. The (undirected) group Steiner tree problem is as follows. Given an undirected graph $G = (V, E)$, a collection of subsets (called groups) $g_1, g_2, \ldots, g_k$ of V, and a weight $w_e \geq 0$ for each edge $e \in E$, the problem is to construct a minimum-weight tree in G that spans at least one vertex from each group. We can assume without loss of generality that there is a distinguished vertex $r \in V$ (called the root) that must be included in the output tree. The case where $|g_i| = 1$ for all i is just the classical Steiner tree problem; the case where G is a star can be used to model Set-Cover (cf. [GKR00]).

A natural flow-based relaxation for this problem is as follows. Find a *capacity* $x_e \in [0, 1]$ for each edge $e \in E$ so that the capacities can support one unit of flow from r to g_i, separately for each g_i (as opposed to supporting a unit flow simultaneously for all g_i). Subject to this constraint, we want to minimize $\sum_e w_e x_e$. It is easy to check that the feasible solutions which satisfy $x_e \in \{0, 1\}$ for all e exactly correspond to feasible solutions for the group Steiner tree problem; hence, the above flow-based relaxation is indeed a valid LP relaxation for the problem. This is a natural relaxation for this problem (and for some of its generalizations) and is the main subject of investigation in this paper.

We start with a useful definition from [Bar96].

Definition 1.1. *Let $c > 1$. A c-hierarchically well-separated tree (c-HST) is a rooted weighted tree such that* (i) *all leaves are at the same distance from the root,* (ii) *the edges in the same level are equal-weighted, and* (iii) *the weight of an edge is exactly $1/c$ times the weight of its parent edge.*

(Remark. Item (ii) *is slightly stronger than the original definition from* [Bar96]*, but can be assumed without loss of generality due to the analyses of* [Bar96, Bar98, KRS01]*.)*

We simply say "HST" when referring to a c-HST for an arbitrary constant $c > 1$.

The first polylogarithmic approximation algorithm for Group-Steiner-Tree was achieved in the elegant work of Garg, Konjevod, and Ravi [GKR00]. A brief sketch of their $O(\log n \log\log n \log N \log k)$-approximation algorithm, where $n = |V|$ and $N = \max_i |g_i|$, is as follows. First, the powerful results of [Bar98] are used to reduce the problem to the case where G is a *tree* T, with an $O(\log n \log\log n)$ factor loss in the approximation ratio. Furthermore, T can be assumed to be a c-HST for any desired constant $c > 1$. Next, solve the flow-based LP relaxation on T and round the fractional solution into an integral solution for T by applying a novel randomized rounding approach that is developed in [GKR00]. It is established in [GKR00] that for any tree T, this randomized rounding leads to an $O(\log N \log k)$-approximation. Thus, for the input graph G, we get an $O(\log n \log\log n \log N \log k)$-approximation. From a technical viewpoint, one of the main difficulties in [GKR00] is that a nontrivial analysis of the randomized process is required. The analysis uses Janson's inequality

[Jan90] in an interesting way. The work of [GKR00] has been extended and expanded in several ways: Their algorithm was derandomized in [CCGG98, Sri01]; an alternative (combinatorial) algorithm is devised in [CEK06]; and the loss incurred by the reduction to an HST is improved to $O(\log n)$ in [FRT04].

Since the first appearance of a polylogarithmic approximation for Group-Steiner-Tree (in the conference version of [GKR00] in 1998), there has been much interest in whether the approximation ratio can be improved. One concrete notable question in this regard has been the following: Can we achieve an approximation ratio better than $O(\log N \log k)$ for trees? This is interesting for at least two reasons. First, since [GKR00] shows a reduction to the case of trees as seen above, an improved approximation for trees (or even for the case of c-HSTs for some constant $c > 1$) would lead directly to an improved approximation for general graphs. Further, even the case where G is a star (which is a tree) captures Set-Cover, for which $o(\log k)$-approximation is hard [LY94, Fei98, RS97]; thus there is an intriguing gap even on trees.

Our main technical result is that the integrality ratio of the flow-based relaxation for HSTs is $\Omega(\log^2 k)$. This bound is, in fact, tight—an $O(\log^2 k)$ bound on the integrality ratio holds for HSTs, as we show in section 2.7. Both bounds hold for c-HSTs where $c > 1$ is any fixed constant. Recall that the upper bound of [GKR00] for trees in general is $O(\log N \log k)$; our methods show an $\Omega(\log N \log k / \log\log N)$ lower bound on the integrality ratio, even for a class of HSTs. The same lower bounds also hold for trees where all weights are the same (i.e., unit-weight trees). Finally, we show randomized rounding algorithms for the flow-based relaxation that lead to improved approximation algorithms for certain special families of HSTs; this sheds light on the type of instances that are most difficult to approximate.

A log-squared lower bound on the integrality ratio for trees was conjectured circa 1998 by Uri Feige. The specific (randomly constructed) instance he suggested for this purpose (see section 2 for more details) has proved to be quite difficult to analyze. Our lower bound is shown via a slightly different random instance, which eliminates one source of correlation in the random choices, and makes the construction more amenable to analysis. Nevertheless, we are required to do intricate estimates that delve into low-order terms, in particular when crafting the precise induction hypothesis lying at the heart of the proof.

1.2. The directed Steiner tree problem. This is the directed version of the (undirected) Steiner tree problem. Given an edge-weighted directed graph $G = (V, E)$ that specifies a *root* vertex r and k *terminal* nodes $v_1, v_2, \ldots, v_k$, the goal is to construct a minimum-weight outbranching rooted at r, which spans all the terminals. This problem is easily seen to generalize the undirected Group-Steiner-Tree, as well as to be equivalent to the directed Group-Steiner-Tree. Aside from intrinsic interest, this problem is also of current interest, e.g., in the context of multicasting in the Internet (where internode distances are often not symmetric). The polynomial time approximation ratio currently known for this problem is k^ϵ for any constant $\epsilon > 0$ [CCC+99]; their algorithm extends to a polylogarithmic approximation ratio in quasi-polynomial running time. The flow-based relaxation here is similar: Install for every edge $e \in E$ a capacity $x_e \in [0, 1]$, so that a unit of flow can be shipped from r to v_i, separately for any given i. Intriguingly, it was recently shown in [ZK02] that this relaxation has an integrality ratio of $\Omega(\sqrt{k})$, precluding a polylog(k)-approximation algorithm based on this relaxation. However, the examples constructed in [ZK02] have $k = \Theta(\frac{\log^2 n}{(\log\log n)^2})$; hence, the result of [ZK02] does not exclude an $O(\log n)$

integrality ratio. Our GROUP-STEINER-TREE lower-bound result above also proves an $\Omega(\frac{\log^2 n}{(\log\log n)^2})$ lower bound on the integrality gap for DIRECTED-STEINER-TREE. The only lower bound known previously for DIRECTED-STEINER-TREE was $\Omega(\log n)$, since this problem generalizes SET-COVER.

As mentioned above, our results have paved the way to the improved hardness of approximation results of [HK03], which show that, for any fixed $\epsilon > 0$, GROUP-STEINER-TREE cannot be approximated within ratio $\log^{2-\epsilon} k$, and DIRECTED-STEINER-TREE cannot be approximated within ratio $\log^{2-\epsilon} n$, unless NP has quasi-polynomial time Las Vegas algorithms. The influence of our work on these hardness results is threefold. First, our lower bounds on the integrality ratio have motivated working on a hardness of approximation result. Second, the insights our analysis provides regarding the (edge-weight) structure of instances that are difficult to approximate inspired specific details of the hardness reduction. Third, our main technical lemma (whose proof is rather nontrivial) is, in fact, crucial to [HK03].

Organization. Our lower bounds on the integrality ratio of GROUP-STEINER-TREE and DIRECTED-STEINER-TREE are shown in section 2. We then prove algorithmically that the integrality ratio of the former problem is much better for certain families of instances in section 3; this pinpoints the type of instances that appear difficult for this relaxation. Finally, concluding remarks are made in section 4.

2. Lower bounds on the integrality ratio. In this section we prove a lower bound of $\Omega(\log^2 k)$ on the integrality ratio of the flow-based relaxation of GROUP-STEINER-TREE even on HSTs. In terms of n, the gap is $\Omega(\frac{\log^2 n}{(\log\log n)^2})$. We start (section 2.1) by describing the LP relaxation and constructing a family of 2-HST instances, accompanied by an overview of the analysis; then, the main technical parts (sections 2.2 and 2.3) analyze the fractional and the integral solutions of this linear program. We also show how simple modifications to this construction extend the integrality ratio to unit-weight trees (section 2.4) and to c-HSTs for an arbitrary constant $c > 1$ (section 2.5). We further point out how this immediately leads to a lower bound of $\Omega(\frac{\log^2 n}{(\log\log n)^2})$ on the integrality ratio for DIRECTED-STEINER-TREE (section 2.6). Finally, the lower bound of $\Omega(\log^2 k)$ is shown to be tight (in section 2.7).

2.1. The relaxation and the instance. The cut-based relaxation (that is equivalent to the flow-based relaxation) for GROUP-STEINER-TREE is as follows (here, $\delta(S)$ is the set of edges with exactly one endpoint in $S \subset V$):

(1)

$$\begin{aligned} \text{Minimize} \quad & \sum_{e\in E} w_e x_e, \\ & \sum_{e\in\delta(S)} x_e \geq 1 \quad \forall S \subseteq V \text{ s.t. } r \in S \text{ and } S \cap g_j = \emptyset \text{ for some group } g_j, \\ & 0 \leq x_e \leq 1 \quad \forall e \in E. \end{aligned}$$

Let $\mathbb{T}_n$ be a 2-HST tree with n nodes defined by the following random process. Let the collection of groups be $\mathcal{G} = \{g_1, g_2, \ldots, g_k\}$. The groups are defined by a random process. The value of k, as well as those of two other parameters H and d, will be defined shortly (in terms of n). The height (i.e., depth) of $\mathbb{T}_n$ is H, and every nonleaf vertex has d children. The root of $\mathbb{T}_n$ is denoted r. The *level* of a vertex is its depth; r is at level 0, and there are $H + 1$ levels. An edge is said to be at level

i if and only if it connects a vertex at level $i-1$ to a vertex at level i. Each edge at level i has weight $1/2^i$; thus, for instance, edges incident at r have weight $1/2$. Each group g_j is a subset of the leaves, described as follows. We shall associate a subset $A(\ell) \subseteq \mathcal{G}$ of the groups with each leaf ℓ and define each group g_j to be the set of leaves ℓ for which $g_j \in A(\ell)$. Thus, a solution that reaches a leaf ℓ by a path from r covers all groups in $A(\ell)$. To define $A(\ell)$ for each leaf ℓ, we now recursively and randomly define a set $A(v)$ for every node v in the tree (including nonleaf nodes) as follows. Proceed *independently* for each group g_j as follows. We start by letting $g_j \in A(r)$ with probability 1. In general, if $g_j \in A(u)$ for some nonleaf node u, then for each child v of u, we independently put g_j in $A(v)$ with probability $1/2$. Thus, this random process goes top-down in the tree, independently for each group. Note that the number of vertices in $\mathbb{T}_n$ is $n \simeq d^H$, where H is the height of the tree. Clearly, the expected size of every group is $d^H/2^H$.

Parameters and notation. We set $d = c_0 \log n$ for some universal constant $c_0 > 0$; this will be used in some Chernoff bound arguments in section 2.2. It then follows that $H = \frac{\log n}{\log d} = \Theta(\log n / \log\log n)$. We further set $k = 2^{2H}$; thus, $\log k = \Theta(\log n / \log\log n)$. Throughout, *with high probability* means with probability that is at least, say, $1 - 1/n$. All probabilities refer to the randomness in constructing the instance $\mathbb{T}_n$.

Feige's instance. The construction suggested by Feige circa 1998 is the following: Take a complete tree of arity 4 (i.e., every nonleaf vertex has four children) and height $\log_2 k$; now generate k groups, each containing k leaves, by an independent randomized branching process that starts from the root and randomly picks two out of four children until the leaves are reached. This random instance differs from $\mathbb{T}_n$ in its degree, which is constant rather than logarithmic, and in that the choices made (when generating a single group) at each child of a vertex are correlated (rather than independent).

Overview of the analysis. We first show (section 2.2) that with high probability the instance $\mathbb{T}_n$ has a feasible fractional solution of cost $O(H)$. In this solution, all edges e in the same level of the tree are assigned the same value x_e, and this value is chosen so that the total cost of every level in the tree is the same, namely, $O(1)$. The feasibility of this solution is shown by employing a sequence of Chernoff bound arguments, and this is the reason why the degree d of the tree must be (at least) logarithmic. This is in contrast to Feige's instance, where the feasibility (of a similar fractional solution) is guaranteed by construction, i.e., with probability 1, and thus the tree can have a constant degree.

We then show (section 2.3) that with high probability the cost of any integral solution for $\mathbb{T}_n$ is lower bounded by $\Omega(H^2 \log k)$. At a high level, we imitate the argument known for Set-Cover; we show that for any single low-cost integral solution (i.e., subtree of $\mathbb{T}_n$), with high probability over the randomness in constructing the groups, this solution is infeasible (i.e., does not cover all the groups), and then we take a union bound over all possible integral solutions. In fact, any single vertex of $\mathbb{T}_n$ together with its children is essentially a "standard" Set-Cover instance with integrality ratio $\Omega(\log k)$.

The main technical work is to estimate the probability that an arbitrary (but fixed in advance) integral solution is feasible. Unlike in the Set-Cover scenario, where this is a straightforward calculation, in the Group-Steiner-Tree instance $\mathbb{T}_n$ the solution's structure comes into play. For instance, the integral solution might not be a regular subtree of $\mathbb{T}_n$, and its cost need not be split evenly among the different levels (of $\mathbb{T}_n$). We prove some upper bound on the probability that this solution for $\mathbb{T}_n$ is

feasible. Our analysis shows that to maximize this upper bound on the probability, it is essentially best to have an even "split" of the cost used under a vertex v (i.e., the edges of the solution that belong to the subtree rooted at v). While we do not claim that this is the worst case for the *exact* probability of feasibility, our upper bound gives a good enough estimate. However, the analysis does not specify how many children of v should have a nonzero cost under them; in fact, this value is very sensitive to lower-order tradeoffs between different levels in the tree.

The main difficulty in the analysis is to distill the effect of H, the height of the tree, on the feasibility probability. Our proof is by induction on the height of the tree and uncovers a very delicate tradeoff between the height of a subtree and its cost. This tradeoff eventually translates to the cost of the integral solution for $\mathbb{T}_n$ having, on top of the $\log k$ term which comes from Set-Cover (i.e., a single level), also a linear dependence on the height H. Due to seemingly technical limitations this proof works only for $H \leq \frac{1}{2}\log k$, but we show in section 2.7 that this is unavoidable. Interestingly, there is an analogy with the approximation algorithm of [GKR00], whose rounding procedure pays, at some intermediate stage, an $O(H \log k)$ factor, and then shows that, in effect, H can be upper bounded by $O(\log N)$.

2.2. The fractional solution. Recall that $d = c_0 \log n$. We start with a couple of propositions which show that if the constant c_0 is sufficiently large, then certain quantities related to our randomly chosen groups stay close to their mean. Henceforth, the phrase "with high probability" will mean "with a probability of $1 - o(1)$."

Proposition 2.1. *Let c_0 be a sufficiently large constant. Then, with high probability, all groups have size at least $(d/2)^H/3$.*

Proof. Fix j. We now show that if c_0 is large enough, then $\Pr\left[|g_j| < (d/2)^H/3\right] \leq 1/n^2$. We may then apply the union bound over all j to conclude the proof.

Let $\delta = 1/4$. Let X_1 be the number of vertices u at level 1 (i.e., children of r) such that $g_j \in A(u)$. Then X_1 has binomial distribution $X_1 \sim B(d, 1/2)$, so by a Chernoff bound on the lower-tail (see, e.g., [MR95]),

$$\Pr\left[X_1 \leq (1-\delta)\frac{d}{2}\right] \leq e^{-\frac{1}{2}\delta^2 \cdot \frac{d}{2}}.$$

Let X_2 be the number of vertices u at level 2 such that $g_j \in A(u)$. Then X_2 has binomial distribution $X_2 \sim B(X_1 \cdot d, 1/2)$. Suppose that $X_1 > (1-\delta)\mathbb{E}[X_1] = (1-\delta)\frac{d}{2}$. Then, it is immediate that $X_2 \mid (X_1 > (1-\delta)\frac{d}{2})$ stochastically dominates a random variable $X_2' \sim B((1-\delta)\frac{d}{2} \cdot d, 1/2)$, i.e., $\Pr[X_2 \leq t] \leq \Pr[X_2' \leq t]$ for all t. By applying the Chernoff bound on X_2' we get

$$\Pr\left[X_2' \leq \left(1 - \frac{\delta}{2}\right)(1-\delta)\left(\frac{d}{2}\right)^2\right] \leq e^{-\frac{1}{2}(\frac{\delta}{2})^2 \cdot (1-\delta)(\frac{d}{2})^2}.$$

Continue similarly for $i = 3, \ldots, H$ by defining X_i to be the number of vertices u at level i such that $g_j \in A(u)$, and by assuming that $X_{i-1} > (1 - \frac{\delta}{2^{i-2}}) \cdot \ldots \cdot (1 - \frac{\delta}{2})(1-\delta)(\frac{d}{2})^i$. We get by the Chernoff bound that

(2)
$$\Pr\left[X_i' \leq \left(1 - \frac{\delta}{2^{i-1}}\right) \cdot \ldots \cdot \left(1 - \frac{\delta}{2}\right)(1-\delta)\left(\frac{d}{2}\right)^i\right] \leq e^{-\frac{1}{2}(\frac{\delta}{2^{i-1}})^2 \cdot (1 - \frac{\delta}{2^{i-2}}) \cdot \ldots \cdot (1-\frac{\delta}{2})(1-\delta)(\frac{d}{2})^i}.$$

For any $0 < \delta' \le \frac{1}{2}$ we have $1-\delta' \ge \frac{1}{1+2\delta'} \ge e^{-2\delta'}$. Thus, $(1-\frac{\delta}{2^{i-1}})\cdot\ldots\cdot(1-\frac{\delta}{2})(1-\delta) \ge e^{-\frac{\delta}{2^{i-2}}-\ldots-\delta-2\delta} \ge e^{-4\delta} > \frac{1}{3}$. It follows that the tail bound obtained in the right-hand side of (2) is at most $e^{-\Omega((d/8)^i)}$. Applying the union bound on these H events, we get that with high probability none of them happens (if the constant c_0 is sufficiently large), and in particular, $X_H \ge (1-\frac{\delta}{2^{H-1}})\cdot\ldots\cdot(1-\frac{\delta}{2})(1-\delta)(\frac{d}{2})^H \ge \frac{1}{3}(\frac{d}{2})^H$. This concludes the proof of Proposition 2.1. □

The following proposition has a similar proof; the main difference is that we will now employ Chernoff bounds on the upper-tail.

PROPOSITION 2.2. *Suppose that the constant c_0 is large enough. Then with high probability, the following holds for every level i and every group g_j: If a vertex u at level i is such that $g_j \in A(u)$, then the number of leaves ℓ in the subtree rooted at u which satisfy $g_j \in A(\ell)$ is at most $3(d/2)^{H-i}$.*

Proof. Fix a pair (i,j) and a vertex u at level i such that $g_j \in A(u)$. Let $L(u)$ be the set of leaves of the subtree rooted at u. We now show that if c_0 is large enough, then $\Pr\left[|g_j \bigcap L(u)| > 3(d/2)^{H-i}\right] \le 1/n^3$. We then apply a union bound over all (i,j,u) to conclude the proof.

Let $\delta = 1/4 < \frac{\ln 3}{2}$. Let X_1 be the number of vertices v at level 1 of the subtree rooted at u (i.e., children of u) such that $g_j \in A(v)$. Then X_1 has binomial distribution $X_1 \sim B(d, 1/2)$, so by a Chernoff bound on the upper-tail (see, e.g., [MR95]),

$$\Pr\left[X_1 \ge (1+\delta)\frac{d}{2}\right] \le e^{-\frac{\delta^2}{3}\cdot\frac{d}{2}}.$$

Let X_2 be the number of vertices v at level 2 of the subtree rooted at u such that $g_j \in A(v)$. Then X_2 has binomial distribution $X_2 \sim B(X_1 \cdot d, 1/2)$. Suppose that $X_1 < (1+\delta)\mathbb{E}[X_1] = (1+\delta)\frac{d}{2}$. Then, it is immediate that $X_2 \mid (X_1 < (1+\delta)\frac{d}{2})$ is stochastically dominated by a random variable $X_2' \sim B((1+\delta)\frac{d}{2}\cdot d, 1/2)$; i.e., $\Pr[X_2 \ge t] \le \Pr[X_2' \ge t]$ for all t. By applying the Chernoff bound on X_2' we get

$$\Pr\left[X_2' \ge \left(1+\frac{\delta}{2}\right)(1+\delta)\left(\frac{d}{2}\right)^2\right] \le e^{-\frac{1}{3}(\frac{\delta}{2})^2\cdot(1+\delta)(\frac{d}{2})^2}.$$

Continue similarly for $l = 3,\ldots,H-i$ by defining X_l to be the number of vertices v at level l of the subtree rooted at u such that $g_j \in A(v)$, and by assuming that $X_l < (1+\frac{\delta}{2^{l-1}})\cdot\ldots\cdot(1+\frac{\delta}{2})(1+\delta)(\frac{d}{2})^l$. We get by the Chernoff bound that

(3)
$$\Pr\left[X_l' \ge \left(1+\frac{\delta}{2^{l-1}}\right)\cdot\ldots\cdot\left(1+\frac{\delta}{2}\right)(1+\delta)\left(\frac{d}{2}\right)^l\right] \le e^{-\frac{1}{3}(\frac{\delta}{2^{l-1}})^2\cdot(1+\frac{\delta}{2^{l-2}})\cdot\ldots\cdot(1+\frac{\delta}{2})(1+\delta)(\frac{d}{2})^l}.$$

The tail bound obtained in the right-hand side of (3) is clearly at most $e^{-\Omega((d/8)^l)}$. Applying the union bound on these $H-i \le H$ events, we get that with probability at least $1-1/n^3$ none of these events happens if the constant c_0 is sufficiently large; in particular, $X_{H-i} \le (1+\frac{\delta}{2^{H-i-1}})\cdot\ldots\cdot(1+\frac{\delta}{2})(1+\delta)(\frac{d}{2})^{H-i} \le 3(\frac{d}{2})^{H-i}$. (This is because of the following. For any δ' we have $1+\delta' \le e^{\delta'}$. Thus, $(1+\frac{\delta}{2^{l-1}})\cdot\ldots\cdot(1+\frac{\delta}{2})(1+\delta) \le e^{\frac{\delta}{2^{l-1}}+\ldots+\frac{\delta}{2}+\delta} \le e^{2\delta} < 3$.) This concludes the proof of Proposition 2.2. □

We now upper bound the value of LP (1) for the tree $\mathbb{T}_n$ by exhibiting a feasible solution for it: Let each edge e at each level i have value $\hat{x}_e = 9\cdot(2/d)^i$.

LEMMA 2.3. *With high probability, $\hat{x}$ is a feasible solution to LP* (1). *Its value is* $9H$.

Proof. Observe that $\hat{x}$ satisfies the constraints of LP (1) if (see also [GKR00]), for every group g_j, every cut $(S, \bar{S})$ separating r from all the vertices of g_j has capacity at least 1, where the capacity of each edge e is $\hat{x}_e$. By the (single-source) max-flow min-cut theorem (or, say, weak duality) it suffices to show that for every group g_j, a unit of flow can be shipped from the root r to the vertices of g_j while obeying the "capacity" $\hat{x}_e$ of each edge e. To this end, fix a group g_j and define the flow f as follows. For every vertex v in g_j (i.e., for every leaf v such that $g_j \in A(v)$), ship $3 \cdot (2/d)^H$ units of flow along the unique simple path from r to v. By Proposition 2.1, the total flow shipped to g_j is at least $|g_j| \cdot 3 \cdot (2/d)^H \geq 1$ with high probability. Next, consider an edge connecting a node u at some level i to its parent. If $g_j \notin A(u)$, no flow is shipped through this edge; if $g_j \in A(u)$, the total flow shipped through this edge (i.e., through u) is, by Proposition 2.2, at most $3(d/2)^{H-i} \cdot 3(2/d)^H = 9(2/d)^i$ with high probability. In both cases, the flow along the edge obeys the edge's capacity. We conclude that with high probability $\hat{x}$ is a feasible solution to LP (1).

The value of the solution $\hat{x}$ is $\sum_{i=1}^{H} d^i \cdot 1/2^i \cdot 9(2/d)^i = 9H$ since each level i contains d^i edges of weight $1/2^i$. □

2.3. The integral solution. We now show that with probability $1 - o(1)$ (over the random choice of the groups), all integral solutions have value $\Omega(H^2 \log k)$. Whenever we say that some T' is a subtree of $\mathbb{T}_n$, we allow T' to be an arbitrary connected subgraph of $\mathbb{T}_n$. Since $\mathbb{T}_n$ is rooted, any subtree T' of $\mathbb{T}_n$ is also thought of as rooted in the obvious way: the node in T' of the smallest depth is the root of T' (and is denoted $\text{root}(T')$). Also, when we say that some T' is a subtree of $\mathbb{T}_n$ with root u, we allow T' to be an arbitrary connected subgraph of $\mathbb{T}_n$ with root u.

Let $M(c)$ be the number of subtrees of $\mathbb{T}_n$ which are rooted at r and have total weight at most c. Fix $g_j \in \mathcal{G}$. For any given subtree T' of $\mathbb{T}_n$, let $p_j(T')$ be the probability that no leaf of T' belongs to the group g_j, conditioned on the event that $g_j \in A(\text{root}(T'))$. Since by symmetry $p_j(T') = p_i(T')$ for all i, j, we will simply denote it by $p(T')$. We now define a key value $f(H, i, c)$ as follows. Choose an arbitrary vertex u at level i. Then $f(H, i, c)$ is the minimum value of $p(T')$, taken over all possible subtrees T' that are rooted at u and have total weight at most c. (If there is no such T', then $f(H, i, c) = 1$. Also, it is easy to see by symmetry that $f(H, i, c)$ does not depend upon the choice of j or u.) Let P_c be the probability that there exists an integral solution of weight c. We wish to show that $P_c = o(1)$ for c that is smaller than a certain threshold of the order $H^2 \log k$. Using the independence between the different groups and applying a union bound over all possible subtrees rooted at r that have total weight c, we obtain

$$P_c \leq M(c)(1 - f(H, 0, c))^k. \tag{4}$$

We now have to lower bound f and upper bound M. We employ the following crude bound on $M(c)$. Note that it suffices to count only subtrees of $\mathbb{T}_n$ that span distinct sets of leaves (since the groups g_i contain only leaves). Observing that $\mathbb{T}_n$ has d^H leaves, and a subtree of total weight at most c spans at most $c2^H$ leaves (since each spanned leaf requires a distinct edge at level H), we get that

$$M(c) \leq \binom{d^H}{c2^H} \leq d^{cH2^H}. \tag{5}$$

Our goal in the next subsection is to prove the following key lemma.

LEMMA 2.4. *For $H \le \frac{1}{2}\log k$ and a constant $\gamma > 0$ that is sufficiently large, we have $f(H,0,c) \ge e^{-\gamma c/H^2}$. Thus $P_c \le M(c)(1-f(H,0,c))^k \le M(c)\cdot\exp\{-k\cdot e^{-\gamma c/H^2}\}$.*

2.3.1. Bounding $f(H,0,c)$. We start with some preliminaries. The main technical result is Lemma 2.7. It gives a more general bound for f than that stated in Lemma 2.4, and hence the proof of Lemma 2.4 would follow quite easily.

PROPOSITION 2.5. *Let $l \ge 2$ and $\beta > 0$. The minimum of $\sum_{S\subseteq\{1,\ldots,l\}}\prod_{i\in S} e^{-\beta x_i}$ over all $(x_1,\ldots,x_l)$ with a given $\sum_{i=1}^{l} x_i$ is then attained when all x_i are equal.*

Proof. The minimum is clearly attained at some point $(x_1,\ldots,x_l)$, so assume to the contrary that at this point not all x_i are equal, say without loss of generality that $x_1 > \sum_i x_i/l > x_2$. We will show that changing both x_1 and x_2 to $\frac{x_1+x_2}{2}$ decreases the above sum while maintaining $\sum_i x_i$, which contradicts the assumption that $(x_1,\ldots,x_l)$ is a minimum point. Actually, it suffices to prove that

$$\sum_{S'\subseteq\{1,2\}}\prod_{i\in S'} e^{-\beta x_i} > \sum_{S'\subseteq\{1,2\}}\prod_{i\in S'} e^{-\beta\cdot\frac{x_1+x_2}{2}}\ , \tag{6}$$

since multiplying (6) by $\prod_{i\in S''} e^{-\beta x_i}$ and summing over all $S'' \subseteq \{3,\ldots,l\}$ shows that changing x_1, x_2 indeed decreases the abovementioned sum. To prove (6), observe that it simplifies to

$$e^{-\beta x_1} + e^{-\beta x_2} > 2e^{-\beta(x_1+x_2)/2},$$

which follows from the arithmetic mean-geometric mean inequality since $x_1 \ne x_2$. This completes the proof of Proposition 2.5. □

It is easy to check (by considering higher derivatives) that for all $B \ge 0$,

$$e^{-B} \ge 1 - B + \frac{B^2}{2} - \frac{B^3}{6}. \tag{7}$$

PROPOSITION 2.6. *For all $B_0 > 0$ there exists $\delta > 0$ such that for all $B \ge B_0$ we have $\frac{1+e^{-B}}{2} \ge e^{-\frac{B}{2+\delta}}$.*

Proof. Fix $B_0 > 0$. We first make sure that the inequality holds at B_0. By the arithmetic mean-geometric mean inequality $\frac{1+e^{-B_0}}{2} > e^{-B_0/2}$ (since $B_0 > 0$), so a sufficiently small $\delta > 0$ satisfies $\frac{1+e^{-B_0}}{2} > e^{-B_0/(2+\delta)}$. It now suffices to make sure that for all $B \ge B_0$ the derivative of the left-hand side is at least that of the right-hand side, i.e., that $-\frac{1}{2}e^{-B} \ge -\frac{1}{2+\delta}e^{-B/(2+\delta)}$. This holds for any $0 < \delta < B_0$ since $\frac{2+\delta}{2} = 1 + \delta/2 \le 1 + B_0/2 \le e^{B_0/2} \le e^{B/2} \le e^{B-B/(2+\delta)}$, completing the proof of Proposition 2.6. □

LEMMA 2.7. *Let γ be a sufficiently large constant. Then $f(H,h,c) \ge \exp(-\frac{\gamma c 2^h}{(H-h)^2})$ for all $c > 0$ and all $0 \le h \le H-1$.*

Proof. Fix B_0 to be an arbitrary positive constant, and let $\delta > 0$ be the corresponding constant from Proposition 2.6.

The proof is by backward induction on h; i.e., we assume that the claim holds for $h+1$ and prove it for h, where $h \le H-2$. We will consider the base case, which is the case that $h \ge H-1-\frac{6}{\delta}$, later on. In order to bound f, we derive a recurrence relation for $f(H,h,c)$. Recall the definition of $f(H,h,c)$: fix an arbitrary vertex u at level h, and take the minimum value of $p(T')$ over all possible subtrees T' rooted at u such that

the total weight of T' is at most c. We bound $f(H,h,c)$ by considering all possibilities of u having $l = 1, 2, \ldots, d$ children and all possible partitions $\vec{x}^{(l)} = (x_1, x_2, \ldots, x_l)$ of the weight c to (the subtrees under) these l children; since the edge from u to each of its children has weight $\frac{1}{2^{h+1}}$, we get that $\sum_{i=1}^{l} x_i = c - \frac{l}{2^{h+1}}$. We then get that

$$f(H,h,c) \geq \min_{1\leq l\leq d} \left\{ \min_{\vec{x}^{(l)}: x_i \geq 0, \sum_i x_i = c - \frac{l}{2^{h+1}}} \left\{ \frac{1}{2^l} \sum_{S\subseteq\{1,\ldots,l\}} \prod_{i\in S} f(H,h+1,x_i) \right\} \right\};$$

since once the l children of u are chosen, we need only consider the subset S of all children with g_j in their $A(\cdot)$ set. (Each such set S occurs with probability $1/2^l$.) Plugging in the induction hypothesis, we get that

$$f(H,h,c) \geq \min_{1\leq l\leq d} \left\{ \min_{\vec{x}^{(l)}: x_i \geq 0, \sum x_i = c - \frac{l}{2^{h+1}}} \left\{ \frac{1}{2^l} \sum_{S\subseteq\{1,\ldots,l\}} \prod_{i\in S} \exp\left(-\frac{\gamma x_i 2^{h+1}}{(H-h-1)^2} \right) \right\} \right\}. \tag{8}$$

For any l, we have by Proposition 2.5 that the right-hand side of (8) is minimized when all x_i are equal to $\frac{c}{l} - \frac{1}{2^{h+1}}$. We thus get that

$$\begin{aligned} f(H,h,c) &\geq \min_{1\leq l\leq d} \frac{1}{2^l} \sum_{S\subseteq\{1,\ldots,l\}} \left(\exp\left(-\frac{\gamma(\frac{c}{l} - \frac{1}{2^{h+1}})2^{h+1}}{(H-h-1)^2} \right) \right)^{|S|} \\ &= \min_{1\leq l\leq d} \frac{1}{2^l} \sum_{i=0}^{l} \binom{l}{i} \left(\exp\left(-\frac{\gamma(\frac{c}{l} - \frac{1}{2^{h+1}})2^{h+1}}{(H-h-1)^2} \right) \right)^{i} \\ &= \min_{1\leq l\leq d} \left(\frac{1 + \exp\left(-\frac{\gamma(\frac{c}{l} - \frac{1}{2^{h+1}})2^{h+1}}{(H-h-1)^2} \right)}{2} \right)^{l}. \end{aligned}$$

Fix l arbitrarily such that $1 \leq l \leq d$. Let $B = \left(\gamma(\frac{c}{l} - \frac{1}{2^{h+1}})2^{h+1}\right)/\left((H-h-1)^2\right)$ and $C = \left(\gamma\frac{c}{l}2^h\right)/\left((H-h)^2\right)$. To complete the induction, we want to prove that $\left(\frac{1+e^{-B}}{2}\right)^l \geq e^{-Cl}$, i.e., that

$$\frac{1+e^{-B}}{2} \geq e^{-C}. \tag{9}$$

We have four cases.

Case 1. In this case we assume that $C \geq \frac{B}{2}$. By the arithmetic mean-geometric mean inequality we have that $\frac{1+e^{-B}}{2} \geq e^{-B/2} \geq e^{-C}$, which proves (9).

Case 2. In this case we assume that $B \geq B_0$ and $\frac{B}{2+\delta} \leq C \leq \frac{B}{2}$; recall that δ is the constant from Proposition 2.6. Then we have from Proposition 2.6 that $\frac{1+e^{-B}}{2} \geq e^{-\frac{B}{2+\delta}} \geq e^{-C}$, which proves (9).

Case 3. In this case we assume that $C \leq \frac{B}{2}$ and $B < B_0$. Then by (7) we have $\frac{1+e^{-B}}{2} \geq 1 - \frac{B}{2} + \frac{B^2}{4} - \frac{B^3}{12}$. Since $C \geq 0$, we have (by Taylor's theorem) that $e^{-C} \leq 1 - C + \frac{C^2}{2}$. Thus, it suffices to prove that

$$1 - \frac{B}{2} + \frac{B^2}{4} - \frac{B^3}{12} \geq 1 - C + \frac{C^2}{2}.$$

Since $B < B_0 \leq \frac{1}{2}$ we have that $\frac{B^3}{12} \leq \frac{B^2}{24}$, and then since $2C \leq B$, we have that $\frac{B^2}{4} - \frac{B^3}{12} \geq \frac{5B^2}{24} \geq \frac{5C^2}{6}$. It therefore suffices to prove that

$$C + \frac{C^2}{3} \geq \frac{B}{2}.$$

Note that

$$\frac{B}{2} - C \leq \frac{\gamma 2^{h+1}\frac{c}{l}}{(H-h-1)^2(H-h)} - \frac{\gamma}{2(H-h-1)^2}.$$

Plugging in the values of B and C and simplifying, we get that it suffices to prove that

$$\frac{\gamma 2^{h+1}\frac{c}{l}}{(H-h-1)^2(H-h)} \leq \frac{\gamma}{2(H-h-1)^2} + \frac{\gamma^2\frac{c^2}{l^2}2^{2h}}{3(H-h)^4}.$$

If $\frac{2^{h+2}\frac{c}{l}}{H-h} \leq 1$, then the desired inequality indeed holds since $\frac{\gamma 2^{h+1}\frac{c}{l}}{(H-h-1)^2(H-h)} \leq \frac{\gamma}{2(H-h-1)^2}$. Otherwise, the inequality holds for any $\gamma \geq 96$, since then,

$$\frac{\gamma^2\frac{c^2}{l^2}2^{2h}}{3(H-h)^4} = \frac{\gamma}{6} \cdot \frac{2^{h+2}\frac{c}{l}}{H-h} \cdot \frac{\gamma\frac{c}{l}2^{h-1}}{(H-h)^3} \geq 16\frac{\gamma\frac{c}{l}2^{h-1}}{(H-h)^3} \geq \frac{\gamma 2^{h+1}\frac{c}{l}}{(H-h-1)^2(H-h)}.$$

Case 4. In this case we assume that $C < \frac{B}{2+\delta}$. Note that for $h \leq H-2$,

$$2+\delta \leq \frac{B}{C} = 2\frac{\frac{c}{l} - \frac{1}{2^{h+1}}}{\frac{c}{l}} \cdot \frac{(H-h)^2}{(H-h-1)^2} \leq 2\frac{(H-h)^2}{(H-h-1)^2} \leq 2 + \frac{6}{H-h-1}.$$

Thus, $h \geq H - 1 - \frac{6}{\delta}$. Since $\delta > 0$ is a constant, this is really the base case of the induction, which we shall prove directly. Consider a subtree T' of weight at most c that is rooted at a vertex u at level h. Since u has at most $c2^{h+1}$ children in T', each *not* having the group g_j in its $A(\cdot)$ set independently with probability $1/2$, with probability at least $2^{-c2^{h+1}}$ the subtree T' does not cover g_j. Thus, $f(H,h,c) \geq e^{-c2^{h+1}}$. Choosing a constant $\gamma \geq 2(1+\frac{6}{\delta})^2$, we get that $\gamma \geq 2(H-h)^2$, and thus

$$f(H,h,c) \geq e^{-c2^{h+1}} \geq \exp\left(-\frac{c2^h \cdot \gamma}{(H-h)^2}\right).$$

This concludes the proof of Lemma 2.7. □

Proof of Lemma 2.4. Lemma 2.7 implies that $f(H,0,c) \geq e^{-\gamma c/H^2}$. Plugging this into (4), we get that

$$P_c \leq M(c)(1-f(H,0,c))^k \leq M(c) \cdot \exp\{-k \cdot e^{-\gamma c/H^2}\}. \qquad \square$$

2.3.2. Bounding the weight of an integral solution. By Lemma 2.4 in conjunction with (5), we have

$$P_c \leq \exp\{cH2^H \log d - ke^{-\gamma\frac{c}{H^2}}\}.$$

Now, suppose that $c \leq \frac{1}{4\gamma}H^2 \ln k$. Then $cH2^H = O(2^H H^3 \log k)$. Recalling that $H = \frac{1}{2}\log k$, we have

$$P_c \leq \exp\{\tilde{O}(\sqrt{k}) - \Omega(k^{3/4})\} = o(1).$$

We conclude that with high probability no subtree of weight at most $\frac{1}{4\gamma}H^2 \log k$ covers all groups, and thus an optimal integral solution has value at least $\Omega(H^2 \log k)$. Since LP (1) has a fractional feasible solution of value $9H$, we get the following theorem.

THEOREM 2.8. *The integrality ratio of the relaxation* (1) *for* GROUP-STEINER-TREE *is* $\Omega(\log^2 k)$. *In terms of* N, k, *the integrality gap is* $\Omega(\log k \log N / \log\log N)$ *and in terms of* n *it is* $\Omega\big(\frac{\log^2 n}{(\log\log n)^2}\big)$.

2.4. Integrality ratio for unit-weight trees. The above analysis gives a lower bound on the integrality gap for GROUP-STEINER-TREE in HSTs. A consequent interesting question is whether the LP is tighter for unit-weight trees. We show that a slight modification of the trees described above gives the same integrality ratio lower bounds for unit-weight trees. The idea is very simple—recall that in our random construction $\mathbb{T}_n$, edges at level i had weight $1/2^i$; replacing each such edge by a path of 2^{H-i} unit-weight edges does not really change our integrality ratio argument, because the resulting instance $\mathbb{T}'_n$ is essentially equivalent to the instance $\mathbb{T}_n$ with edge weights scaled up by a factor of 2^H. Formally, it is easy to verify that our fractional solution for $\mathbb{T}_n$ naturally yields a fractional solution for $\mathbb{T}'_n$ with value $9H2^H$, and that an optimal integral solution for $\mathbb{T}'_n$ with value OPT' corresponds to an integral solution with value $\mathrm{OPT}'/2^H$ for $\mathbb{T}_n$. Since we know the latter value is at least $\Omega(H^2 \log k)$, we conclude that $\mathrm{OPT}' = \Omega(H^2 2^H \log k)$, and the integrality ratio of $\mathbb{T}'_n$ is $\Omega(H \log k) = \Omega(\log^2 k) = \Omega(\log k \log N / \log\log N)$. Furthermore, the total number of vertices in $\mathbb{T}'_n$ is at most $2^H n = O(n^2)$, and hence the integrality ratio is also $\Omega(\frac{\log^2 n}{(\log\log n)^2})$ in terms of the number of vertices in $\mathbb{T}'_n$.

2.5. Integrality ratio for c-HSTs. Straightforward modifications of our integrality ratio proof for GROUP-STEINER-TREE in 2-HSTs lead to the same lower bounds for c-HSTs, for arbitrary constant $c > 1$. Here, we take an alternative approach; instead of going through the whole proof, we show that our lower bounds for 2-HSTs imply (in a black-box manner) similar bounds for c-HSTs, where $c > 1$ is an arbitrary constant. We first consider the case $c > 2$ and then use it to handle the case $1 < c < 2$.

An arbitrary constant $c > 2$. In this case our 2-HST instance $\mathbb{T}_n$ can be modified into a c-HST instance $\mathbb{T}'_n$ as follows. The set of vertices of $\mathbb{T}'_n$ is a subset of the vertices of $\mathbb{T}_n$. For every $j = 0, 1, 2, \ldots$ iteratively (up to about $\log_c 2^H$), let $i = i(j)$ be the (unique) integer such that $2^i \le c^j < 2^{i+1}$, and include all the level i vertices of $\mathbb{T}_n$ as the level j vertices in $\mathbb{T}'_n$. For example, at $j = 0$, we include the root of $\mathbb{T}_n$ as the root of $\mathbb{T}'_n$ because $1 \le c^0 < 2$. We may assume that the height of $\mathbb{T}_n$ is chosen so that at some iteration j_0, we include in $\mathbb{T}'_n$ the leaves of $\mathbb{T}_n$ (i.e., $i(j_0) = H$), at which point the iterations are stopped. With this assumption (and since all the groups in $\mathbb{T}_n$ contain only leaves), we also get that $\mathbb{T}_n$ and $\mathbb{T}'_n$ have exactly the same k groups. Finally, two vertices at two consecutive levels $j-1, j$ in $\mathbb{T}'_n$ are connected by an edge of weight $1/c^j$ whenever, in $\mathbb{T}_n$, one of the two vertices is an ancestor of the other. For example, the edges incident at the root of $\mathbb{T}'_n$ have weight $1/c$. Notice that $\mathbb{T}'_n$ is a c-HST with height $j_0 \simeq \log_c 2^H = H/\log_2 c$.

A fractional solution LP for $\mathbb{T}_n$, of value LP, say, naturally induces a fractional solution to $\mathbb{T}'_n$ with value at most LP. Indeed, we let the fractional value of an edge connecting a vertex u to its parent v' in $\mathbb{T}'_n$ be equal to the fractional value of the edge connecting (the same vertex) u to its parent v in $\mathbb{T}_n$. It is easy to see that whenever this fractional solution for $\mathbb{T}'_n$ pays (fractionally) $1/c^j$ for an edge, the solution LP

pays $1/2^i$ for the corresponding edge in $\mathbb{T}_n$, with $1/c^j \le 1/2^i$. Since the corresponding edges in $\mathbb{T}_n$ are distinct, the value of the constructed solution for $\mathbb{T}'_n$ is at most LP.

An optimal integral solution OPT′ for $\mathbb{T}'_n$, of value OPT′, say, naturally induces an integral solution INT for $\mathbb{T}_n$ with value at most $O(c) \cdot \text{OPT}'$. Indeed, simply take in $\mathbb{T}_n$ the (minimal) subtree that spans exactly the same leaves (that are spanned by the solution for $\mathbb{T}'_n$). It is easy to see that whenever OPT′ pays $1/c^j$ for an edge connecting a vertex u to its parent v' in $\mathbb{T}'_n$, the solution INT has to pay for the path between u and its ancestor v' in $\mathbb{T}_n$. The total weight of this path is at most $1/2^{i'} + 1/2^{i'-1} + \cdots + 1/2^i = O(1/2^{i'}) = O(c) \cdot 1/2^i = O(c) \cdot 1/c^j$. We thus get an integral solution for $\mathbb{T}_n$ with value $O(c) \cdot \text{OPT}'$.

Combining these arguments with our bounds on the fractional and integral solutions for $\mathbb{T}_n$ yields a lower bound of $\Omega(\frac{1}{c} H \log k)$ on the integrality ratio in c-HSTs. Notice also that the number of vertices in $\mathbb{T}'_n$ is similar to that in $\mathbb{T}_n$ because they have the same leaves. For fixed $c > 2$, we thus get the same integrality ratio lower bounds in c-HSTs as in 2-HSTs, namely, $\Omega(\log^2 k) = \Omega(\log k \log N / \log\log N)$ and also $\Omega(\frac{\log^2 n}{(\log\log n)^2})$ in terms of the number of vertices in $\mathbb{T}'_n$.

An arbitrary constant $1 < c < 2$. Let t be the smallest integer such that $c^t > 2$, and define $q = 1 + c + c^2 + \cdots + c^{t-1}$. The above construction then yields a c^t-HST instance $\mathbb{T}'_n$. Now replace every edge of weight $1/(c^t)^j$ in $\mathbb{T}'_n$ with a path of t edges having weights $1/(qc^{tj-(t-1)}), 1/(qc^{tj-(t-2)}), \ldots, 1/(qc^{tj})$. Clearly, the resulting instance $\mathbb{T}''_n$ is a c-HST. Notice further that the total weight of the above t-path in $\mathbb{T}''_n$ is $(c^{t-1} + c^{t-2} + \cdots + 1)/(qc^{tj}) = 1/c^{tj}$, i.e., equal to the edge in $\mathbb{T}'_n$ it replaced. Hence, any fractional solution for $\mathbb{T}'_n$ yields a fractional solution for $\mathbb{T}''_n$ with the same value, and also an optimal integral solution in $\mathbb{T}'_n$ yields an integral solution for $\mathbb{T}''_n$ with the same value. Since the number of vertices in $\mathbb{T}''_n$ is larger than that in $\mathbb{T}'_n$ by only a constant factor of $t \le \log_c 4 = O(\frac{1}{c-1})$, we get the same integrality ratio lower bounds in c-HSTs as in 2-HSTs, namely, $\Omega(\log^2 k) = \Omega(\log k \log N / \log\log N)$ and also $\Omega(\frac{\log^2 n}{(\log\log n)^2})$ in terms of the number of vertices in $\mathbb{T}'_n$.

2.6. Integrality ratio for directed Steiner tree. The above results immediately lead to a lower bound of $\Omega(\frac{\log^2 n}{(\log\log n)^2})$ on the integrality ratio for DIRECTED-STEINER-TREE. Let $\mathbb{T}_n$ be an instance as described above with $\Omega(\frac{\log^2 n}{(\log\log n)^2})$ integrality ratio for GROUP-STEINER-TREE, and construct a DIRECTED-STEINER-TREE instance as follows. Orient all the edges of $\mathbb{T}_n$ away from the root r. Then introduce new nodes $v_1, v_2, \ldots, v_k$, and for each j and each $u \in g_j$, introduce a zero-weight arc from u to v_j. This defines a DIRECTED-STEINER-TREE instance I which is essentially the same as I: fractional solutions for the two problems map bijectively, with identical total weights, and the same holds also for integral solutions. Observe that the number of vertices in the resulting graph is $n + k \le 2n$, and thus the lower bound of $\Omega(\frac{\log^2 n}{(\log\log n)^2})$ on the integrality ratio for $\mathbb{T}_n$ holds also for I.

2.7. An $O(\log^2 k)$-approximation for the group Steiner tree problem in HSTs. We now show a tight $O(\log^2 k)$-approximation algorithm for GROUP-STEINER-TREE on HSTs. This was obtained jointly with Anupam Gupta and R. Ravi, and we thank them for allowing us to include this algorithm here.

Our algorithm uses the rounding procedure of [GKR00] as a subroutine and takes advantage of the geometrically-decreasing-weights property of HSTs. Let T be the HST instance of GROUP-STEINER-TREE. We assume this tree is a 2-HST; i.e., the weight of each edge in the $(i+1)$st level equals half the weight of its parent edge in

the ith level, and each edge in the first level has weight exactly *one*; the algorithm extends in a simple way to c-HSTs. We also assume that the height of the tree is $H > \log k$; otherwise, the approximation ratio $O(H \log k)$ in [GKR00] already implies the $O(\log^2 k)$ upper bound. It is not difficult to arrange that all members of each group are at the leaves of the HST (with only a constant factor increase in the optimum value). Our algorithm is as follows:

1. Create a new tree T' consisting of only the first $H' = \log k$ levels of T. For each leaf ℓ' in T' find its corresponding node ℓ in level $\log k$ of T, and assign to ℓ' all groups that appear in the subtree rooted at ℓ in T.
2. Run the approximation algorithm of [GKR00] on the GROUP-STEINER-TREE instance T'. Let SOL' be the value of the solution obtained and let OPT' be the value of the optimal solution in T'. The analysis of [GKR00] shows that $\mathrm{SOL}' = O(H' \log k \cdot \mathrm{OPT}') = O(\log^2 k \cdot \mathrm{OPT}')$.
3. From the solution SOL' (which is a subtree of T'), we construct a solution SOL for T as follows:
   (a) Find the subtree S in T that corresponds to SOL', and include S in SOL.
   (b) For each group $g \in \mathcal{G}$, repeat the following steps. Find a leaf ℓ' in SOL' that belongs to g (there must be such a leaf since SOL' covers g), and let ℓ be the level H' vertex in T corresponding to ℓ'. Now find in the subtree under ℓ in T a leaf u that belongs to g (there must be such a leaf because of the way we assigned groups in T'), and add to SOL the path that connects ℓ to its descendant u.

It is easy to verify that the above procedure produces a valid solution SOL to the 2-HST instance T. We claim that $\mathrm{SOL} = O(\log^2 k \cdot \mathrm{OPT})$, where OPT is the optimum solution value in T. Indeed, SOL consists of SOL' and at most k paths (one path per group) added in step 3(b). Because T is an HST, each of these paths has total weight $O(1/k)$ (since its edges have geometrically decreasing weights), and thus, $\mathrm{SOL} = \mathrm{SOL}' + O(1)$. Since the optimal solution must contain at least one edge in the first level (and thus having weight *one*), we get that $\mathrm{OPT} \geq 1$. It is also obvious that $\mathrm{OPT}' \leq \mathrm{OPT}$ and, therefore, $\mathrm{SOL} = \mathrm{SOL}' + O(1) = O(\log^2 k \cdot \mathrm{OPT}') + O(1) = O(\log^2 k \cdot \mathrm{OPT})$.

3. Improved approximations for certain families of trees. To better understand the approximability of GROUP-STEINER-TREE, one may consider the following question: What are the instances (in particular, trees) that are difficult to approximate better than within ratio $O(\log k \log N)$? We partially answer this question by presenting a significantly better approximation ratio for a certain family of trees, which differs from the trees constructed in section 2 in a crucial way.

This improved approximation also sheds light on the instances $\mathbb{T}_n$ constructed in section 2. For example, it may be tempting to believe, at first glance, that the edge weights pose an unnecessary complication to $\mathbb{T}_n$. Notice that the *uniform weight* version of $\mathbb{T}_n$ (i.e., a tree similar to $\mathbb{T}_n$, except that all its edges have the same weight) has the same fractional solutions as $\mathbb{T}_n$. Furthermore, it can be verified that for both $\mathbb{T}_n$ and its uniform weight version, the fractional solution presented in section 2.2 is, with high probability, near-optimal, and that applying to it the rounding procedure of [GKR00] yields an integral solution with value larger by a $\Theta(\log k \log N)$ factor than the relaxation value. However, in the uniform weight version of $\mathbb{T}_n$, the contribution of level-i edges to the relaxation value increases significantly with i, and as we shall soon see, this implies that an approximation ratio better than $O(\log k \log N)$ is possible.

This explains why the weights in $\mathbb{T}_n$ are necessary—they make every level have the same contribution to the relaxation value. This also elucidates the disparity between the performance of a rounding procedure for a relaxation and the integrality ratio of the relaxation—the uniform weight version of $\mathbb{T}_n$ exhibits a large ratio according to the former measure, but a significantly smaller ratio according to the latter.

Technically, fix a GROUP-STEINER-TREE instance T on a tree of height H, and an optimal solution x_e to its flow-based relaxation LP (1). Define z_i^* to be the total contribution of the edges at level i (of T) to the objective function of the relaxation. We show that the relationship between the different z_i^* plays a crucial role in the strength/weakness of the LP: If for some constant $\alpha > 1$ we have $z_{i+1}^* \geq \alpha z_i^*$ for all i, then we can achieve an $O(\log k \cdot \log\log(kN)/\log\alpha) = O(\log k \cdot \log\log(kN))$-approximation. This may suggest that instances with $z_i^* \simeq z_{i+1}^*$ for all or most i are among the worst cases for the relaxation.

The following lemma proves the improved approximation ratio for the case where $\alpha = 2$. The argument easily extends to any constant factor $\alpha > 1$. We sometimes refer to a valid integral solution simply as a cover.

LEMMA 3.1. *If $z_{i+1}^* \geq 2z_i^*$ for all i, then we can find an integral solution of value $O(z^* \cdot \log k \cdot \log\log(kN))$, where $z^* = \sum_{i=1}^{H} z_i^*$ denotes the optimal LP value.*

Before getting into the formal proof, let us outline the main ideas. We separate the tree T into a lower part that contains the lowest $\Theta(\log\log(kN))$ levels of the tree and an upper part that contains the rest of the tree. Let $z^*(U), z^*(L)$ be the contributions of the upper and lower parts to the fractional solution value z^*. Notice that $z^*(U) \leq z^*/\text{polylog}(kN)$ since $z_{i+1}^* \geq 2z_i^*$ for all i. We can thus take care of the upper part as follows. We use the same randomized rounding as in [GKR00] for the upper part, only now we repeat the process about $O(\log N)$ more times (multiplicatively)—this results in a solution that is considerably more expensive with respect to $z^*(U)$ but is still not too much with respect to the total fractional solution z^*. Since we repeat the rounding procedure more times, we cover each group more "times" (in a way that is formalized in the proof). Now every leaf of the upper part that we managed to cover can be regarded, in the lower part, as the root of a subtree. This allows us to apply the algorithm of [GKR00] to some of the subtrees in the lower part, namely, to those subtrees whose root was covered by our upper part solution. By the analysis of [GKR00] we need only pay proportionally to the height of the lower part (times $O(\log k)$); i.e., the lower part solution has value $O(z^*(L)\log k \log\log(kN))$. In the case where $z_{i+1}^* \geq \alpha z_i^*$ for all i, we define the lower part to be the lowest $\Theta(\log\log(kN)/\log\alpha)$ levels of the tree.

Proof. We may assume that all groups contain only leaves of T, by adding zero weight edges. Let L_i be the set of edges at level i. Let $h = 2\log\log(kN)$. Let $U = \{e : e \in L_i \text{ for } i \leq H - h\}$ and $L = \{e : e \in L_i \text{ for } i > H - h\}$. For every $e \in U$, let y_e be x_e rounded upwards to the nearest power of 2, increasing the LP value by a factor of at most 2.

We first find a cover of U, as follows. Let $c_1 > 0$ be a suitably large constant. For every $e \in U$, assign $\hat{x}_e = \min\{1, x_e \cdot c_1 \log k \log^2(kN)\}$, and use one iteration of the rounding scheme presented in [GKR00] (with respect to $\hat{x}_e$) to solve the problem in U. The expected total weight of this solution is at most $O(z^*(U)\log k \log^2(kN)) \leq O(z^* \log k)$, where $z^*(U) = \sum_{i=1}^{H-h} z_i^*$ is the total contribution to z^* of the edges in U.

Using arguments similar to those in [GKR00], we now wish to show that from the perspective of U, every group g is covered "sufficiently many times," with high probability. Let $e_1, \ldots, e_m$ be the leaves of (the subtree induced on) U that "lead" to g

(i.e., g contains at least one of their descendants in T'). A unit amount of flow can be shipped in T from the root to g, under the LP values x_e (as capacities), because x_e is a feasible solution. Let $f_1, \ldots, f_m$ be the corresponding flows on the edges $e_1, \ldots, e_m$. Clearly, $\sum_j f_j = 1$. Partition the m flows, letting $A_i = \{j : \frac{1}{2^i} < f_j \leq \frac{1}{2^{i-1}}\}$. Let $B(g) = \{i : i \leq 2 \log N \text{ and } \sum_{j \in A_i} f_j > 1/4 \log N\}$. It is easy to see that the flow in the remaining sets A_i is at most $(\frac{2}{N^2} + \frac{2}{2N^2} + \cdots) + (2 \log N)\frac{1}{4 \log N} < 3/4$, and thus $\sum_{i \in B(g)} \sum_{j \in A_i} f_j \geq 1/4$. We can therefore ignore the remaining sets and focus on the flows in $B(g)$. Fix $i \in B(g)$ and let V_i be the set of leaves (of U) e_j, for $j \in A_i$, that are chosen by the [GKR00] procedure in U according to $\hat{x}_e$.

For the sake of upper bounding the lower-tail of $|V_i|$, we may assume that the capacity x_e on every edge e equals the total flow shipped along the edge e (since a larger capacity x_e just increases $|V_i|$). Thus, the expectation of $|V_i|$ is $\mu_i = \sum_{j \in A_i} \min\{1, f_j \cdot c_1 \log k \log^2(kN)\}$. If $\frac{c_1}{2^i} \cdot \log k \log^2(kN) \geq 1$, then $|V_i| = |A_i| = \mu_i$ with probability 1. Otherwise, $\mu_i \geq \sum_{j \in A_i} \frac{c_1 f_j}{2} \cdot \log k \log^2(kN) \geq \frac{c_1 \log k \log(kN)}{8}$, and by Janson's inequality [Jan90],

$$\Pr[|V_i| \leq \mu_i/2] \leq e^{-\Omega(\frac{\mu_i}{2+\Delta_i/\mu_i})},$$

where $\Delta_i = \sum_{e \sim e'} \Pr[e \text{ and } e' \text{ are chosen}]$; here, the sum is over pairs of distinct edges $e \in A_i$ and $e' \in A_i$ whose events of being chosen are not independent. By the proofs in [GKR00, KRS02], it is easy to see that $\Delta_i \leq O(\mu_i \log k)$, where the constant in the "$O(\cdot)$" is an absolute constant that is independent of c_1. Thus,

$$\Pr[|V_i| \leq \mu_i/2] \leq e^{-\Omega(\mu_i/\log k)} \leq e^{-\Omega(c_1 \log(kN))},$$

where the constants in the "$\Omega(\cdot)$" are absolute constants that are independent of c_1. There are only k groups, and for each one $|B(g)| \leq O(\log N)$. Thus, by choosing c_1 sufficiently large, we get by the union bound (over a polynomial in kN number of V_i's) that with high probability, for every group g and every $i \in B(g)$, $|V_i| \geq \mu_i/2$. Recall that at least $1/4$ of the total flow in $f_1, \ldots, f_m$ is shipped through sets A_i with $i \in B(g)$ and that the flows among each such set A_i are all equal up to a constant factor. Hence, at least $\Omega(1)$ of the unit amount of flow into g must be shipped using the leaves of U chosen by the [GKR00] procedure.

We next apply the rounding algorithm of [GKR00] to L with the values x_e, starting from every chosen vertex of U. Since we know from above that one can ship to any group g an $\Omega(1)$ amount of flow from the level $H - h$ vertices chosen in the solution for U, we get that x_e satisfies the LP constraints up to a constant factor. It is proved in [GKR00] that after $O(h \log k)$ iterations of the rounding scheme, with high probability all the groups are covered. We now claim that the expected cost of each such iteration is at most z^*. Indeed, the probability to choose an edge e is proportional to its fractional value x_e, and the claim follows by the linearity of expectation.

Therefore, the expected cost of this solution is $O(z^* h \log k + z^* \log k) = z^* \cdot O(\log k \cdot \log\log(kN))$. ∎

4. Discussion. Our results improve the current understanding of the integrality ratio of the flow-based relaxation for the group Steiner tree problem, but some very intriguing gaps still remain. Although for HSTs our $\Omega(\log^2 k)$ lower bound is tight, for general trees there is a slight slack between our $\Omega(\log k \log N / \log\log N)$ lower bound and the $O(\log k \log N)$ upper bound of [GKR00]. Interestingly, an $O(\log^2(kN)/\log\log(kN))$-approximation by a quasi-polynomial time algorithm is

devised in [CEK06]; their algorithm is combinatorial (i.e., not LP-based). Does their algorithm hint that the known upper bound on the integrality ratio in trees is not tight? Or perhaps there is a separation between polynomial and quasi-polynomial (approximation) algorithms? A possible step toward closing this small gap (in the integrality ratio on trees) is to analyze the random instance suggested by Feige (see our description in section 2).

For general graphs, there is an even bigger slack, as the known upper bound is $O(\log n \log k \log N)$ [GKR00, FRT04] and the lower bound is just the lower bound for trees described above. It is worth noting that a significantly better upper bound can be achieved in (general) graphs of small diameter. In particular, an $O(\log k)$ upper bound for expander graphs is shown in [BM04]; this bound is tight since expanders contain a large star metric. We therefore set forth the following question, which was formulated together with Yair Bartal: What is the integrality ratio of Group-Steiner-Tree on the (say two-dimensional) grid graph?

The shortest-path metric of a grid contains, up to constant distortion, an HST which is a complete regular tree (see, e.g., [BBM01]). This tree is similar to our tree $\mathbb{T}_n$ (and to Feige's tree described above), but differs in parameters like arity and edge weight; thus, one may suspect that the integrality ratio in grids is at least as large as in HSTs. In comparison, the best upper bound that we are aware of for two-dimensional grids is the one known for general graphs.

A broader message of our paper is that in some cases the study of inapproximability lower bounds is well served by proving a preliminary integrality gap result. If for no other reason, the proof of an integrality gap might be somewhat easier. Recall that in our case, the hardness of approximation result of [HK03] uses our integrality gap as a "gadget." Recently, in yet another breakthrough along these lines (an integrality gap result that was strengthened into an inapproximability result), an $\Omega(\log^* n)$ hardness was obtained for Asymmetric k-Center [CGH$^+$05]. It would be nice to see how far this paradigm can be taken.

On the other hand, if a proof of integrality gap does not seem to be possible, one should perhaps try to improve the approximation algorithm. For example, for Covering-Steiner (another generalization of Group-Steiner-Tree), a logarithmic term appearing in the approximation ratio of [KRS02] was not known to have a counterpart in the integrality ratio. Recently it turned out that this term is spurious—the approximation algorithm is improved in [GS06] by designing a better rounding procedure.

Acknowledgments. Our work has been influenced by discussions with several people. We thank Moses Charikar, Chandra Chekuri, Uri Feige, Anupam Gupta, Sanjeev Khanna, Adam Meyerson, Seffi Naor, R. Ravi, and Leonid Zosin for very helpful conversations. We also thank the conference and journal reviewers for their very helpful comments.

REFERENCES

[Bar96] Y. Bartal, *Probabilistic approximation of metric spaces and its algorithmic applications*, in Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science, 1996, pp. 184–193.

[Bar98] Y. Bartal, *On approximating arbitrary metrics by tree metrics*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, 1998, pp. 161–168.

[BBM01] Y. Bartal, B. Bollobás, and M. Mendel, *A Ramsey-type theorem for metric spaces and its applications for metrical task systems and related problems*, in Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science, 2001, pp. 396–405.

[BM04] Y. Bartal and M. Mendel, *Multiembedding of metric spaces*, SIAM J. Comput., 34 (2004), pp. 248–259.

[CCC+99] M. Charikar, C. Chekuri, T. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li, *Approximation algorithms for directed Steiner problems*, J. Algorithms, 33 (1999), pp. 73–91.

[CCGG98] M. Charikar, C. Chekuri, A. Goel, and S. Guha, *Rounding via trees: Deterministic approximation algorithms for group Steiner trees and k-median*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, 1998, pp. 114–123.

[CEK06] C. Chekuri, G. Even, and G. Kortsarz, *A greedy approximation algorithm for the Group Steiner problem*, Discrete Appl. Math., 154 (2006), pp. 15–34.

[CGH+05] J. Chuzhoy, S. Guha, E. Halperin, S. Khanna, G. Kortsarz, R. Krauthgamer, and J. Naor, *Asymmetric k-center is* $\log^* n$*-hard to approximate*, J. ACM, 52 (2005), pp. 538–551.

[Fei98] U. Feige, *A threshold of* $\ln n$ *for approximating set cover*, J. ACM, 45 (1998), pp. 634–652.

[FRT04] J. Fakcharoenphol, S. Rao, and K. Talwar, *A tight bound on approximating arbitrary metrics by tree metrics*, J. Comput. System Sci., 69 (2004), pp. 485–497.

[GKR00] N. Garg, G. Konjevod, and R. Ravi, *A polylogarithmic approximation algorithm for the Group Steiner tree problem*, J. Algorithms, 37 (2000), pp. 66–84.

[GS06] A. Gupta and A. Srinivasan, *An improved approximation ratio for the covering Steiner problem,* Theory of Computing, 2 (2006), pp. 53–64.

[HK03] E. Halperin and R. Krauthgamer, *Polylogarithmic inapproximability*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, 2003, pp. 585–594.

[Jan90] S. Janson, *Poisson approximations for large deviations*, Random Structures Algorithms, 1 (1990), pp. 221–230.

[KRS01] G. Konjevod, R. Ravi, and F. S. Salman, *On approximating planar metrics by tree metrics*, Inform. Process. Lett., 80 (2001), pp. 213–219.

[KRS02] G. Konjevod, R. Ravi, and A. Srinivasan, *Approximation algorithms for the covering Steiner problem*, Random Structures Algorithms, 20 (2002), pp. 465–482.

[LY94] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, J. ACM, 41 (1994), pp. 960–981.

[MR95] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.

[RS97] R. Raz and S. Safra, *A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, 1997, pp. 475–484.

[Sri01] A. Srinivasan, *New approaches to covering and packing problems*, in Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, 2001, pp. 567–576.

[ZK02] L. Zosin and S. Khuller, *On directed Steiner trees*, in Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2002, pp. 59–63.

SIAM J. COMPUT.
Vol. 36, No. 6, pp. 1513–1543

ZAPS AND THEIR APPLICATIONS*

CYNTHIA DWORK† AND MONI NAOR‡

Abstract. A *zap* is a 2-round, public coin witness-indistinguishable protocol in which the first round, consisting of a message from the verifier to the prover, can be fixed "once and for all" and applied to any instance. We present a zap for every language in NP, based on the existence of noninteractive zero-knowledge proofs in the shared random string model. The zap is in the *standard* model and hence requires no common guaranteed random string. We present several applications for zaps, including 3-round concurrent zero-knowledge and 2-round concurrent deniable authentication, in the timing model of Dwork, Naor, and Sahai [*J. ACM*, 51 (2004), pp. 851–898], using moderately hard functions. We also characterize the existence of zaps in terms of a primitive called *verifiable pseudorandom bit generators*.

Key words. zero-knowledge, nonmalleable cryptosystems

AMS subject classifications. 94A60, 03D15, 68P25, 38Q10, 68Q15

DOI. 10.1137/S0097539703426817

1. Introduction.

The concept of zero-knowledge, introduced in the groundbreaking paper of Goldwasser, Micali, and Rackoff [36], has proved to be an invaluable tool in the design of cryptographic primitives and protocols. For example, consider an identification protocol based on pseudorandom function evaluation: I am identified by my ability to evaluate a function f_s, where only I know the seed s and there is some form of public commitment to f_s. Given a challenge x, I produce y and prove that $y = f_s(x)$ critically without revealing any information about s.

An appealing and frequently useful relaxation of zero-knowledge, called *witness-indistinguishability*, was proposed by Feige and Shamir [28]. Roughly speaking, in the context of NP, the difference is as follows: an interactive proof system is zero-knowledge if a prover, knowing a witness for membership of a string x in an NP language L, can correctly "convince" a verifier to accept x while revealing no information whatsoever about the witness. If there are two witnesses for $x \in L$, a proof system is witness-indistinguishable if the verifier cannot tell which of the two witnesses is being used by the prover to carry out the proof, even if the verifier knows both witnesses. We restrict our attention to NP because we are interested in the realistic setting in which parties are restricted to probabilistic polynomial-time computations.[1]

In this work we obtain surprising results on the number of rounds needed to achieve zero-knowledge and witness-indistinguishability. For this purpose we introduce and investigate zaps. A zap is a 2-round witness-indistinguishable protocol in which

(i) the first round, consisting of a message from the verifier to the prover, can be

*Received by the editors April 20, 2003; accepted for publication (in revised form) July 5, 2006; published electronically February 9, 2007.
http://www.siam.org/journals/sicomp/36-6/42681.html

†Microsoft Corporation, 1065 L'Avenida, Mountain View, CA 94043 (dwork@microsoft.com). Most of this work was performed while this author was at the IBM Almaden Research Center.

‡Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel (moni.naor@weizmann.ac.il). Some of this work was performed while the author was at the IBM Almaden Research Center and at Stanford University. This author was supported in part by a grant from the Israel Science Foundation and ONR grant N00014-97-1-0505.

[1] The literature on these subjects is extensive. See in particular the papers [18, 27, 33], the lecture notes [19], and the textbooks [29, 43].

fixed "once and for all" and applied to any instance, and
(ii) the verifier uses only *public coins*.

That is, the system remains sound and witness-indistinguishable even if the statements to be proved are chosen after the first-round message is fixed. Thus, if we think of the participating parties as families of *nonuniform*, rather than uniform, probabilistic polynomial-time-bounded Turing machines, the existence of a zap for a language L implies the existence of a 1-message witness-indistinguishable proof system for L.

Throughout the paper we distinguish between the *shared random string model*, in which the parties have access to a common *guaranteed random* string, and what we call the *standard* model, in which no such assumption is made. Whenever we refer to noninteractive zero-knowledge proofs (NIZKs), we mean in the shared random string model (the definition of NIZK forces a shared object). We present zaps for every language $L \in \mathrm{NP}$ based on the existence of a NIZK system for L in the shared random string model. The zap is in the standard model and hence requires no common guaranteed-random string. Using current NIZK technology, this means that zaps can be based on any family of enhanced certified trapdoor permutation [30].

Not only can zaps be constructed from NIZKs, but the converse holds as well: if every language in NP has a zap and one-way functions exist, then every language in NP has a NIZK. In fact, the NIZKs we obtain from zaps are zero-knowledge against adaptive selection of the theorem to be proved. This yields a proof that if NIZKs secure against nonadaptive selection exist and one-way functions exist, then adaptive NIZKs exist.

This result (and its proof) gives a somewhat formal view of zaps but yields little intuition for why zaps and NIZKs exist at all. Indeed, our first constructions of zaps were not based on NIZKs, but relied on the new notion of a *verifiable pseudorandom bit generator* (VPRG). Roughly speaking, a pseudorandom sequence is *verifiable* if a party knowing the pseudorandom seed can construct verifiable proofs of the bits of the pseudorandom sequence. Moreover, a VPRG with some number k of output bits passes what we call the ith bit test for all $1 \leq i \leq k$: given proofs of the values of all but the ith bit in the sequence, it is computationally infeasible to guess the ith bit with a nonnegligible advantage over $1/2$. Thus, VPRGs can be viewed as a special case of the verifiable pseudorandom functions (VPRFs) of Micali, Rabin, and Vadhan [45], in which the domain is very small. We give constructions for VPRGs and a relaxation, *approximate VPRGs*.

The importance of VPRGs is this: Zaps (and NIZKs) exist if and only if approximate VPRGs exist in the standard model. In this paper we construct VPRGs using multiple certified trapdoor permutations with a common domain; this yields the first NIZK construction for which the trapdoor permutations need not be enhanced. In addition, recent constructions of VPRFs based on assumptions on bilinear maps [44, 16, 17] also necessarily yield NIZKs (and zaps).

1.1. Applications of zaps. We present applications of zaps in several models. Specifically, we construct faster implementations of important cryptographic primitives in each of the standard, timing-based, and resettable models. Although in some cases the absolute improvement in rounds may be modest, the number of rounds that we achieve in each case is within one of the best possible. For example, all previous witness-indistinguishable proof systems require at least *three* rounds of communication, while zaps achieve witness-indistinguishability in two rounds. The fact that zaps also yield nonuniform 1-round witness-indistinguishability suggests that *proving* a lower bound of two rounds is unlikely. (See also the recent work of Barak, Ong, and Vadhan [3].)

An interesting set of applications for zaps is in the *timing* model of Dwork, Naor, and Sahai [23], where, using moderately hard functions [20] and timed commitments [11], we obtain 3-round *concurrent* black-box[2] zero-knowledge proofs of knowledge for all of NP. A 3-round black-box zero-knowledge protocol with timing (even without concurrency) is interesting in its own right: it is known that in the standard model (no timing) this is impossible to achieve (with negligible soundness error assuming NP $\not\subset$ BPP) [32], while the possibility of concurrency implies that at least $\Omega(\log n/\log\log n)$ rounds are required [14]; thus, adding timing allows us to go well below the lower bounds in the standard model. Recently, using zaps, Dwork and Stockmeyer obtained 2-round timing-based black-box (concurrent) zero-knowledge interactive proofs under the assumption that certain functions have no fast *auditors*; they also provide a prover-advice–based variant for which soundness is absolute (in this variant the prover can have arbitrary computation time) [25]. We note that even in the timing-based model, zero-knowledge proof systems for languages outside of BPP require two rounds of interaction. No such result is known for the bounded-advice model.

Still more recently, Barak and Pass obtained 1-round *weak* zero-knowledge arguments, under (less) nonstandard assumptions [4]. Under the weakened definition, soundness holds only against uniform probabilistic polynomial-time cheating provers, and the zero-knowledge condition is obtained using a simulator that runs in quasi-polynomial (rather than polynomial) time.

We also use zaps to construct 2-round deniable authentication protocols [18, 21, 23, 24]. Intuitively, deniable authentication is like a signature scheme in that it permits one party to authenticate messages to another party, based on a public key; however, unlike in the case of digital signatures, the authenticating conversation leaves no trace, for example, it may be simulatable and hence can be *effectively* repudiated.

The relative ease with which we are able to reduce the amount of interaction provides further motivation for the timing model of [23]—in our opinion a more realistic one than the shared guaranteed random string model (see, e.g., [15])—and a complexity theory of moderately hard functions [20].

Using zaps and timed commitments we also obtain a different type of improvement on the results in [23, 24]. The timing model requires a mild (α, β) assumption about the relative rates of the clocks of nonfaulty processors, and the protocols in [23] require processors (typically, the prover), to wait until an interval of at least $\beta \geq \alpha$ time has elapsed (as measured on the processor's own clock). α and β are chosen so as to tolerate actual system and communication delays. The proofs in [23, 24] require the parameters to be set according to the slowest nonfaulty processors. Our new techniques permit flexibility in this respect: fast verifiers with good communication links to the prover are not forced to suffer delays due to slower concurrent verifiers.

In the standard model, without timing assumptions, we give a 2-round oblivious transfer protocol based on the quadratic residuousity assumption and using public keys; without previously established public keys the protocol requires three rounds.

Finally, we consider a model of computation in which the prover's use of randomness is severely restricted, as, for example, in the case of a smart card, in which the prover may have a short embedded truly random seed and read-only memory. Canetti et al. [13] give one formalization, termed *resettable* zero-knowledge (rZK). Informally, a protocol protects a witness (either in the zero-knowledge sense or in the indistin-

[2] A protocol is black-box zero-knowledge if there is a universal simulator, which when given "black-box" access to any verifier strategy is able to simulate an interaction of that verifier with the prover. Virtually all zero-knowledge proofs until very recently were black-box (but see [1] for an example of a protocol which does not fit this category).

guishability sense) in the resettable model if the protection holds even if the prover may be restarted (reset) many times and forced to repeatedly use the same random tape. (The prover may also be restarted using a different, but still random, tape.)

Using zaps and timed commitments, we construct a 3-round timing-based rZK proof system for any language in NP. As noted in [13], rZK proofs *cannot* be proofs of knowledge, so, despite the connections between the smart-card setting as described above, resettable, and concurrent zero-knowledge [13, 38], this result is incomparable with our 3-round concurrent-ZK proofs of knowledge.

We also observe that 2-round (and even nonconstructive 1-round) resettable witness-indistinguishability is easily obtained from a zap, simply by having the prover's random bits in the zap be a pseudorandom function of the verifier's initial message and the input. This improves (both in conceptual and round complexity) upon the 5-round resettable witness-indistinguishability results in [13].

In all our protocols that employ timing, only the verifier needs access to a (local) clock. This is particularly appealing in the resettable case, in which the prover may be a smart card, since the card may not be equipped with a clock.

1.2. Outline. In section 2 we review the definitions of known cryptographic primitives. A formal definition of a zap is given in section 3. In section 4 we prove the existential equivalence of zaps (in the standard model) and NIZKs (in the shared random string model). Section 5 defines and constructs VPRGs and approximate VPRGs, together with a proof that zaps (and hence, by the above-mentioned result, NIZKs) exist if and only if approximate VPRGs exist in the standard model. Section 6 contains our zap-based oblivious transfer protocol. In section 7 we discuss the timing-based applications (3-round concurrent zero-knowledge and 2-round deniable authentication). In section 8 we discuss uses of zaps in the resettable model of [13]. Finally, open questions are discussed in section 9.

2. Brief review of cryptographic primitives. We now review the cryptographic primitives used in this paper. For the standard ones we follow Goldreich [29]. Throughout this paper, unless otherwise noted, all "good" parties (the nonfaulty prover and verifier) are uniform probabilistic polynomial-time Turing machines. However, our protocols remain sound regardless of the computational power of the prover, and we achieve zero-knowledge against nonuniform probabilistic polynomial time cheating verifiers. (This is assuming the classical underlying primitives are secure against nonuniform adversaries. Security against nonuniform adversaries is not essential to our work and we chose to express it this way for simplicity.)

In general we will be using n as our security parameter and the input length, but in some places we will also be using k_s to denote the length of the input to a cryptographic primitive which is sufficient for obtaining hardness, for instance, a one-way function or a trapdoor permutation. In general n and k_s are polynomially related. While we do not emphasize efficiency in this paper (rather our aim is to point out feasibility of various constructions), we prefer to have two parameters for future comparisons. Let $\nu(n)$ denote a function that grows more slowly than the inverse of any polynomial, i.e., for all $c > 0$ there is an n_0 such that $\nu(n) < 1/n^c$ for all $n \geq n_0$. We say such a $\nu(\cdot)$ is *negligible*. We use the term *with overwhelming probability* to mean with probability that is at least $1 - \nu(n)$ for negligible ν.

2.1. Witness-indistinguishability. The concept of *witness-indistinguishability* was proposed by Feige and Shamir [28] as a relaxation of zero-knowledge. Unlike the case with zero-knowledge, witness-indistinguishability is closed under parallel and

concurrent composition. Let L be an NP language accepted by a nondeterministic polynomial-time Turing machine M_L. A computation path is a sequence of nondeterministic choices made by M_L. The set of accepting computation paths on input $x \in L$ is the *witness set* of x, denoted $w(x)$.

DEFINITION 2.1 (witness-indistinguishability). *A proof system (P, V) for language L is witness-indistinguishable if for any polynomial time V', for all $x \in L$, for all $w_1, w_2 \in w(x)$, and for all auxiliary inputs z to V', the distribution on the views of V' following an execution $(P, V')(x, w_1, z)$ is indistinguishable from the distribution on the views of V' following an execution $(P, V')(x, w_2, z)$ to a nonuniform probabilistic polynomial-time distinguisher receiving one of the above transcripts as well as (x, w_1, w_2, z).*

Note that the auxiliary input z can even be the two witnesses w_1, w_2. Thus, *even knowing both witnesses*, V' should not be able to distinguish which witness is being used by P.

THEOREM 2.2 (see [28]). *Every zero-knowledge protocol is witness-indistinguishable.*

THEOREM 2.3 (see [28]). *Witness-indistinguishability is preserved under parallel and concurrent composition of protocols.*

2.2. Noninteractive zero-knowledge proof systems. The following discussion is based on [18, 27, 49]: a (single-theorem) noninteractive proof system for a language L allows one party P to prove membership in L to another party V for any $x \in L$. P and V initially share a string σ, of length polynomial in the security parameter n, which is trusted to have been chosen at random. To prove membership of a string x in $L_n = L \cap \{0,1\}^n$, P sends a message π as a proof of membership. V decides whether to accept or to reject the proof π as function of x and σ. Noninteractive zero-knowledge proof systems were introduced in [8, 7]. Noninteractive zero-knowledge schemes for proving membership in any language in NP may be based on any enhanced certified trapdoor permutation. (See [27, 40] and [30] for a discussion of enhancement.) As for the complexity of the NIZKs, assuming a trapdoor permutation on k_s bits, the length of a proof of a satisfiable circuit of size M (and the size of the shared random string) is $O(Mk_s^2)$.

We assume that the shared string σ is generated according to the uniform distribution on strings of length polynomial in the security parameter n, where the polynomial depends on the particular protocol. The running time of the verifier is also polynomial in n.

Let L be in NP and for any $x \in L$, $n = |x|$, let $w(x)$ be the set of strings that witness the membership of x in L, as described above. For the proof system to be of any use, P must be able to operate in polynomial in n time if it is given a witness $w \in w(x)$. We call this the *tractability* assumption for P. In general w is *not* available to V.

Let $PN(x, w, \sigma)$ be the distribution of the proofs generated by P on input x, witness w, and shared string σ. Suppose that P sends V a proof π when the shared random string is σ. Then the pair (σ, π) is called the *conversation*. Any $x \in L$ and $w \in w(x)$ induces a probability distribution $CONV(x, w)$ on conversations (σ, π), where σ is a shared string and $\pi \in PN(x, w, \sigma)$ is a proof.

For the system to be zero-knowledge, there must exist a simulator Sim which, on input x, generates a conversation (σ, p). Let $Sim(x)$ be the distribution on the conversations that Sim generates on input x, let $Sim_U(x) = Sim_U$ be the distribution on the σ part of the conversation, and let $Sim_P(x)$ be the distribution on the proof

component. In the definitions of [7, 27] the simulator has two steps: it first outputs Sim_U without knowing x, and then, given x, it outputs $Sim_P(x)$.

DEFINITION 2.4. *A pair of probabilistic polynomial-time machines (P, V) with shared random string σ is a* noninteractive zero-knowledge proof system *for the language $L \in NP$ if*

- *Completeness: For all $x \in L_n$, for all $w \in w(x)$ and for random σ, with overwhelming probability over $\pi \in_R PN(x, w, \sigma)$, we have that V accepts on input (σ, x, π). The probability is over the choice of the shared string σ and the internal coin flips of P.*
- *Soundness: For all $y \notin L$ we have that $\Pr_\sigma[\exists \pi' \in \{0,1\}^*$ such that (s.t.) V accepts $(\sigma, y, \pi')]$ is negligible. Note that the probability is* only *over the choices of the shared string σ.*
- *Zero-knowledge: There is a probabilistic polynomial-time machine Sim which is a simulator for the system: for all nonuniform polynomial time distinguishers $\mathcal{T}$, for all nonnegligible $\nu(\cdot)$, for all sufficiently large $x \in L$, and $w \in w(x)$,*

$$\left|\Pr[\mathcal{T}(s, x, w) = 1 | s \in_R Sim(x)] - \Pr[\mathcal{T}(s, x, w) = 1 | s \in_R \mathcal{CONV}(x, w)]\right| \leq \nu(n),$$

where the probability space is taken over the random choices of σ and over the random choices of the Sim and $\mathcal{P}$.

Remark 2.5. This definition of NIZK does not require that the system be *sound* if the instance x is adaptively chosen, that is, selected after the public random string is known. Nevertheless, it is sufficiently strong for our purposes; also it is easy to reduce the soundness error in NIZK by parallel repetition. Similarly, we do not assume zero-knowledge against adaptive choice of x. As we will see in Corollary 4.4, going through zaps allows us to transform any NIZK satisfying Definition 2.4 into one that allows adaptive selection of x.

As shown in [27], any NIZK satisfying Definition 2.4 is also *general witness-indistinguishable* in the following sense.

CLAIM 2.1 (see [27]). *Any NIZK for a language L in NP is general witness-indistinguishable; that is, for all polynomial distinguishers $\mathcal{T}$ for a random string σ, for any (nonadaptively chosen[3]) sequence $\{(x_i, w_i^1, w_i^2)\}_{i=1}^m$ chosen by $\mathcal{T}$, where $x_i \in L_n$ and $w_i^1, w_i^2 \in w(x_i)$ for all $1 \leq i \leq m$, we have*

$$\left|\Pr[\mathcal{T}(\pi_1^1, \pi_2^1, \ldots, \pi_m^1) = 1] - \Pr[\mathcal{T}(\pi_1^2, \pi_2^2, \ldots, \pi_m^2) = 1]\right| < \nu(n),$$

where for all $1 \leq i \leq m$ and $b \in \{0,1\}$ we let $\pi_i^b \in_R PN(x_i, w_i^b, \sigma)$. The probability space is over $\mathcal{P}$'s and $\mathcal{T}$'s random coins and the choice of σ.

Note that general witness-indistinguishability implies witness-indistinguishability even if $x_1 = \cdots = x_m$, which will be the case of interest here.

2.3. Deniable authentication. A *public key authentication* scheme permits an authenticator $A\mathcal{P}$ to convince a second party V, only having access to $A\mathcal{P}$'s public key, that $A\mathcal{P}$ is willing to authenticate a message m. However, unlike in the case of digital signatures, deniable authentication does not permit V to convince a third party that $A\mathcal{P}$ has authenticated m—there is no "paper trail" of the conversation (say, other than what could be produced by V alone). Thus, deniable authentication

[3] If the NIZK is nonadaptive, then the claim refers to nonadaptively chosen sequences; if the NIZK is adaptive, then the claim also holds for adaptively chosen sequences. In our case, we have assumed the weaker NIZK.

is incomparable with digital signatures. Deniable authentication first appeared in [18, 21] and was formalized in [23] (see also [24]). Several 4-round timed concurrent deniable authentication protocols are given in [23, 24].

The authentication protocol should satisfy the following.

- Completeness: For any message m, if the prover and verifier follow the protocol for authenticating m, then the verifier accepts.
- Soundness—existential unforgeability against concurrent chosen message attack: Suppose that the copies of $A\mathcal{P}$ are willing to authenticate any polynomial number of messages $m_1, m_2, \ldots$, which may be chosen adaptively and in a concurrent manner by an adversary $\mathcal{A}$ who also controls the verifier V'. We say that $\mathcal{A}$ successfully attacks the scheme if a forger C, under control of $\mathcal{A}$ and pretending to be $A\mathcal{P}$, succeeds in authenticating to a third-party D (running the protocol of the original verifier V) a message $m \neq m_i$, $i = 1, 2, \ldots$. The soundness requirement is that all probabilistic polynomial-time $\mathcal{A}$ will succeed with at most negligible probability.
- Zero-knowledge—deniability: Consider an adversary $\mathcal{A}$ as above and suppose that the copies of $A\mathcal{P}$ are willing to authenticate any polynomial number of messages. Then for each $\mathcal{A}$ there exists a polynomial-time simulator that outputs an indistinguishable transcript from the one $\mathcal{A}$ produces from its interaction with $A\mathcal{P}$.

  Two natural variants are (1) the distinguisher has access to the secret authentication key and (2) the distinguisher does not have access to the secret authentication key. The first best captures our intuitive notion of deniable authentication, since even obtaining access to the key, say, via legal compulsion, will not destroy the deniability.

2.4. Security of encryption. We will need public-key cryptosystems for two of our applications: resettable zero-knowledge (section 8.2) and deniable authentication (section 7.2). The security requirements of these two applications are different. To specify the security of an encryption scheme, one must describe the power of the attacker in terms of access to the system (chosen plaintext, chosen ciphertext) and what it means to break the system (semantic-security, nonmalleability). See [18] or [5] for a discussion of notions of security. The deniable authentication application requires a system that is nonmalleable against chosen-ciphertext attacks in the postprocessing mode (called CCA-2 in [5]). The resettable zero-knowledge application requires semantic security against chosen plaintext attacks. (There are some other requirements from the encryption scheme which transcend security.)

2.5. Using time in the design of protocols. Dwork, Naor, and Sahai [23] have shown the power of time in the design of zero-knowledge protocols through the use of an (α, β) assumption. This says that all good parties are assumed to have clocks that satisfy the (α, β)-*constraint* (where $\alpha \leq \beta$): for any two (possibly the same) nonfaulty parties P_1 and P_2, if P_1 measures α elapsed time on its local clock and P_2 measures β elapsed time on its local clock, and P_2 begins its measurement in real time after P_1 begins, then P_2 will finish after P_1 does.

The protocols in [23, 24] use time in two explicit ways: (i) delays—one party must delay the sending of some message until at least some specified time β has elapsed on its local clock; (ii) time-outs—one party requires that the other deliver its next message before some specified time α has elapsed on its (first party's) local clock. In this work we are able to eliminate the use of delays; the protocols use only time-outs. Furthermore, we do not require a *global* (α, β)-constraint, rather each instantiation of

the protocol can fix its own values based on the local characteristics of the network. An essential ingredient of our protocols is the *implicit* use of time via *moderately hard functions* [20]. In particular, we use *timed commitments with verifiable recovery,* described next.

Timed commitment. A string commitment protocol allows a sender to commit, to a receiver, to some value. The protocol has two phases. At the end of the *commit* phase the receiver has gained no information about the committed value, while after the *reveal* phase the receiver is assured that the revealed value is indeed the one to which the sender originally committed. Timed commitments, defined and constructed by Boneh and Naor [11], are an extension of the standard notion of commitments in which there is a potential forced opening phase permitting the receiver, by computation of some moderately hard function, to recover the committed value without the help of the committer. The price paid in terms of security is that the committed value is hidden for only a limited amount of time.

DEFINITION 2.6. *A* (T, t, ϵ) timed commitment *scheme for a string* $y \in_R \{0,1\}^n$ *enables Alice to give Bob a commitment* C *to the string. At a later time, Alice can prove to Bob that* C *represents a commitment to the value* y*. However, if Alice refuses to reveal the value of* C*, then Bob can spend time* T *to forcibly retrieve this value. Alice is assured that within time* t *on a parallel machine with polynomially many processors, where* $t < T$*, Bob will succeed in obtaining* y *with probability at most* ϵ*. Formally, a* (T, t, ϵ) *timed commitment scheme consists of three phases:*

- *Commit phase: To commit to a string* $y \in \{0,1\}^n$ *Alice and Bob execute a protocol whose outcome is a* commitment string $C = TC(y)$ *which is given to Bob.*
- *Open phase: At a later time Alice may reveal the string* y *to Bob. They execute a protocol so that at the end of the protocol Bob has a proof that* y *is the committed value.*
- *Forced open phase: Suppose Alice refuses to execute the* open *phase and does not reveal* y*. Then there exists an algorithm, called* forced-open*, that takes the commitment string* C *as input and outputs* y *and a proof that* y *is the committed value. The algorithm's running time is at most* T*.*

The commitment scheme must satisfy a number of security constraints:

- *Binding: During the open phase Alice cannot convince Bob that* C *is a commitment to* $y' \neq y$*. That is, binding is absolute, independent of computational power: there is at most one "de-commitment,"* y*, consistent with* $TC(y)$*.*
- *Soundness: At the end of the* commit *phase Bob is convinced that, given* C*, the* forced-open *algorithm will produce the committed string* y *in time* T*.*
- *Privacy: Every PRAM algorithm* $\mathcal{A}$ *whose running time is at most* t *for* $t < T$ *on polynomially many processors will succeed in distinguishing* y *from a random string, given the transcript of the commit protocol as input, with advantage at most* ϵ*. In other words,*

$$\left|\Pr[\mathcal{A}(\textit{transcript}, y) = \textit{"yes"}] - \Pr[\mathcal{A}(\textit{transcript}, R) = \textit{"yes"}]\right| < \epsilon,$$

where the probability is over the random choice of y *and* R *and the random bits used to create* C *from* y *during the commit phase.*

The important requirements of timed commitments are (i) the future recoverability of the committed value is verifiable: if the commit phase ends successfully, then the receiver is correctly convinced that forced opening will yield the value; (ii) forcibly

recovered values and decommitments are verifiable: the receiver obtains not only the value but also a proof of its validity, so that anyone who has the commitment (or the transcript of the commit phase) can verify the value *without going through a recovery process*, i.e., in fixed amount of time; (iii) the commitment is immune to parallel attacks, i.e., even if the receiver has much more computing power than assumed, it cannot recover the value substantially more quickly than a single-processor receiver. We denote by T the bound on the time below which it is safe to assume that the timed commitment cannot be broken with nonnegligible probability, even by a PRAM.

Specifically, we are interested in timed commitment schemes with the following structure. The committer sends to the receiver a string ζ, which constitutes the commitment. For every "valid" commitment ζ, it is possible, through moderately hard computation, to recover a pair (y, π) such that π is an easily checked witness to the fact that ζ is a commitment to y. The set of valid commitments is in NP: for every valid commitment ζ there is a witness to the statement "ζ is a valid commitment to a string that can be recovered through the forced recovery process." Finally, the forced recovery time is relatively large compared to the time of all other operations in the protocol (such as constructing ζ, verifying a correctly decommitted value, verifying future recoverability, etc.). Thus, we think of all other operations as "easy" while recovery is "moderately hard." The scheme in [11] has this structure and properties, albeit based on a nonstandard assumption regarding the computation of number of the form g^{2^k} mod N for composites N without knowing the factorization of N.

2.6. Oblivious transfer. In a 1-*out-of*-2 *oblivious transfer (OT)* protocol, one party, the *sender*, has two strings (M_1, M_2) as its input, and the second party, the *chooser*, has a bit b. The chooser should learn M_b and nothing regarding M_{1-b}, while the sender should gain no information about b. 1-out-of-2 OT was suggested by Even, Goldreich, and Lempel [26] as a generalization of Rabin's "oblivious transfer" [50].

3. Formal definition of a zap. A *zap* is a 2-round (2-message) protocol for proving membership of $x \in L$, where L is a language in NP. Let the first-round (verifier to prover) message be denoted ρ and the second-round (prover to verifier) response be denoted π satisfying the following conditions:

- Public coins: There is a polynomial $k(\cdot)$ such that the first-round messages form a distribution on strings of length $k(n)$ which depends only on the *length* n of x. The verifier's decision whether to accept or reject is a polynomial-time function of x, ρ, and π only.
- Completeness: Given x, a witness $w \in w(x)$, and a first-round ρ, the prover, running in time polynomial in $|x|$, can generate a proof π that will be accepted by the verifier. Note that this is *perfect completeness*. We can relax this requirement and ask the prover to succeed with overwhelming probability over the choices made by the prover and the verifier.
- Soundness: With overwhelming probability over the choice of ρ, there exists no $x' \notin L$ and second-round message π such that the verifier accepts (x', ρ, π).
- Witness-indistinguishability: Let $w, w' \in w(x)$ for $x \in L$. Then for all ρ, the distribution on π when the prover has input (x, w) and the distribution on π when the prover has input (x, w') are nonuniform probabilistic polynomial-time (in $n = |x|$) indistinguishable, even given both witnesses w, w'.

It follows immediately from the definitions that every zap yields a *nonconstructive* nonuniform single-round witness-indistinguishable protocol; that is, the first-round message can be fixed once and for all. Also since we require unconditional soundness

(soundness against unbounded provers) the private coins versus secret coins really does not show up.

CLAIM 3.1. *Every zap yields a* nonconstructive *nonuniform single round witness-indistinguishable protocol: for each n, there exists a string $\hat{\rho}_n$ such that, letting $L_n = L \cap \{0,1\}^n$, the following hold:*

1. *Given $x \in L_n$ and a witness $w \in w(x)$, the prover can generate a proof π that will be accepted by the verifier. Moreover, the verifier's decision whether to accept or reject is a polynomial-time function of $x, \hat{\rho}_n$, and π.*
2. *There exists no $x' \notin L_n$ and message π such that the verifier accepts $(x', \hat{\rho}_n, \pi)$.*
3. *For all $x \in L_n$ and all $w, w' \in w(x)$, the distributions $\mathcal{P}(x, w, \hat{\rho}_n)$ and $\mathcal{P}(x, w', \hat{\rho}_n)$ are indistinguishable by any nonuniform probabilistic polynomial-time distinguisher.*

Comparison with NIZKs. Zaps differ from NIZKs in two respects, making the two concepts incomparable. On the one hand, zaps do not require that the prover and verifier have access to a common guaranteed random string. On the other hand, NIZKs provide more provable protection of the witness than do zaps, since NIZKs can be simulated without access to the witness while zaps provide no such guarantee.

4. The NIZK-based construction. Assume we have a NIZK system (in the shared string model) satisfying Definition 2.4 for a language L. We will construct a zap for L (in the standard model). We will first provide some intuition for the construction. Consider a NIZK in the shared string model; we try to convert it into a zap by somehow generating the shared string σ. Suppose we let the verifier choose a string B and fix $\sigma = B$. The danger with this approach is that there may be "bad" choices for σ that leak information about the witness, and the verifier might choose B to be one of them, thus harming the witness protection. If, to compensate, we have the prover choose its own random string C and we set $\sigma = B \oplus C$ (that is, σ is the bitwise exclusive-OR of B with C), then the danger is that the prover will use the simulator to come up with a σ' that "proves" that $x \in L$ (that is, causing V to accept x), even for $x \notin L$. The prover could then set $C = \sigma' \oplus B$, violating soundness.

The actual protocol strikes a balance between these two ideas: a NIZK is repeated many times in parallel, but not quite independently, as follows. The jth instance has common string σ_j, defined to be the bitwise exclusive-OR of two strings, one chosen by the prover and the other chosen by the verifier. The verifier's choice for the jth instance may be any string B_j; however, the prover may choose only a single string C that is used in all instances. This sort of idea can be called reverse randomization and has been previously used in the bit commitment protocol of Naor [46] and can be traced back to Lautemann's proof that BPP is in the second level of the hierarchy [42]; it has since been applied by Dwork, Naor, and Reingold [22] for removing decryption errors.

Choice of parameters (general construction). We now specify the parameters we need. Note that in general it is possible to reduce the soundness error of a NIZK by repetition (with a fresh part of the shared random string for each repetition) without hurting the zero-knowledge properties. Note that parallel repetition reduces the error in a straightforward manner here, since it is combinatorial. The price of course is in the string and proof length.

Assume that we have a NIZK for L which, for proving membership of strings of length $|x|$, with security parameter n, uses a common shared string of length $\ell = \ell(n, |x|)$. Assume further that on any input $y \notin L$ of length $|x|$ the NIZK errs with probability at most $q = q(n)$ over the choice of the common random string σ. In (4.1),

$k = k(n, |x|) = |\rho|$, the number of random bits sent by the verifier in the first-round message. The number of copies $m = m(n, |x|)$ of the NIZK will be k/ℓ. To achieve soundness guarantee δ for the zap (that is, a cheating prover should succeed with probability at most δ), we choose k satisfying

$$q^{k/\ell} < \frac{\delta}{2^{|x|+\ell}}. \tag{4.1}$$

4.1. Protocol Z: A zap. To achieve soundness against an arbitrarily powerful prover and yet to require only feasible computation from the "good" prover, we must assume the existence of a NIZK with these properties, such as the systems in [27, 40].

Let $x \in L$ be an NP-statement to be proved to the verifier. We do not need x to be fixed before execution of the protocol begins. Let w be the witness to $x \in L$ known by the prover, let n be the security parameter, and let $PN(x, w, \sigma)$ be the distribution on messages sent in the NIZK by a (noncheating) prover when the common random string is σ of length $\ell(n, |x|)$. For simplicity, in the remainder of this discussion we assume n and $|x|$ are related by some fixed polynomial so that it suffices to think of $\ell(n, |x|)$ as a function solely of n. Let $k = k(n)$ and $m = m(n)$ satisfy (4.1).

First round: $V \longrightarrow P$. The verifier sends to the prover a random k-bit string $\rho = b_1 \dots b_k$, which is interpreted as $B_1 \dots B_m$, where B_j denotes the jth block of ℓ consecutive bits and ℓ is the length of the common random string used by the NIZK.

Second round: $P \longrightarrow V$. The prover responds as follows. First, it chooses a random ℓ-bit string $C = c_1 \dots c_\ell$. For $j = 1 \dots m$ define σ_j to be the bitwise exclusive-OR of B_j and C:

$$\sigma_j = B_j \oplus C.$$

Then the prover sends to the verifier x, C, and the m noninteractive proofs

$$\{\pi_j \in_R PN(x, w, \sigma_j)\}_{j=1\dots m}.$$

Final check. The verifier V checks that each of the m NIZKs $\pi_1, \pi_2, \dots, \pi_m$ with common strings $\sigma_1, \sigma_2, \dots, \sigma_m$, where $\sigma_j = C \oplus B_j$ results in acceptance; if so, then the verifier accepts the zap; otherwise the verifier rejects. This completes the description of Protocol Z.

LEMMA 4.1. *Protocol Z is sound; moreover, for all n, there exists a choice $\hat{\rho_n} = \hat{b}_1 \dots \hat{b}_{k(n)}$ for the first-round message that yields perfect soundness: for all $x \notin L_n$ for all π $V(x, \hat{\rho_n}, \pi)$ rejects.*

Proof. Let $\ell = \ell(n)$ and $k = k(n)$. Fix an $x \notin L$ and random bit string $C = c_1 \dots c_\ell$. Recall that in a NIZK the faulty prover's probability of succeeding on an $x \notin L$ is a function of the common random string only, and this probability is at most q. We will show that with overwhelming probability, over the choice of $b_1, \dots, b_k$, the prover will fail to convince the verifier to accept x. The key point is that once everything but the b has been fixed, the σ_j are truly random—because the B_j are. Therefore each copy of the NIZK proof has probability at most q of failing to cause rejection. Since each proof is independent (because the random b_i used in each copy of the NIZK proof are independent), the overall probability that all $m = k/\ell$ copies fail to reject is at most q^m.

The number of possible assignments to pairs (c, x) where $c \in \{0,1\}^\ell$ and $x \notin L$ is at most $2^{\ell+|x|}$. Hence, as long as

$$2^{\ell+|x|} q^m = 2^{\ell+|x|} q^{k/\ell} \leq \delta$$

(which is guaranteed by our choice of k in (4.1)), the probability over $b_1, \ldots, b_k$ that there even exists a "bad" choice of $c_1 \ldots c_\ell$, an $x \notin L$, and a zap π that erroneously causes the verifier to accept x is at most δ (cf. the soundness requirement in Definition 2.4). Since $\delta < 1$, there must exist some $\hat{\rho_n} = \hat{b}_1 \ldots \hat{b}_k$ that provides soundness against all $x \notin L_n$: for all $x \notin L_n$ for all π $V(x, \hat{\rho_n}, \pi)$ rejects. □

LEMMA 4.2. *Protocol Z is witness-indistinguishable.*

Proof. We prove witness-indistinguishability *for every* ρ. We will be using only the witness-indistinguishability property of the proof system (Theorem 2.2). Thus, fix an arbitrary ρ for the entire proof. We will carry out a standard hybrid argument with the following steps along the chain. Let w and w' be two witnesses that $x \in L$, and let $n = |x|$. At one extreme of the chain the witness w is used in each of the m NIZKs; at the other extreme the witness w' is used in every copy. At each step along the chain we increase by one the number of copies of the NIZK in which w' is used (and decrease the number in which w is used).

Let $\langle w, w', i \rangle$ denote the distribution on transcripts in which the first i copies of the NIZK are constructed using w and the remaining $m - i$ copies are constructed using w'. The transcripts also include x, w, w'. The distribution is over random choices made by the prover (since ρ is fixed). Let $\mathcal{T}$ be a nonuniform polynomial-time test that takes as input a transcript and outputs a single bit. We write $\mathcal{T}(\langle w, w', i \rangle)$ to denote $\mathcal{T}$'s behavior on a transcript chosen uniformly from $\langle w, w', i \rangle$.

Assume for the sake of contradiction that there exists a probabilistic polynomial-time test $\mathcal{T}$ and $1 \leq j \leq m$ such that for some fixed a and infinitely many n

$$\Pr[\mathcal{T}(\langle w, w', j-1 \rangle) = 1] - \Pr[\mathcal{T}(\langle w, w', j \rangle) = 1] \geq \frac{1}{n^a}.$$

The probability space is over the choices made by the prover and the randomness of $\mathcal{T}$. We will show that this contradicts the witness-indistinguishability of the underlying NIZK.

Let $(\widehat{P}, \widehat{V})$ be the underlying NIZK protocol (running in the shared random string model). Let τ be a truly random string of ℓ bits. Choose $u \in_R \{w, w'\}$ and give u to $\widehat{P}$. Let $\widehat{P}$ generate a proof $\pi \in_R PN(x, u, \tau)$. By the witness-indistinguishability of the NIZK, with overwhelming probability over choice of τ, no nonuniform probabilistic polynomial-time machine, even given w and w', has nonnegligible advantage of guessing the value of u from π. We will show how to use T to violate this indistinguishability.

Using w and w', construct a simulated transcript of Protocol Z as follows. Break $\rho = b_1, \ldots, b_k$ into $m = k/\ell$ blocks $B_1, \ldots, B_m$. Set $C = \tau \oplus B_j$, so that $\sigma_j = B_j \oplus C = \tau$, which is truly random by assumption. For all $i < j$, construct $\pi_i \in_R PN(x, w, \sigma_i)$. For all $i > j$, construct $\pi_i \in_R PN(x, w', \sigma_i)$. Set $\pi_j = \pi$, which, by assumption, is a uniformly chosen element of $PN(x, u, \tau)$. Let Π denote the resulting transcript.

Run $\mathcal{T}$ on Π. Since τ is truly random and uniformly distributed, C is uniformly distributed as well, so the resulting transcript of m NIZKs is a uniformly chosen element of either $\langle w, w', j-1 \rangle$ (if $u = w'$) or $\langle w, w', j \rangle$ (if $u = w$). We can therefore use $\mathcal{T}$'s assumed ability to distinguish these two cases to obtain a nonnegligible advantage in guessing whether $u = w$ or $u = w'$. □

THEOREM 4.3. *Protocol Z is a zap.*

Proof. Soundness and witness-indistinguishability have been argued. If the underlying NIZK has perfect completeness, then the resulting zap inherits this property. Otherwise, if the underlying NIZK has negligible chance of failure, then completeness for Protocol Z follows from the fact that for any $\hat{\rho}$, since C is random, the probability

that there is some block $\hat{B}_i$ such that $\pi \in_R PN(x, w, \hat{B}_i \oplus C)$ but $V^*(\hat{B}_i \oplus C, x, \pi)$ does not accept is negligible. (Here, as earlier, $\hat{B}_i$ is the ith consecutive block of ℓ bits in $\hat{\rho}$.) In fact, by resampling C, the prover can actually achieve perfect completeness even if the underlying NIZK has negligible chance of failure. □

Our main conclusion is therefore the following corollary.

COROLLARY 4.4. *Let $L \in NP$ be arbitrary.*

1. *If there exists a NIZK for L in the common guaranteed-random string model, then there exists a zap for L in the standard model.*
2. *If there exist zaps in the standard model for* every *language in NP, and if there exist nonuniform one-way functions, then there is a NIZK for L in the common guaranteed-random string model. Furthermore, this NIZK remains zero-knowledge under an adaptive selection of x, that is, when x may depend on σ.*

Proof. The first claim is immediate from the construction and correctness of Protocol Z.

For the second claim we directly apply the idea of Feige, Lapidot, and Shamir [27] of transforming the proof of the statement $x \in L$ into a witness-indistinguishable proof for the statement, "The common shared random string σ is pseudorandom OR $x \in L$." As we will explain, to carry out this approach it is sufficient to have

- a pseudorandom generator G that, say, doubles the length of the seed (in this case a random string is unlikely to be the output of the generator for any seed), and
- a zap for the language $L' = \{(x, \sigma) | x \in L \text{ or } \exists s\ \sigma = G(s)\}$.

The desired pseudorandom generators exist if and only if nonuniform one-way functions exist [37]; moreover, since L' is clearly in NP, it has a zap by the hypothesis. We assume for simplicity (and without loss of generality) that the verifier's message in the zap is chosen uniformly at random.

Recall that we are trying to show that if one-way functions and zaps exist, then there exists a NIZK in the shared random string model. Given a shared random string, treat it as (σ, ρ), where ρ is the verifier's first-round message in the zap for the language L'. The prover simply transforms its witness for $x \in L$ to a witness for $(x, \sigma) \in L'$. Soundness follows from the fact that most σ are not equal to $G(s)$ for any s (this holds because G is length-doubling and σ is truly random).

The system is zero-knowledge since, critically, the simulator for a NIZK is permitted to choose the common string and may in particular choose it to be $G(s)$ for some random s. Then for a random ρ it uses s as the witness for $(x, G(s)) \in L'$. The nonuniform probabilistic polynomial-time indistinguishability of outputs of G from truly random strings, and the witness-indistinguishability of the zaps for L', imply that the output of the simulator is indistinguishable from a real transcript.

Note that since a zap maintains its witness-indistinguishability even when x is chosen after the first-round message is known, we get that the zero-knowledge is maintained even if x is selected in an adaptive manner. □

5. Zaps and verifiable pseudorandom bit generators. In this section we characterize zaps in terms of a new cryptographic primitive: the VPRG, which is inspired by the definition of VPRF [45] (but note the differences). A VPRG is a pseudorandom generator where the holder of the seed can generate proofs of consistency for some parts of the sequence without hurting the unpredictability of the remaining bits. In the standard model we will exhibit a construction of zaps from VPRGs (Protocol VZ below). As we will see, the construction works even if the

VPRG is *approximate*, in that the proofs of the bit values are occasionally incorrectly accepted, so it is possible to cheat a little (this "little" need not be polynomial). We will also show that if zaps exist, then so do approximate VPRGs. Very roughly, *approximate* VPRGs can be designed to have multiple witnesses, so zaps, with their witness-indistinguishability, are sufficiently strong to yield the necessary proofs π of consistency with some member of the set of vectors related to the public verification string (denoted $S(\mathrm{VK})$). In contrast, we do not know how to design *strict* VPRGs to have multiple witnesses.

The following summarizes the relationships between zaps, VPRGs, and NIZKs, both in the standard model and in the common guaranteed-random string model.

SUMMARY 5.1.

1. *NIZKs exist in the common guaranteed-random string model if and only if VPRGs exist in the common guaranteed-random string model (Theorem* 5.10*).*
2. *NIZKs exist in the common guaranteed-random string model if and only if zaps exist in the standard model (Theorem* 4.3 *and Corollary* 4.4*).*
3. *Zaps exists in the standard model if and only if approximate VPRGs (with certain parameters) exist in the standard model (Corollary* 5.7 *and Theorem* 5.8*).*

DEFINITION 5.2. *An (s,k) VPRG is a pseudorandom sequence generator which, for security parameter 1^n, maps a random seed v of length $s(n)$ to an output sequence $a_1,\ldots,a_k$ of length $k=k(n)$ and a verification key* VK *where $s(n)$ and $k(n)$ and the length of* VK *are fixed polynomials. The mapping should satisfy the following requirements:*

- *Verifiability: For any subset $I\subseteq\{1,\ldots,k\}$ of indices, given the seed $v\in\{0,1\}^{s(n)}$ it is possible to construct a proof π of the consistency, with* VK*, of the values of $\{a_i\}_{i\in I}$. We call this a* proof for $\{a_i\}_{i\in I}$*. The construction takes polynomial time and the proof is of polynomial length. Given* VK*, the verifier can check the proof π in polynomial time. The generation of π may be randomized.*
- *Binding: The public verification key* VK *binds the sequence. That is, for each* VK *there is* at most one *associated sequence $a_1,a_2,\ldots,a_k$:*
  1. *This sequence is in the range of the generator on input a seed of length $s(n)$.*
  2. *For all subsets $I\subseteq\{1,\ldots,k\}$, if the verifier accepts a proof π of values $\{b_i\}_{i\in I}$, then there exists a sequence $a_1,\ldots,a_k$ associated with* VK *and $b_i=a_i$ for all $i\in I$. (There can be two different seeds v and v' that yield that same* VK*; in this case they will yield the same $a_1,a_2,\ldots,a_k$.)*
- *Passing the ith bit test: For all $1\le i\le k$ and nonuniform polynomial-time adversaries $\mathcal{T}$ the following holds. Suppose that $\mathcal{T}$ receives for a random $v\in\{0,1\}^{s(n)}$ the verification key* VK *and*

$$a_1,a_2,\ldots,a_{i-1},a_{i+1},\ldots,a_k.$$

The adversary $\mathcal{T}$ selects $I\subset\{1,\ldots,k\}$ such that $i\notin I$ and receives a randomly generated proof π for $\{a_j\}_{j\in I}$. It then attempts to guess a_i. The probability, over the choice of the seed, the random choices in the construction of the proof π, and the random choices by $\mathcal{T}$, that $\mathcal{T}$ guesses a_i correctly is negligibly in n close to $1/2$.

Remark 5.3. Consider a *subset* test, i.e., instead of a single $1\le i\le k$ there is a missing subset of indices and the distinguisher gets the values of $a_{i'}$ at all other locations plus candidate values for the missing locations. It can then ask to see a

proof of consistency for any subset I not including any of the missing indices and then has to guess if the candidate values are correct or just random. This test is equivalent to the ith bit test, just as the distinguishing test and the next bit test are equivalent for regular pseudorandom generators. Note that in the case of VPRF such an equivalence is not clear. The relation between VPRGs and VPRFs is further discussed in sections 5.2 and 9.

We also use a relaxation of VPRGs, which we call $d(n)$-*approximate* VPRGs. The differences are as follows:

- *Relaxed binding:* Intuitively, for any VK, there are at most $d(n)$ values for the revealed string that are accepted as consistent with VK. Rigorously, for each seed v (of length $s(n)$) there are at most $d(n)$ associated sequences of length k, $\mathcal{S}_v = \{\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_{d(n)}\}$ such that for all subsets $I \subseteq \{1, \ldots, k\}$, if the verifier accepts a proof π of values $\{b_i\}_{i \in I}$, then there exists a $1 \leq j \leq d(n)$ such that $\{b_i\}_{i \in I}$ is consistent with $\vec{a}_j$ (same j for all the $i \in I$). In addition, for each "claimed" VK (including those for which there is no corresponding seed) there exists at most one consistent $\mathcal{S}$, and this $\mathcal{S}$ is in fact $\mathcal{S}_v$ for some $v \in \{0,1\}^{s(n)}$.
- *Two-round communication:* The proof of consistency may be "zap-like." On a first round the verifier sends a public-coins message ρ and only then VK and the subset to be proven are chosen. The binding and verifiability conditions hold with high probability over the choice of message of the first round.

Finally, for completeness, we also consider VPRGs in the shared random string model. The binding and verifiability conditions hold with high probability over the choice of the shared string.

5.1. Zaps based on VPRG.

Proofs based on hidden random strings. We find the following "physical" intuition helpful for describing certain types of proofs of membership. The prover is dealt a sequence of ℓ binary cards, where each card has value 1 with probability 1/2. The prover knows the values of the cards and can choose any subset to reveal to the verifier. The verifier learns absolutely nothing about the values of cards that are not explicitly revealed. The prover has no control over the values of the cards. The sequence of cards is a *hidden random string* (HRS).

To prove that $x \in L$, the prover, holding witness $w \in w(x)$, can choose any subset of the hidden bits to reveal to the verifier (cards to turn over). Let α be the locations and values of the revealed bits in the HRS. In addition to α, the prover may send extra information, β, to the verifier. The verifier decides whether to accept or reject x as a function of α, β, and x.

The *soundness* requirement is that for some $q < 1$ such that $1 - q$ is nonnegligible (that is, q is nonnegligibly far from 1), the probability (over the values of the hidden random bits) that the prover can cause the verifier to accept an $x \notin L$ is at most q, even if the prover is arbitrarily powerful. That is, with nonnegligible probability $1 - q$ there is no triple (x, α, β) such that $x \notin L$ and the verifier accepts (x, α, β).

The *witness protection* requirement is that there exist a simulator that on input $x \in L$ (but without a witnesses to $x \in L$),

1. can create (α, β) identically distributed to the (α, β) pairs created in real executions of the proof;
2. given α, β, and *any* witness w^* to $x \in L$, can generate an assignment to the remaining cards so that the distribution on *extended transcripts*, that is, the hidden cards, the revealed cards α, and β, is *identical* to the distribution on

extended transcripts in real executions by a prover holding witness w^*. We call this "completing the simulation with w^*," or "forming a completion with w^*."

Again, α and β are chosen without access to a witness; then, given any witness $w^* \in w(x)$, the simulator can create a completion with w^*, that is, an assignment to all the cards, hidden and exposed, so that the distribution on triples containing α, β, and the values for all the cards is exactly the distribution on these values in real executions with witness w^*.

The concept of an HRS-based proof is exemplified by the noninteractive zero-knowledge proof systems of Feige, Lapidot, and Shamir [27] and of Kilian and Petrank [40]. The idea is to implement the HRS using the output of the VPRG and the opening using the proof capabilities of the VPRG (in contrast to the reliance on the trapdoor properties in [27, 40]). We do not provide new HRS-based proofs in this paper. Our results work with any (existing or future) HRS-based scheme.

Note that although an implementation of an HRS-based proof may be cryptographic, an HRS-based proof is itself a combinatorial, and hence unconditional, object.

Protocol VZ: A VPRG-based zap. The choice of parameters for VPRG-based zaps differs slightly from the choice in the case of NIZK-based zaps. This is because in the case of the VPRG we have less freedom: $k = k(n)$ (the length of ρ) is tied to the parameters of the VPRG.

Choice of parameters. Assume we have an HRS-based proof that for string x and security parameter n polynomially related to $|x|$ uses $\ell(n)$ cards, and on any input x errs with probability at most q. Let $s = s(n)$ be the length of a seed permitting the VPRG to output $k = k(n, |x|)$ bits. To achieve soundness guarantee δ (that is, a cheating prover should succeed with probability at most δ), we require that $k(n, |x|)$ will sufficiently expand the input: it should satisfy

$$q^{k(n,|x|)/\ell(n)} < \frac{\delta}{2^{n+s(n)+\ell(n)}}.$$

The protocol. Let $m = k/\ell$. The HRS proof will be repeated m times. The verifier sends to the prover random bits $\rho = b_1, \ldots, b_k$.

The prover chooses ℓ random bits $C = c_1, \ldots, c_\ell$ and a random seed $v \in \{0,1\}^s$ for the VPRG. Let VK and $a_1, a_2, \ldots, a_k$ be the output of the VPRG on v. The ith bit of the HRS is defined to be

$$a_i \oplus b_i \oplus c_{(i-1 \bmod \ell)+1}.$$

The prover sends to the verifier: VK, $c_1, \ldots, c_\ell$, and m HRS-based proofs that $x \in L$, where the jth proof uses the jth block of ℓ bits of the HRS. For all revealed cards $1 \le i \le k$ the prover provides a_i and a proof π for the consistency of the revealed values.

Let (α_j, β_j) be the values of the revealed cards and additional information in the jth copy of the HRS-based proof, for $j = 1, \ldots, m$. For the revealed cards the verifier, using VK, checks that the value revealed is the correct one. If not, the verifier rejects; otherwise the verifier accepts if and only if, for all m instances of the HRS-based proof, the HRS-based verifier accepts.

LEMMA 5.4. *Protocol VZ is witness-indistinguishable.*

Proof. The proof involves a pair of nested hybrid arguments. The outer hybrid moves from a case in which all copies of the NIZK use one witness (w) to a case in which

all copies use the other witness (w'). Once a distinguishing gap has been identified, the inner hybrid is over proof strings: one extreme has completion γ consistent with w, and the other has completion γ' consistent with w'.

Let w and w' be two witnesses that $x \in L$. Assume there exists a sequence $b_1^*, \ldots, b_k^*$ and a distinguisher $\mathcal{T}$ that, given (x, w, w') and a transcript consisting of $b_1^*, \ldots, b_k^*$ followed by the responses of the m HRS-based proofs of $x \in L$, succeeds with nonnegligible advantage ε to guess which witness, w or w', was used by the prover in generating the response.

By the pigeonhole argument used in hybrid argument, for some $1 \leq j \leq m$ there exists a distinguisher for the following two types of transcripts that distinguishes between them with advantage at least ε/m:

1. The prover uses witness w for the first $j-1$ copies of the HRS-based proof and w' for copies $j \ldots m$.
2. The prover uses witness w for the first j copies of the HRS-based proof and w' for copies $j+1 \ldots m$.

Let us fix such a j for the remainder of the proof.

We first use the simulator, whose existence is guaranteed by the definition of an HRS-based proof, to choose α and β for the jth copy of the HRS-based proof. For a given seed v to the VPRG, for the positions $1 \leq i \leq \ell$ indicated by α, we choose c_i so that the value $a_{(j-1)\ell+i} \oplus b_{(j-1)\ell+i} \oplus c_i$ opened is the value indicated by α.

By the definition of witness-indistinguishability for an HRS-based proof, the simulator, now given w and w', can efficiently find a completion (choices for the unopened values) γ corresponding to the case in which the witness used is w, as well as a completion γ', corresponding to the case in which the witness used is w'. Together with the seed v, the completions γ and γ' determine, respectively, the values of the bits c_i for each position i that is *not* indicated by α. (The values for the positions indicated by α were fixed above and will remain unchanged throughout the rest of the proof.) Let $r = |\gamma| = |\gamma'|$. For $0 \leq d \leq r$, we denote by $c(\gamma, \gamma')_{[d]}$ values for the c not indicated by α that agree with γ in positions $1, \ldots, d$ and agree with γ' in positions $d+1, \ldots, r$. Thus, when $d = 0$ the values all agree with γ', while when $d = r$ the values all agree with γ.

We will now form a hybrid chain on proof strings. In every element in the chain, the seed v remains unchanged, as do the b^* and the values for the c in the positions indicated by α. Only the c not indicated by α will change as we move from one element in the chain to the next. The first element in the chain has values $c(\gamma, \gamma')_{[r]}$ for the c not indicated by α. Thus, these values all agree with γ, where the witness is w. Having fixed all the c for this element of the chain, we can complete the description of the first element of the chain. The first $j-1$ blocks are HRS-based proofs constructed with witness w, and blocks $j+1$ through m are constructed with witness w'. Moreover, by choice of the c, the jth block has been completed with w.

The next element in the chain has values $c(\gamma, \gamma')_{[r-1]}$ for the c not indicated by α. Everything else remains the same: the values for the remaining c that were fixed in the description of the first element in the chain are again used here. Then, having again fixed all the c, the first $j-1$ blocks are HRS-based proofs constructed with witness w, and blocks $j+1$ through m are constructed with witness w'. Note that the jth block might not really be something that could have been generated by the prover, since it is not completely consistent with a proof constructed using either w or w'.

In general, for $0 \leq d \leq r$, the $d+1$th element in the chain has values $c(\gamma, \gamma')_{[r-d]}$ for the c not indicated by α, for $0 \leq d \leq r$. The last element in the chain has values $c(\gamma, \gamma')_{[0]}$, that is, it agrees completely with γ'.

We note that the chain is nonempty, since otherwise the behavior of the prover on witnesses w and w' is identical and therefore yields no possibility of distinguishing between the two witnesses. Thus, the number of steps in the $\gamma \to \gamma'$ hybrid chain is $1 \le r \le \ell$ (including the endpoints, the chain has $r+1$ elements). We assumed an ε/m advantage in distinguishing the two endpoints of the chain, hence there is an $i \le \ell + 1$ where the adversary has advantage at least $\varepsilon/(m\ell)$ to distinguish between the $i-1$ and ith elements in the chain. The pseudorandomness of the VPRG can be broken at this location. The subset I is the one determined by α and the HRS proofs used in the other $m-1$ blocks. □

LEMMA 5.5. *Protocol VZ is sound; moreover, the first round can be fixed nonuniformly.*

Proof. Let $x \notin L$, $c_1, \ldots, c_\ell$, and let the VPRG verification key VK be fixed. We will show that with overwhelming probability, over the choice of $b_1, \ldots, b_k$, the prover will fail to convince the verifier to accept x. The key point is that once everything but the b has been fixed, the hidden random string is truly random—because $b_1, \ldots, b_k$ have not been chosen yet and are to be chosen at random. Therefore each copy of the HRS-based proof has probability at most q of failing to cause rejection. Since each proof is independent (because the b_i used in each copy of the HRS-based proof are independent), the overall probability that all $m = k/\ell$ copies fail is at most q^m.

The number of possible assignments to the c, VK,[4] and $x \notin L$ is at most $2^{\ell+s+|x|}$. Hence, as long as

$$2^{\ell+s+|x|} q^m \le \delta$$

for a *random* $b_1, \ldots, b_k$, the probability that there even exists a "bad" choice of $c_1, \ldots, c_\ell$, VK, and x that erroneously causes the verifier to accept is at most δ. Therefore, not only is the protocol sound, but also the first message (the b) can be fixed nonuniformly. □

THEOREM 5.6. *Given an HRS proof system for a language L using ℓ cards and with probability of error at most q and given a VPRG mapping a seed s to k bits, if*

$$q^{k/\ell} < \frac{\delta}{2^{|x|+s+\ell}},$$

then protocol VZ is a zap for L.

Note that if instead of a VPRG we use a $d(n)$-approximate VPRG, then we can obtain a similar result by adjusting the counting argument to accommodate the $d(n)$ possible openings consistent with VK; see the next corollary.

COROLLARY 5.7. *Given an HRS proof system for L using ℓ cards and with probability of error at most q and given a $d(n)$-approximate VPRG mapping a seed s to k bits, if*

$$q^{k/\ell} < \frac{1}{d(n)} \frac{\delta}{2^{|x|+s+\ell}},$$

then protocol VZ is a zap for L.

As we show next, the converse holds as well and we can use zaps to obtain *approximate* VPRGs.

[4]Note that we should count only the number of seeds $v \in \{0,1\}^s$ and not the various possible public commitment strings, since what matters is the value $a_1, a_2, \ldots, a_k$ of the sequence associated with VK, and this sequence, by Definition 5.2, must correspond to one in the range of the generator on input (seed) of length $s(n)$.

THEOREM 5.8. *Let $\ell(n)$ be any polynomial. Fix $m \geq 2$. Let G be any pseudorandom generator taking a seed of length $s(n)$ and producing an output of length $\ell(n)$. Then, assuming every language $L \in NP$ has a zap, one can construct a $d(n)$-approximate VPRG expanding a seed of length $m \cdot s(n)$ to a string of length $k(n) = m \cdot \ell(n)$, where $d(n) = m2^{\ell(n)}$.*

Note that the expansion is arbitrary, since $\ell(n)$ is an arbitrary polynomial and pseudorandom generators exist for any polynomial expansion, based on any one-way function.

Proof. We use the commitment scheme of [46]. (In this scheme, the receiver sends an initial message, which can be fixed nonuniformly.) The prover commits to m seeds of length $s(n)$; VK is the concatenation of the m commitments. Using the pseudorandom generator, each seed yields a block of length $\ell(n)$, for a total output length of $m \cdot \ell(n)$. For any set I of indices, the prover can reveal the values of the pseudorandom bits $\{a_i\}_{i \in I}$ and can prove using a zap that the revealed bits in at least $m - 1$ of the blocks are consistent with VK. (This is certainly in NP, so it has a zap by assumption.)

Verifiability is immediate from the zap. Relaxed binding is also simple, since given VK, the number of possible strings the prover can convince the verifier to accept is $m2^{\ell(n)} = d(n)$. (The prover has freedom to choose one of m blocks on which he can cheat and which of $2^{\ell(n)}$ values to plug in there.)

It remains to show passing of the ith bit test. Suppose the construction fails this test with some bias δ. We will use the block B containing i to distinguish pairs of the form $(C(v), \tau)$ from $(C(v), G(v))$, where $C(v)$ is a commitment to a seed v of length $s(n)$ and τ is random of length $k(n)$. Given a pair $(C(v), \mu)$, construct a key VK as follows. Choose $m-1$ seeds $v_1, \ldots, v_{m-1}$, and arrange commitments to these seeds and the commitment $C(v)$ so that $C(v)$ is the commitment to the supposed seed for block B. Open the values for all positions other than i, and provide a zap of approximate consistency with VK, using the chosen seeds $v_1, \ldots, v_{m-1}$ as the witnesses to the fact that the revealed bits in at least $m - 1$ of the blocks are consistent with VK.

If μ is pseudorandom with seed v, then by the witness-indistinguishability of the zap, the advantage in guessing the ith bit is close to δ. (The witness-indistinguishability may introduce a negligible error, so we don't get exact advantage δ.) On the other hand, if μ is truly random, then there can be no bias. Therefore we have a distinguisher for $(C(v), \tau)$ from $(C(v), G(v))$. ∎

Remark 5.9. In the case of ordinary pseudorandom generators, it is known that the ability to expand by even one bit can be used to obtain arbitrary expansion. Is the same true of (approximate) verifiable pseudorandom generators? From Corollary 4.4, Theorem 5.10, and Corollary 5.7 we have only a higher threshold: if any *polynomial* expansion is possible (from n to $n^{1+\varepsilon}$ for fixed ε), then we can build zaps and hence arbitrary expansion. See more open problems in section 9.

5.2. Construction of VPRGs. A nontrivial VPRG, with a given desired (polynomial) expansion from seed to output, can be constructed from any VPRF. The idea is simply that a VPRG is a VPRF with a small domain. This is almost true, except for some technical issues, which we describe next.

There is a difference in the binding requirement for a VPRF, according to the definition in [45], and the binding requirement for a VPRG (Definition 5.2): a VPRF allows the total number of "legitimate" functions (accepted by the verifier) to be proportional to the number of public keys, whereas a VPRG counts them according to the seeds. We can take the domain of the VPRG to be any polynomial that we

wish. We therefore take it to be larger than the length of the public key of the VPRF plus the security parameter n. We must ensure that for any public key of the VPRF accepted by the verifier (of both the VPRF and the VPRG), there is a corresponding seed mapping to this public key. We therefore modify the mapping of VPRG seeds to VPRG public keys to allow any public key of the VPRF. Specifically, the new seed is of length n + the maximum of the seed length of the VPRF and the length of the public key of the VPRF. The VPRG is now as follows: If the first n bits are all 0, then map the suffix (or as many bits as the public key needs) to the public key; otherwise operate as in our original VPRG construction. Note that the event that the leading bits are all 0 occurs with negligible probability, so the security of the VPRF is preserved.

However, such a construction is overkill; moreover, the only known constructions we have of VPRFs require specific assumptions such as the strong-RSA assumption [45] or various Diffie–Hellman assumptions for groups with bilinear mappings [44, 16, 17].[5]

The goal of this section is to provide an alternate construction of VPRGs, based on general trapdoor permutations. We do not require the "enhanced" property, as defined in [30]. The construction follows along the lines of the trapdoor-based synthesizer construction of Naor and Reingold [48]. To obtain (nonapproximate) VPRGs we require that the trapdoor permutation be certified (see [6]).

We assume the existence of a family $\mathcal{F}_n$ of certified trapdoor permutations with common domain $\mathcal{D}_n$, together with a hard-core predicate (n is a security parameter). The VPRG output is given as a binary matrix (say, in row-major order). The matrix has r rows and c columns, where $rc = k$. Choose r functions $f_1, \ldots, f_r$ from $\mathcal{F}$ (one for each row) and c random y (one for each column) in the common range of all the trapdoor permutations, $\mathcal{D}_n$. The (i, j) entry of the matrix will be the hard-core predicate of $f_i^{-1}(y_j)$.

Let VK $= f_1, \ldots, f_r, y_1, \ldots, y_c$. To prove the value of the (i, j) entry, reveal $f_i^{-1}(y_j)$. Verification is immediate using VK and the fact that each f_i is a permutation that is easy to compute in the forward direction.

The length of the seed s is $r \log |\mathcal{F}_n| + c \log |\mathcal{D}_n|$. As n is fixed and k grows, the expansion is roughly quadratic. This completes the description of our VPRG construction. The proof that it satisfies the ith bit test closely follows the proof in [48].

An alternative to trapdoor permutations is to use Diffie–Hellman in groups with efficient bilinear mappings where the computational Diffie–Hellman is assumed to be hard [10, 39]. The easiness of the decisional Diffie–Hellman problem in these groups yields a simple method for verification. (These are less stringent requirements than in the existing constructions of VPRFs in [44, 16, 17].)

The standard example of a certifiable trapdoor function is RSA with a *prime* public exponent e satisfying $e > N$. This assures that e and $\phi(N)$ are relatively prime. If we relax the *perfect* binding requirement and instead aim for an approximate VPRG, then we can use certain uncertified trapdoor permutations, as in the next example, which is inspired by Shamir's pseudorandom generator [52].

Consider RSA with small exponent. Choose a random RSA modulus N and $y_1, \ldots, y_c \in Z_{N^*}$. These form the verification key. Associate with the ith row the ith smallest prime. The (i, j)th output bit is the hard-core bit of $y_j^{1/p_i} \bmod N$. The

[5] In light of Corollary 4.4 and Theorem 5.6, we therefore get zaps and NIZKs based on the same assumptions.

possible problem is that p_i may divide $\phi(N)$. In this case y_j may have multiple p_ith roots, possibly with different hard-core bits, and the owner of the generator can cheat. However, even if the key is incorrectly chosen, so that N is *not* a product of two primes, there can be at most $\log N/\log\log N$ such primes, and hence we get *relaxed* binding. (Note that if N is not a product of two primes, then presumably the output sequence is not even close to that of a legal (two prime modulus) output; but this can be resolved by allowing a small probability of any N being chosen, which does not affect the pseudorandomness property.) We can take this into account in setting the parameters.

5.3. Shared string VPRGs and NIZKs.

THEOREM 5.10. *VPRGs in the* shared random string *model exist if and only if NIZKs exist in the shared random string model. Moreover, in the shared random string model NIZKs imply VPRGs of arbitrary expansion.*

Proof sketch. To construct VPRGs from NIZKs in the shared random string model, commit (say, using the protocol of [46], taking the first several bits of the common random string to be the first-round message of the receiver) to the seed of a pseudorandom sequence and use a NIZK to prove that the revealed value is the correct one. For the converse, given a VPRG in the common random string model, construct essentially the NIZK of Feige, Lapidot, and Shamir [27], in which the bits of the hidden random string (see more about them above) are the bits of the VPRG. □

6. Oblivious transfer in the standard model. Although there are many protocols under various assumptions for oblivious transfer, to date no 3-round protocol has been shown secure, without resorting to a random oracle model. We provide a protocol for 1-out-of-2 OT for which we are able to prove that the chooser's privacy is protected by the quadratic residuousity assumption (QRA) [35], and the sender's privacy is protected statistically (that is, with overwhelming probability over choices made by the sender, at most one value is transmitted to the chooser).[6] The protocol is not known to ensure correctness, that is, the sender may choose what to send as a function of the chooser's message.

For simplicity, we describe the protocol for the case in which the sender's two inputs are bits b_0, b_1. The first round of the protocol, described next, can be eliminated if the sender has a public key. In this case, the public key is chosen to be a random first-round message ρ for zaps.

1. If the sender has no public key, then it chooses a first-round message ρ for a zap and sends it to the chooser. (If the sender instead has a public key, then this round is not needed.)
2. Let $i \in \{0,1\}$ be the chooser's input. The chooser chooses a random 2-prime modulus N and two random strings y_0, y_1 in $\mathbb{Z}_N^*$ such that y_{1-i} is a quadratic residue modulo N and y_i is a nonresidue with Jacobi symbol 1. Using ρ, the chooser gives a zap π of the statement: "y_0 is a QR mod N OR y_1 is a QR mod N."
3. The sender verifies the zap (ρ, π) and, if verification fails, the sender aborts. If verification succeeds, the sender chooses $x_0, x_1 \in_R \mathbb{Z}_N^*$ and sends the following two values to the chooser in any order: $\{y_{b_0}^{b_0} x_0^2 \bmod N, y_{b_1}^{b_1} x_1^2 \bmod N\}$.

We now give a proof sketch of correctness of the protocol. Assume first that both parties are following the protocol correctly. Let y_i be the unique quadratic nonresidue modulo N among y_0, y_1. Then $y_i^{b_i} x_i^2$ is a quadratic residue modulo N if and only if

[6] Previous applications of QRA to OT appear, for example, in [12, 41].

$b_i = 0$. On the other hand, since y_{1-i} is a quadratic residue modulo N, so is $y_{1-i}^{b_{1-i}} x_{1-i}^2$, independent of the value of b_{1-i}. Thus, the ability of the chooser to compute quadratic residuousity yields only and exactly the value of b_i.

Now assume the sender follows the protocol correctly but the chooser does not. The soundness of the zap ensures that at least one of y_0, y_1 is a quadratic residue modulo N. Assume then that y_j is a quadratic residue modulo N. Then $y_j^{b_j} x_j^2 \bmod N$ is always a quadratic residue, independent of b_j, and independent of how N is chosen. Thus, the chooser can learn at most one of b_0, b_1. Finally, by the QRA and the way in which a good chooser constructs N, y_0, y_1, the sender cannot distinguish which of y_0, y_1 is the quadratic residue. In particular, the (polynomial-time bounded) sender cannot distinguish among the following four distributions $(N, y_0, y_1, (\rho, \pi))$, where ρ is fixed in step 1, N is chosen according to the protocol, and the other elements are chosen as follows:

1. y_0 is a random quadratic residue modulo N, y_1 is a random nonresidue with Jacobi symbol 1, and y_0 is the witness used in constructing π;
2. y_0 and y_1 are both quadratic residues modulo N and y_0 is the witness used in constructing π;
3. y_0 and y_1 are both quadratic residues modulo N and y_1 is the witness used in constructing π; and
4. y_1 is a random quadratic residue modulo N, y_0 is a random nonresidue with Jacobi symbol 1, and y_1 is the witness used in constructing π.

Distributions 2 and 3 are indistinguishable by the witness-indistinguishability of the zap. Distributions 1 and 2 (and, similarly, distributions 3 and 4) are indistinguishable by the QRA. Thus, distributions 1 and 4 are computationally indistinguishable, so the sender does not learn which of b_0, b_1 has been transferred to the chooser.

Remark 6.1. Naor and Pinkas [47] were able to modify this approach to produce a different protocol with similar security properties; their protocol is based on decisional Diffie–Hellman and does not explicitly use zaps.

7. Timing-based applications. In this section we describe two delay-free timing-based (see section 2.5) applications for zaps:

- 3-round concurrent zero-knowledge proofs of knowledge for any language $L \in$ NP, and
- 2-round deniable authentication.

7.1. 3-round concurrent zero-knowledge proofs of knowledge. At a high level, the protocol consists of two steps. Let $x \in L$ be the statement to be proved. First, the verifier chooses a statement S and proves, using a zap, that S is true. Second, the prover gives a proof of knowledge of a witness to the statement "$x \in L \vee S$." Intuitively, soundness comes from the fact that the verifier's proof does not reveal a witness to S. This is achieved by constructing S to be the logical-OR of two independent statements—in such a case witness-indistinguishability is known to imply witness-hiding [28]. A single preprocessing step is needed for both the proof of knowledge and to provide the first-round ρ for the verifier's zap of S.

In a little more detail, the statement S is a claim that of two given timed commitments to two random strings, at least one is *valid*—forced recovery of the committed value is possible (see the discussion in section 2.5). Verifiable recovery implies the existence of a knowledge extractor. The extractor is used in constructing the simulator for proving zero-knowledge.

Let f be a one-way function. Let $f^{(k)}(s)$ be the kth iterate of f applied to s.

Associated with any randomly chosen s, there is a k-bit pseudorandom string B consisting of the hard-core bits of

$$s, f(s), f^{(2)}(s), \ldots, f^{(k-1)}(s),$$

respectively (this is the Blum–Micali [9] generator). The basic technique for proving knowledge of a witness $w \in w(x)$ is to commit to B^0 and B^1 by giving a pair $f^{(k)}(s^0), f^{(k)}(s^1)$. The verifier then chooses one of the two blocks, say, B^i, to be revealed. The prover releases s^i and gives $w \oplus B^{1-i}$, together with a proof of consistency with the initial commitment $f^{(k)}(s^{1-i})$. Because this gives only a probability $1/2$ of detecting cheating, the process is repeated p many times in parallel. (Choose p, the number of parallel repetitions, according to the required probability of soundness error.) The preprocessing step (step 1 in the protocol) is just the transmission of sufficiently many pairs of the form $f^{(k)}(s^0), f^{(k)}(s^1)$, together with a ρ for the verifier's zap in step 2.

3-*round timed concurrent ZK POK for* $L \in$ NP. Common input $x \in L$, input to prover $w \in w(x)$.

1. (a) Let f be a fixed one-way permutation (f is part of the protocol, known to both parties). The prover sends to the verifier $2p$ pairs $(f^{(k)}(s_1^0), f^{(k)}(s_1^1), \ldots, (f^{(k)}(s_{2p}^0), f^{(k)}(s_{2p}^1)))$ for randomly chosen s_i^j, $i = 1, \ldots, 2p$, and $j = 0, 1$.
   (b) The prover also sends to the verifier ρ, a round-one message for a zap.
2. (a) The verifier selects a random $2p$-bit string $c_1 \ldots c_{2p}$.
   (b) The verifier chooses two random values y_0 and y_1 of length p and constructs from them two commitment strings $\zeta_0 \in_R TC(y_0)$ and $\zeta_1 \in_R TC(y_1)$ using the timed commitment protocol. Using ρ, the verifier sends π, proving that at least one of the ζ_i is valid ((ρ, π) constitutes a zap).
   (c) The verifier sends to the prover a new round-one message ρ'.
3. (a) For each $1 \le i \le p$, the prover sends to the verifier $s_i^{c_i}$. For each such i the prover also computes B_i, the pseudorandom k-bit string consisting of the hard-core bits of
$$s_i^{1-c_i}, f(s_i^{1-c_i}), f^{(2)}(s_i^{1-c_i}), \ldots, f^{(k-1)}(s_i^{1-c_i}).$$
   (b) The prover checks the zap $(\zeta_0, \zeta_1, \rho, \pi)$. If the proof is invalid, the prover terminates the protocol.
   (c) The prover chooses z at random.
   (d) Using $B_1, \ldots, B_p$ the prover commits to z and w. Specifically, it sends $z \oplus B_1, \ldots, z \oplus B_p$; similarly it commits to w, using blocks $B_{p+1} \ldots B_{2p}$. We call the commitments to z the *first group*, and the commitments to w the *second group*. Using ρ', the prover constructs a proof π' that at least one of the following two statements holds: (1) there exists z consistent with all of the commitments in the first group and z is the value committed to in one of the timed commitments ζ_0 or ζ_1; or (2) there exists w consistent with all of the commitments in the second group and $w \in w(x)$. The witness used for constructing the zap is the set of strings $\{s_{p+1}^{1-c_{p+1}}, \ldots, s_{2p}^{1-c_{2p}}\}$.

Timing constraints: V accepts P's round-three message only if it arrives within time α on V's local clock from the time at which V sent its round-two message. α and β (for the timing assumption) should be chosen to satisfy $\alpha \le \beta$ and $2\beta + \gamma < t$, where

the value t is the time below which it is safe to assume that the timed commitment cannot be broken, even by a PRAM, and γ is an upper bound on the time it takes to create a zap by a program that is given a witness. For completeness, α must be sufficiently large to permit the necessary computations by P and the round-trip message delay.

The protocol is concurrent zero-knowledge because it is *straight-line simulatable* via the forced openings: every interaction can be simulated without rewinding the prover [24]. To see this, consider a single interaction. The simulator generates a real round-one message, which is given to the verifier. The verifier constructs its timed commitments and their proof π. The simulator checks π and, if it is correct, continues with the protocol. The clocks are frozen and the simulator computes the forced opening of the timed commitments, obtaining y, the decommitment of one of ζ_0 and ζ_1. The clocks are started again, the simulator sets $z = y$, commits to z and a random string (instead of w), and constructs π' using the commitment to z as the witness. When the adversarial scheduler schedules P's next message, the simulator sends π'.

Now consider four classes of transcripts: they differ according to the value committed to in the first block (random or $z = y$), the value committed to in the second block (w or random), and which witness is used in creating the zap π' (w or z). Only four of the eight possibilities are relevant:

1. First block: random; second block: w; witness is w.
2. First block: $z = y$; second block: random; witness is y.
3. First block: $z = y$; second block: w; witness is w.
4. First block: $z = y$; second block: w; witness is y.

The real transcripts are the first class. The simulator outputs the second class. Classes 1 and 3 are computationally indistinguishable by the one-wayness of f and the properties of hard-core bits. Classes 2 and 4 are indistinguishable for the same reason. Classes 3 and 4 are indistinguishable by the witness-indistinguishability of zaps. Hence, classes 1 and 2 are computationally indistinguishable.

We now argue that the interaction is sound and a proof of knowledge. If the prover completes the proof with probability δ, then standard extraction techniques, i.e., forcing P to explore two computational paths, can be used to obtain a witness (strings $s_i^{1-c_i}$ for the appropriate set of indices i) with probability negligibly close to δ^2.

Suppose $x \notin L$, and that a cheating prover succeeds with nonnegligible probability δ to cause the verifier to accept. Then the timed commitment scheme can be broken with probability negligibly close to $\delta^2/2$, as follows. Consider a (possibly fictitious) nonfaulty process running a perfect clock. By the (α, β) assumption, if V is nonfaulty and measures time at most α on its own clock between the time at which it sent its round-two message and the time at which it received P's round-three reply, at most β real time has elapsed.

Assume we are given a timed commitment $\xi_1 \in_R TC(y)$. Run the cheating prover for one step. Choose $c_1 \dots c_p$ at random. Choose y' and give $\xi_1 \in_R TC(y')$; then, using the witness based on y', act as the verifier and in step 2 give a zap that at least one of ξ_1 and ξ_2 is valid. By definition, such a zap can be constructed within time γ. If the prover responds (which it will do with probability at least δ), repeat steps 2 and 3, using the same timed commitments and zap in step 2, but with a new random string $c'_1, \dots, c'_p$. If the prover responds again, use the revealed $s_{c'_i}$ to obtain at least one of $y, y', w \in w(x)$. Since $x \notin L$, the value obtained is either y or y'. By the witness-indistinguishability of the verifier's zap, the value will be y with probability

1/2. The total time required for extraction is at most $2\beta + \gamma < t$, contradicting the assumption that breaking the timed commitment requires time at least $t < T$. Thus, the system is sound. That the system is a proof of knowledge is immediate from the extraction procedure described above.

THEOREM 7.1. *If TC is a timed commitment protocol satisfying the requirements of section* 2.5*, then the protocol described above is a* 3*-round timed concurrent zero-knowledge proof of knowledge system for any language L in NP.*

Remark 7.2. The straight-line simulability also permits the prover to use differing (α, β) pairs for the different verifiers.

7.2. Timed 2-round deniable authentication. We now describe a 2-round timed concurrent deniable authentication protocol (see section 2.3 for definition and discussion), based on zaps and timed commitments.

The $A\mathcal{P}$ has a public key $\langle E_1, E_2, \rho \rangle$, where E_1 and E_2 are public encryption keys chosen according to a public-key cryptosystem generator that is nonmalleable against chosen-ciphertext attacks in the postprocessing mode, and ρ is a first-round message for a zap.

1. The verifier chooses random strings y_0, y_1, r and sends to the prover $c \in_R E_1(m \circ r)$ and timed commitments $\zeta_0 \in_R TC(y_0)$ and $\zeta_1 \in_R TC(y_1)$. In addition, using ρ, the verifier gives a zap that at least one of the ζ_i is valid. Finally, the verifier also sends to the prover a first-round message ρ' for a zap.
2. The prover checks the zap (ρ, π) and aborts if verification fails. Otherwise, the prover sends to the verifier $\eta \in_R E_1(r)$, $\delta \in_R E_2(s)$ for a randomly chosen s. Using ρ', the prover sends a zap π' that at least one of the following holds: $\eta \in E_1(r)$ or $s \in \{y_0, y_1\}$. (More specifically, π' is a proof that η is an encryption under E_1 of the suffix of the message encrypted by ciphertext c OR δ is an encryption under E_2 of one of the values committed to by ζ_1, ζ_2.) The witness used in creating π' is the set of random bits in creating η or δ. In a regular execution η is used.

V accepts if and only if both (1) the zap (ρ', π') is accepted and (2) P's response is received in a *timely* fashion, as specified in the timing constraints.

Timing constraints. P's round-two message must arrive within time α on V's local clock from the time at which V sent its round-one message. α and β are chosen to satisfy $\alpha \leq \beta$ and $\beta + \gamma < t$, where the value t is the time below which it is safe to assume that the timed commitment cannot be broken, even by a PRAM, and γ is an upper bound on the time it takes to create a zap by a program that is given a witness. For completeness, α must be sufficiently large to permit the necessary computations by P and the round-trip message delay.

This completes the description of the deniable authentication protocol.

THEOREM 7.3. *If TC is a timed commitment protocol satisfying the requirements of section* 2.5*, then the* 2*-round protocol is sound and deniable to a distinguisher that has access to the public key of $A\mathcal{P}$.*

Proof. We first argue unforgeability. Suppose that the adversary is trying to forge message m and is given by the verifier the "challenge" $E(m \circ r)$. Then by the nonmalleability of E_1 it cannot produce $E_1(r)$, even if it has access to a decrypting oracle for E_1 on all messages with prefix different than m.[7] Therefore, given that the

[7] Actually it seems that we do not need E_1 to resist any chosen-ciphertext attacks and it is enough that it is nonmalleable against chosen-plaintext attacks. The reason is that we can give the adversary an encryption of a random string instead of $E_1(r)$ and use the forced opening of the timed commitment to obtain a zap in the second step.

adversary provides a zap at step 2, it must be the case that $s = y_i$ for some $i \in \{0, 1\}$. In this case, the real prover, who knows D_2, and the adversary together can be used to break the timed commitment scheme with probability 1/2: given $TC(y)$, choose y' at random and give $TC(y')$; then, using the witness based on y', give a zap that at least one of $TC(y)$ or $TC(y')$ (in random order) is recoverable. By definition, such a zap can be constructed within time γ. If the forger gives back $s = y$ within time α, then TC has been broken in time at most $\beta + \gamma < T$.

We now argue deniability. The simulator extracts from $TC(y_0)$ and $TC(y_1)$ either y_0 or y_1 (for at least one of them this should be possible). It then creates $\eta = E_1(r')$ for a random r' and creates $\delta = E_2(y_i)$ and uses it as a witness to a zap that $\eta \in E_1(r)$ or $s = y_i$. The proof of indistinguishability of simulated and real transcripts is analogous to the proof of Theorem 7.1 and relies on the indistinguishability of encryptions of E_1 and E_2.

Note that there is no real need to choose E_2 different from E_1. □

The need to add ρ to the public key of the authenticator may increase its size significantly. However, ρ is used only to show the recoverability of TC. If we are equipped with a timed commitment where recoverability is self-evident, then there is no need to have it at all and we can use any public key of a sufficiently strong encryption.

Deniability when the distinguisher has the private keys of $A\mathcal{P}$. We now describe a protocol that is deniable even for a distinguisher who has the private keys of $A\mathcal{P}$, based on the (not deniable) authentication protocol given in [18]. The $A\mathcal{P}$ has a public key $\langle E, \sigma \rangle$, where E is a public encryption key chosen according to a public-key cryptosystem generator that is nonmalleable against chosen-ciphertext attacks in the postprocessing mode, and σ is a random string to be used in a NIZK of a language defined below:

1. The verifier chooses a random string r and sends to the prover $c \in_R E_1(m \circ r)$ and timed commitment $\zeta \in_R TC(r)$. In addition, using σ, the verifier gives a NIZK proof π that ζ is valid and the committed value equals the suffix of the plaintext of c.
2. The prover checks the proof (σ, π) and aborts if verification fails. Otherwise, the prover decrypts c and obtains m and r and sends to the verifier r in the clear (of course only if the decrypted m equals the value it wishes to authenticate).

V accepts if and only if both (1) the received r' equals the value r he selected and (2) P's response is received in a *timely* fashion, as specified in the timing constraints.

Timing constraints. P's round-two message must arrive within time α on V's local clock from the time at which V sent its round-one message. α and β are chosen to satisfy $\alpha \leq \beta$ and $\beta < t$, where the value t is the time below which it is safe to assume that the timed commitment cannot be broken, even by a PRAM. For completeness, α must be sufficiently large to permit the necessary computations by P and the round-trip message delay.

THEOREM 7.4. *If TC is a timed commitment protocol satisfying the requirements of section* 2.5*, then the* 2*-round protocol is sound and deniable to a distinguisher that has access to the public and private keys of $A\mathcal{P}$.*

Proof. Unforgeability follows along the lines of the unforgeability in [18], the zero-knowledge property of (σ, π), and the timing requirements. We now argue deniability. The simulator extracts from $\zeta = TC(r)$ the value r and by the soundness of the NIZK proof system this is the same r as in the ciphertext. It then adds r to the transcript. □

8. Witness protection in the resettable model.

8.1. Resettable witness-indistinguishability. For a formal definition of resettable witness-indistinguishability, see [13]. We will motivate the definition informally by focusing on smart cards. Intuitively, a smart card is loaded with x, $w \in w(x)$, and a seed s for a pseudorandom function, at the time it is created. This seed is the only source of randomness the card has; furthermore, we assume that the card is stateless, i.e., does not change its internal memory between sessions (so it cannot store a counter and use it in conjunction with the seed to define the randomness of the current session). Our interest is in protecting the prover from a verifier V^* that runs the prover many times on the same x, w, s. Let us use the notation $(P(x, w, s), V^*(x, z))$ to denote the transcript of exactly this kind of attack where z is auxiliary information known to V^* (in particular, we may even have $z = w, w'$). Letting $w, w' \in w(x)$, a proof that $x \in L$ is resettable witness-indistinguishable if for all probabilistic polynomial-time T, V^*, and z

$$\left|\Pr_{(V^*,s)}[T(P(x,w,s)V^*(x,z))] - \Pr_{(V^*,s')}[T(P(x,w',s')V^*(x,z))]\right| \leq \nu(n).$$

Every zap for a language $L \in NP$ yields a 2-round resettable witness-indistinguishable proof system for L as follows. On input ρ, the prover computes $R = f_s(x, \rho)$, where f_s is a pseudorandom function with seed s. It then uses the bits R as the random bits in computing the zap response π.

Soundness holds because the round-one message ρ is not needed for *unpredictability*—indeed, soundness holds even if some $\hat{\rho}$ is fixed nonuniformly and before x is chosen. As for witness-indistinguishability, from the WI of the zap it follows that an assumed distinguisher for the resettable system can be used to distinguish the output of the pseudorandom function from truly random, a contradiction.

8.2. Resettable zero-knowledge. We first present our 3-round timing-based rZK protocol for any $L \in NP$ and then compare it to previous results.

Let (E, D) be the encryption and decryption algorithms of a semantically secure against chosen-plaintext attack (CPA) encryption method. The scheme need not be public-key, but there should be a public description pd of the encryption key with the following two properties: (1) it is easy to verify that decryption is unique, that is, given ciphertext c and a public description pd there should be at most one p satisfying $c \in E(p)$; and (2) given pd it is easy to verify that there exists decryption key dk such that given $c \in E(p)$ we have $D_{dk}(c) = p$.

An example of such an encryption scheme can be based on RSA with large public exponent, as in section 5.2. That is, the public key is (e, N), in which the exponent e is prime and sufficiently large (so that e cannot possibly divide $\phi(N)$), $pd = (e, N)$ in this case, and the actual encryption is done using the hard-core predicate of the exponentiation with e function. Alternatively, E could be a pseudorandom permutation cipher, which can be turned into a semantically secure against CPA encryption scheme using random padding, and where pd is a (perfectly binding) commitment to the seed. The fact that E is a permutation assures unique decryption.

For this application, we require that the timed commitment scheme be secure *nonuniformly*, i.e., that there does not exist a PRAM with fixed advice tape that can break the commitment scheme with nonnegligible probability in time less than t. This is one of the cases where security against nonuniform adversaries is used in an essential way.

3-*round timing-based rZK for* $L \in NP$.

1. The prover chooses pd (the public description of the encryption key of E) and a random string ρ and sends both to the verifier.
2. The verifier checks that encryptions under E are uniquely decryptable (as discussed above) and if not, rejects. Assuming E passes the test, the verifier chooses random strings y_0, y_1 and sends to the prover timed commitments $\zeta_0 \in_R TC(y_0), \zeta_1 \in_R TC(y_1)$ and, using ρ, a zap π that at least one of the two timed commitments is valid. The verifier also sends a string ρ' to the prover.
3. The prover checks (π, ρ). If it is accepted, then the prover uses the random bits defined by an application of its pseudorandom function on the message sent by the verifier to generate $a \in_R E(w)$ and $b \in_R E(z)$, where $w \in w(x)$ and z is random. Using ρ' and part of the output of the pseudorandom function the prover also generates a zap π' that $w \in w(x)$ OR $z \in \{y_0, y_1\}$. The witness used consists of the random bits used in generating a. a, b, and π' are sent.

The verifier checks that (ρ', π') is accepted, that b has unique decryption, and that the prover's response was timely, as defined by the timing constraints, accepting if and only if all conditions are satisfied.

Timing constraints. P's round-two message must arrive within time α on V's local clock from the time at which V sent its round-one message. α and β (from the timing assumption) are chosen to satisfy $\alpha \leq \beta$ and $\beta + \gamma < t$, where the value t is the time below which it is safe to assume that the timed commitment cannot be broken, even by a PRAM, and γ is an upper bound on the time it takes to create a zap by a program that is given a witness. For completeness, α must be sufficiently large to permit the necessary computations by P and the round-trip message delay.

Note that the only party that has to measure time is V, which is considered more resourceful than the prover (who may be a smart card with no independent clock) in the resettable setting.

THEOREM 8.1. *If TC is a timed commitment protocol satisfying the requirements of section* 2.5*, then for any $L \in NP$ the above protocol is rZK.*

Proof. A straight-line simulator can be constructed in a similar fashion to the construction in the proof of Theorem 7.1, thus settling the zero-knowledge issue. For soundness we use the *existence* of a decryption algorithm D with decryption key dk. If the protocol is not sound, then this key can be used to break the timed commitment in exactly the same way as the proof of knowledge was used in the proof of Theorem 7.1, violating the assumed nonuniform security of the timed commitment.

The properties of the encryption and decryption algorithms (E, D) assure us that given pd a decryption key dk exists (or the verifier will reject in step 2). Suppose now that there is a soundness adversary, succeeding on infinitely many sizes to make the verifier accept nontrue statements. For each such size we can have a slightly different prover, one that sends for size n the same key pd_n, the key that maximizes his chance of proving a false statement. This prover has at least as high a chance of proving false statements than the original adversary. Let dk_n be the decryption key of pd_n. Since the zaps generally prove true statements, the prover's chance of giving a false proof is only if $b \in_R E(z)$ (uniquely) corresponds to a $z \in \{y_0, y_1\}$. Given dk_n as the advice for size n, it is possible to obtain $z \in \{y_0, y_1\}$ and guess the value of the timed commitments. So the nonuniform advice for breaking the timed commitments is d_n, contradicting the assumption that it is secure against nonuniform adversaries.

We therefore have a nonconstructive reduction: given an algorithm for providing false proofs for L we know that there exists an algorithm for breaking the timed commitment; however, the reduction does not yield an effective method for the conversion (since there is no effective way of finding dk).

Note that a proof of security which does not yield an effective procedure to break the underlying assumptions is rare. □

9. Open questions. One vein of open problems induced by this work is with respect to the new primitive VPRG: can VPRGs be composed "á la GGM," as can ordinary pseudorandom generators? This is related to the issue of constructing VPRGs with better expansion as well as to the question whether there is a general construction of VPRFs from VPRGs. A different issue is whether VPRGs can be based on an assumption weaker than trapdoor permutations. For example, is it possible to base VPRGs on the Diffie–Hellman assumption (either computational or the decisional version, for groups without a bilinear mapping)?

What is the relationship between NIZKs in the public parameters model and NIZKs in the public random string model? The answer to this will clarify the relationship between VPRGs and NIZKs in the public parameters model.

A second vein of questions deals with efficiency and practicality. We have used general NIZKs; thus any proof must go through a reduction to an NP-complete problem. It would be useful to have more efficient, special-purpose zaps, for instance, a zap that one of x and y is a quadratic residue modulo N. Another concrete question regarding zaps is to construct one in conjunction with a timed commitment, so that it will be simple to prove consistency.

A third vein of questions deals with round efficiency: in which cases are our protocols round-optimal? It is not hard to argue that 2-round (non-black-box) zero-knowledge proofs *of knowledge* are impossible, even using timing. It is also known that, assuming $P \neq NP$, there is no 2-round proof system *with perfect completeness* for NP-hard languages either with [34] or without [2] auxiliary input. As mentioned earlier, 2-round and 1-round argument systems do exist under nonstandard assumptions [24, 4].

Acknowledgment. We thank the anonymous referees for their zealous reading of the paper and helpful suggestions.

REFERENCES

[1] B. Barak, *How to go beyond the black-box simulation barrier*, in Proceedings of the 42nd Annual IEEE Symposium on Foundation of Computer Science, 2001, pp. 106–115.

[2] B. Barak, Y. Lindell, and S. Vadhan, *Lower bounds for non-black-box zero knowledge*, in Proceedings of the 44th Annual IEEE Symposium on Foundation of Computer Science, 2003, pp. 384–393.

[3] B. Barak, S. J. Ong, and S. P. Vadhan, *Derandomization in Cryptography*, in Advances in Cryptology—CRYPTO 2003, Lecture Notes in Comput. Sci. 2729, Springer, New York, 2003, pp. 299–315.

[4] B. Barak and R. Pass, *On the possibility of one-message weak zero-knowledge*, in Proceedings of the First Annual Theory of Cryptography Conference, TCC 2004, Lecture Notes in Comput. Sci. 2951, Springer, New York, 2004, pp. 121–132.

[5] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway, *Relations among notions of security for public-key encryption schemes*, in Advances in Cryptology—CRYPTO'98, Lecture Notes in Comput. Sci. 1462, Springer, New York, 1998, pp. 26–45.

[6] M. Bellare and M. Yung, *Certifying permutations: Noninteractive zero-knowledge based on any trapdoor permutation*, J. Cryptology, 9 (1996), pp. 149–166.

[7] M. Blum, A. De Santis, S. Micali, and G. Persiano, *Noninteractive zero-knowledge*, SIAM J. Comput., 20 (1991), pp. 1084–1118.

[8] M. Blum, P. Feldman, and S. Micali, *NonInteractive zero-knowledge proof systems*, in Proceedings of the 20th Annual ACM Symposium on Theory of Computing, Chicago, 1988, pp. 103–112.

[9] M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudo-random bits*, SIAM J. Comput., 13 (1984), pp. 850–864.

[10] D. Boneh and M. Franklin, *Identity based encryption from the Weil pairing*, SIAM J. Comput., 32 (2003), pp. 586–615.

[11] D. Boneh and M. Naor, *Timed commitments*, in Advances in Cryptology—CRYPTO'2000 Proceedings, Lecture Notes in Comput. Sci. 1880, Springer, New York, 2000, pp. 236–254.

[12] G. Brassard, C. Crepeau, and J. M. Roberts, *All-or-nothing disclosure of secrets*, in Advances in Cryptology—CRYPTO '86, Lecture Notes in Comput. Sci. 263, Springer, New York, 1987, pp. 234–238.

[13] R. Canetti, O. Goldreich, S. Goldwasser, and S. Micali, *Resettable zero-knowledge*, in Proceedings of the 32nd ACM Symposium on Theory of Computing, 2000, pp. 235–244.

[14] R. Canetti, J. Kilian, E. Petrank, and A. Rosen, *Concurrent zero-knowledge requires* $\tilde{\Omega}(\log n)$ *rounds*, SIAM J. Comput., 32 (2002), pp. 1–47.

[15] I. Damgård, *Efficient concurrent zero-knowledge in the auxiliary string model*, in Advances in Cryptology—EUROCRYPT 2000, Lecture Notes in Comput. Sci. 1807, Springer, New York, 2000, pp. 418–430.

[16] Y. Dodis, *Efficient construction of (distributed) verifiable random functions*, in Public Key Cryptography—PKC 2003 Proceedings, Lecture Notes in Comput. Sci. 2567, Springer, New York, 2003, pp. 1–17.

[17] Y. Dodis and A. Yampolskiy, *A verifiable random function with short proofs and keys*, in Public Key Cryptography—PKC 2005 Proceedings, Lecture Notes in Comput. Sci. 3386, Springer, New York, 2005, pp. 416–431.

[18] D. Dolev, C. Dwork, and M. Naor, *Nonmalleable cryptography*, SIAM J. Comput., 30 (2000), pp. 391–437.

[19] C. Dwork, *The Nonmalleability Lectures*, CS 359 Course Notes, Stanford University, 1999, http://theory.stanford.edu/~gdurf/cs359-s99.

[20] C. Dwork and M. Naor, *Pricing via processing-or-combatting junk mail*, in Advances in Cryptology—CRYPTO'92, Lecture Notes in Comput. Sci. 740, Springer, New York, 1993, pp. 139–147.

[21] C. Dwork and M. Naor, *Method for Message Authentication from Nonmalleable Crypto Systems*, US Patent 05539826, 1996.

[22] C. Dwork, M. Naor, and O. Reingold, *Immunizing encryption schemes from decryption errors*, in Advances in Cryptology—EUROCRYPT 2004, Lecture Notes in Comput. Sci. 3027, Springer, New York, 2004, pp. 342–360.

[23] C. Dwork, M. Naor, and A. Sahai, *Concurrent zero-knowledge*, J. ACM, 51 (2004), pp. 851–898.

[24] C. Dwork and A. Sahai, *Concurrent zero-knowledge: Reducing the need for timing constraints*, in Advances in cryptology—CRYPTO '98, Lecture Notes in Comput. Sci. 462, Springer, New York, 1998, pp. 442–457.

[25] C. Dwork and L. J. Stockmeyer, 2*-round zero knowledge and proof auditors*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, 2002, pp. 322–331.

[26] S. Even, O. Goldreich, and A. Lempel, *A Randomized protocol for signing contracts*, Commun. ACM, 28 (1985), pp. 637–647.

[27] U. Feige, D. Lapidot, and A. Shamir, *Multiple noninteractive zero knowledge proofs under general assumptions*, SIAM J. Comput., 29 (1999), pp. 1–28.

[28] U. Feige and A. Shamir, *Witness indistinguishable and witness hiding protocols*, in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, 1990, pp. 416–426.

[29] O. Goldreich, *Foundations of Cryptography Volume* 1: *Basic Tools*, Cambridge University Press, Cambridge, UK, 2001.

[30] O. Goldreich, *Foundations of Cryptography Volume* 2: *Applications*, Cambridge University Press, Cambridge, UK, 2004.

[31] O. Goldreich, S. Goldwasser, and S. Micali, *How to construct random functions*, J. ACM, 33 (1986), pp. 792–807.

[32] O. Goldreich and H. Krawczyk, *On the composition of zero knowledge proof systems*, SIAM J. Comput., 25 (1996), pp. 169–192.

[33] O. Goldreich, S. Micali, and A. Wigderson, *Proofs that yield nothing but their validity, and a methodology of cryptographic protocol design*, J. ACM, 38 (1991), pp. 691–729.

[34] O. Goldreich and Y. Oren, *Definitions and properties of zero-knowledge proof systems,* J. Cryptology, 7 (1994), pp. 1–32.

[35] S. Goldwasser and S. Micali, *Probabilistic encryption*, J. Comput. System Sci., 28 (1984), pp. 270–299.
[36] S. Goldwasser, S. Micali, and C. Rackoff, *The knowledge complexity of interactive proof systems,* SIAM J. Comput., 18 (1989), pp. 186–208.
[37] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby, *A pseudorandom generator from any one-way function*, SIAM J. Comput., 28 (1999), pp. 1364–1396.
[38] R. Impagliazzo, M. Naor, O. Reingold, and A. Shamir, *personal communication*, 1998.
[39] A. Joux and K. Nguyen, *Separating decision Diffie–Hellman from computational Diffie–Hellman in cryptographic groups*, J. Cryptology, 16 (2003), pp. 239–247.
[40] J. Kilian and E. Petrank, *An efficient noninteractive zero-knowledge proof system for NP with general assumptions*, J. Cryptology, 11 (1998), pp. 1–27.
[41] E. Kushilevitz and R. Ostrovsky, *Replication is not needed: Single database, computationally-private information retrieval*, in Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, 1997, pp. 364–373.
[42] C. Lautemann, *BPP and the polynomial time hierarchy*, Inform. Process. Lett., 17 (1983), pp. 215–217.
[43] M. Luby, *Pseudorandomness and Cryptographic Applications*, Princeton University Press, Princeton, NJ, 1996.
[44] A. Lysyanskaya, *Unique signatures and verifiable random functions from the DH-DDH separation*, in Advances in Cryptology—CRYPTO 2002, Lecture Notes in Comput. Sci. 2442, Springer, New York, 2002, pp. 597–612.
[45] S. Micali, M. Rabin, and S. Vadhan, *Verifiable random functions*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, 1999, pp. 120–130.
[46] M. Naor, *Bit commitment using pseudo-randomness*, J. Cryptology, 4 (1991), pp. 151–158.
[47] M. Naor and B. Pinkas, *Efficient oblivious transfer protocols*, in Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2001, pp. 448–457.
[48] M. Naor and O. Reingold, *Synthesizers and their application to the parallel construction of pseudo-random functions*, J. Comput. Systems Sci., 58 (1999), pp. 336-375.
[49] M. Naor and M. Yung, *Public-key cryptosystems provably secure against chosen ciphertext attacks*, in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, Baltimore, 1990, pp. 427–437.
[50] M. O. Rabin, *How to Exchange Secrets by Oblivious Transfer*, Tech. Memo TR-81, Aiken Computation Laboratory, Harvard University, Cambridge, MA, 1981.
[51] A. Rosen, *A note on the round-complexity of concurrent zero-knowledge*, in Advances in Cryptology—CRYPTO'2000 Proceedings, Lecture Notes in Comput. Sci. 1880, Springer, New York, 2000, pp. 451–468.
[52] A. Shamir, *On the generation of cryptographically strong pseudorandom sequences*, ACM Trans. Comput. Systems, 1 (1983), pp. 38–44.

SIAM J. COMPUT.
Vol. 36, No. 6, pp. 1544–1569

COMPLEXITY OF SELF-ASSEMBLED SHAPES*

DAVID SOLOVEICHIK† AND ERIK WINFREE†

Abstract. The connection between self-assembly and computation suggests that a shape can be considered the output of a self-assembly "program," a set of tiles that fit together to create a shape. It seems plausible that the size of the smallest self-assembly program that builds a shape and the shape's descriptional (Kolmogorov) complexity should be related. We show that when using a notion of a shape that is independent of scale, this is indeed so: in the tile assembly model, the minimal number of distinct tile types necessary to self-assemble a shape, at some scale, can be bounded both above and below in terms of the shape's Kolmogorov complexity. As part of the proof, we develop a universal constructor for this model of self-assembly that can execute an arbitrary Turing machine program specifying how to grow a shape. Our result implies, somewhat counterintuitively, that self-assembly of a scaled-up version of a shape often requires fewer tile types. Furthermore, the independence of scale in self-assembly theory appears to play the same crucial role as the independence of running time in the theory of computability. This leads to an elegant formulation of languages of shapes generated by self-assembly. Considering functions from bit strings to shapes, we show that the running-time complexity, with respect to Turing machines, is polynomially equivalent to the scale complexity of the same function implemented via self-assembly by a finite set of tile types. Our results also hold for shapes defined by Wang tiling—where there is no sense of a self-assembly process—except that here time complexity must be measured with respect to nondeterministic Turing machines.

Key words. Kolmogorov complexity, scaled shapes, self-assembly, Wang tiles, universal constructor

AMS subject classifications. 68Q30, 68Q05, 52C20, 52C45

DOI. 10.1137/S0097539704446712

1. Introduction.

Self-assembly is the process by which an organized structure can spontaneously form from simple parts. The tile assembly model [22, 21], based on Wang tiling [20], formalizes the two-dimensional self-assembly of square units called "tiles" using a physically plausible abstraction of crystal growth. In this model, a new tile can adsorb to a growing complex if it binds strongly enough. Each of the four sides of a tile has an associated bond type that interacts with a certain strength with matching sides of other tiles. The process of self-assembly is initiated by a single seed tile and proceeds via the sequential addition of new tiles. Confirming the physical plausibility and relevance of the abstraction, simple self-assembling systems of tiles have been built out of certain types of DNA molecules [23, 15, 14, 12, 10]. The possibility of using self-assembly for nanofabrication of complex components such as circuits has been suggested as a promising application [6].

The view that the "shape" of a self-assembled complex can be considered the output of a computational process [2] has inspired recent interest [11, 1, 3, 9, 4]. While it was shown through specific examples that self-assembly can be used to construct interesting shapes and patterns, it was not known in general which shapes could be self-assembled from a small number of tile types. Understanding the complexity of

*Received by the editors December 21, 2004; accepted for publication (in revised form) June 30, 2006; published electronically February 9, 2007. An extended abstract version of this work was previously published as *Complexity of self-assembled shapes*, in DNA Computing, Lecture Notes in Comput. Sci. 3384, Springer, Berlin, 2005, pp. 344–354. This work was supported by NSF CAREER grant 0093486.

http://www.siam.org/journals/sicomp/36-6/44671.html

†California Institute of Technology, Pasadena, CA 91125 (dsolov@caltech.edu, winfree@caltech.edu).

shapes is facilitated by an appropriate definition of shape. In our model, a tile system generates a particular shape if it produces any scaled version of that shape (section 3). This definition may be thought to formalize the idea that a structure can be made up of arbitrarily small pieces, but more importantly this leads to an elegant theory that is impossible to achieve without ignoring scale. Computationally, it is analogous to disregarding computation time and is thus more appropriate as a notion of output of a *universal* computation process.[1] Using this definition of shape, we show (section 4) that for any shape $\tilde{S}$, if $K_{sa}(\tilde{S})$ is the minimal number of distinct tile types necessary to self-assemble it, then $K_{sa}(\tilde{S}) \log K_{sa}(\tilde{S})$ is within multiplicative and additive constants (independent of $\tilde{S}$) of the shape's Kolmogorov complexity. This theorem is proved by developing a universal constructor [19] for self-assembly which uses a program that outputs a fixed size shape as a list of locations to make a scaled version of the shape (section 5). This construction, together with a new proof technique for showing that a tile set produces a unique assembly (*local determinism*), might be of independent interest. Our result ties the computation of a shape and its self-assembly and, somewhat counterintuitively, implies that it may often require fewer tile types to self-assemble a larger instance of a shape than a smaller instance thereof. Another consequence of the theorem is that the minimal number of tile types necessary to self-assemble an arbitrary scaling of a shape is uncomputable. Answering the same question about shapes of a fixed size is computable but NP-complete [1].

The tight correspondence between computation (ignoring time) and self-assembly (ignoring scale) suggests that complexity measures based on time (for computation) and on scale (for self-assembly) could also be related. To establish this result, we consider "programmable" tile sets that will grow a particular member of a family of shapes, dependent upon input information present in an initial seed assembly. We show that, as a function of the length of the input information, the number of tiles present in the shape (a measure of its scale) is polynomially related to the time required for a Turing machine (TM) to produce a representation of the same shape. Furthermore, we discuss the relationship between complexities for Wang tilings (in which the existence of a tiling rather than its creation by self-assembly is of relevance) and for self-assembly, and we show that while the Kolmogorov complexity is unchanged, the scale complexity for Wang tilings is polynomially related to the time for *nondeterministic* TMs. These results are presented in section 6.

2. The tile assembly model. We present a description of the tile assembly model based on Rothemund and Winfree [11] and Rothemund [9]. We will be working on a $\mathbb{Z} \times \mathbb{Z}$ grid of unit square locations. The *directions* $\mathcal{D} = \{N, E, S, W\}$ are used to indicate relative positions in the grid. Formally, they are functions $\mathbb{Z} \times \mathbb{Z} \to \mathbb{Z} \times \mathbb{Z}$: $N(i, j) = (i, j + 1)$, $E(i, j) = (i + 1, j)$, $S(i, j) = (i, j - 1)$, and $W(i, j) = (i - 1, j)$. The inverse directions are defined naturally: $N^{-1}(i, j) = S(i, j)$, etc. Let Σ be a set of *bond types*. A *tile type* t is a 4-tuple $(\sigma_N, \sigma_E, \sigma_S, \sigma_W) \in \Sigma^4$ indicating the associated bond types on the north, east, south, and west sides. Note that tile types are oriented; thus a rotated version of a tile type is considered to be a different tile type. A special bond type *null* represents the lack of an interaction, and the

[1] The production of a shape of a fixed size cannot be considered the output of a universal computation process. Whether a universal process will output a given shape is an undecidable question, whereas this can be determined by exhaustive enumeration in the tile assembly model. Thus it is clear that the connection between Kolmogorov complexity and the number of tile types we obtain in our main result (section 4) cannot be achieved for fixed-scale shapes: this would violate the uncomputability of Kolmogorov complexity.

special tile type $empty = (null, null, null, null)$ represents an empty space. If T is a set of tile types, a *tile* is a pair $(t, (i, j)) \in T \times \mathbb{Z}^2$ indicating that location (i, j) contains the tile type t. Given the tile $t = (t, (i, j))$, $type(t) = t$ and $pos(t) = (i, j)$. Further, $bond_D(t)$, where $D \in \mathcal{D}$, is the bond type of the respective side of t, and $bond_D(t) = bond_D(type(t))$. A *configuration* is a set of non*empty* tiles, with types from T, such that there is no more than one tile in every location $(i, j) \in \mathbb{Z} \times \mathbb{Z}$. For any configuration A, we write $A(i, j)$ to indicate the tile at location (i, j) or the tile $(empty, (i, j))$ if there is no tile in A at this location.

A *strength function* $g : \Sigma \times \Sigma \to \mathbb{Z}$, where $null \in \Sigma$, defines the interactions between adjacent tiles: we say that a tile t_1 interacts with its neighbor t_2 with strength $\Gamma(t_1, t_2) = g(\sigma, \sigma')$, where σ is the bond type of tile t_1 that is adjacent to the bond type σ' of tile t_2.[2] The *null* bond has a zero interaction strength (i.e., $\forall \sigma \in \Sigma$, $g(null, \sigma) = 0$). We say that a strength function is *diagonal* if it is nonzero only for $g(\sigma, \sigma')$ such that $\sigma = \sigma'$. Unless otherwise noted, a tile system is assumed to have a diagonal strength function. Our constructions use diagonal strength functions with the range $\{0, 1, 2\}$. We say that a bond type σ has *strength* $g(\sigma, \sigma)$. Two tiles are *bonded* if they interact with a positive strength. For a configuration A, we use the notation $\Gamma^A_D(t) = \Gamma(t, A(D(pos(t))))$.[3] For $L \subseteq \mathcal{D}$ we define $\Gamma^A_L(t) = \sum_{D \in L} \Gamma^A_D(t)$.

A *tile system* $\mathbf{T}$ is a quadruple (T, t_s, g, τ) where T is a finite set of nonempty tile types, t_s is a special *seed tile*[4] with $type(t_s) \in T$, g is a strength function, and τ is the threshold parameter. Self-assembly is defined by a relation between configurations. Suppose A and B are two configurations, and t is a tile such that $A = B$ except at $pos(t)$ and $A(pos(t)) = null$ but $B(pos(t)) = t$. Then we write $A \to_{\mathbf{T}} B$ if $\Gamma^A_{\mathcal{D}}(t) \geq \tau$. This means that a tile can be added to a configuration if and only if the sum of its interaction strengths with its neighbors reaches or exceeds τ. The relation $\to^*_{\mathbf{T}}$ is the reflexive transitive closure of $\to_{\mathbf{T}}$.

Whereas a configuration can be any arrangement of tiles (not necessarily connected), we are interested in the subclass of configurations that can result from a self-assembly process. Formally, the tile system and the relation $\to^*_{\mathbf{T}}$ define the partially ordered set of *assemblies*, $Prod(\mathbf{T}) = \{A \text{ such that (s.t.) } \{t_s\} \to^*_{\mathbf{T}} A\}$, and the set of *terminal assemblies*, $Term(\mathbf{T}) = \{A \in Prod(\mathbf{T}) \text{ and } \nexists B \neq A \text{ s.t. } A \to^*_{\mathbf{T}} B\}$. A tile system $\mathbf{T}$ *uniquely produces* A if $\forall B \in Prod(\mathbf{T})$, $B \to^*_{\mathbf{T}} A$ (which implies $Term(\mathbf{T}) = \{A\}$).

An *assembly sequence* $\vec{A}$ of $\mathbf{T}$ is a sequence of pairs (A_n, t_n), where $A_0 = \{t_0\} = \{t_s\}$ and $A_{n-1} \to_{\mathbf{T}} A_n = A_{n-1} \cup \{t_n\}$. Here we will exclusively consider finite assembly sequences. If a finite assembly sequence $\vec{A}$ is implicit, A indicates the last assembly in the sequence.

The tile systems used in our constructions have $\tau = 2$ with the strength function ranging over $\{0, 1, 2\}$. It is known that $\tau = 1$ systems with strength function ranging over $\{0, 1\}$ are rather limited [11, 9]. In our drawings, the bond type σ may be

[2] More formally,

$$\Gamma(t_1, t_2) = \begin{cases} g(bond_{D^{-1}}(t_1), bond_D(t_2)) \text{ if } \exists D \in \mathcal{D} \text{ s.t. } pos(t_1) = D(pos(t_2)), \\ 0 \text{ otherwise.} \end{cases}$$

[3] Note that $t \neq A(pos(t))$ is a valid choice. In that case $\Gamma^A_D(t)$ tells us how t would bind if it were in A.

[4] While having a single seed tile is appropriate to the complexity discussion of the main part of this paper, it is useful to consider whole *seed assemblies* (made up of tiles not necessarily in T) when considering tile systems capable of producing multiple shapes (section 6.5).

illustrated by a combination of shading, various graphics, and symbols. Strength-2 bond types will always contain two dots in their representation. All markings must match for two bond types to be considered identical. For example, the north bond type of the following tile has strength 2, and the others have strength 1.

The constructions in this paper do not use strength-0 bond types (other than in *empty* tiles); thus, there is no confusion between strength-1 and strength-0 bond types. Strength-0 interactions due to mismatches between adjacent tiles do occur in our constructions.

2.1. Guaranteeing unique production. When describing tile systems that produce a desired assembly, we would like an easy method for showing that this assembly is uniquely produced. While it might be easy to find an assembly sequence that leads to a particular assembly, there might be many other assembly sequences that lead elsewhere. Here we present a property of an assembly sequence that guarantees that the assembly it produces is indeed the uniquely produced assembly of the tile system.

Rothemund [9] describes the deterministic-RC property of an assembly that guarantees its unique production and is very easy to check. However, this property is satisfied only by convex (in the sense of polyaminos) assemblies and thus cannot be directly invoked when making arbitrary shapes.[5] A more general poly-time test for unique production was also shown by Rothemund [9], but it can be difficult to prove that a particular assembly would satisfy this test. On the other hand, the notion of locally deterministic assembly sequences introduced here is easily checkable and sufficient for the constructions in this paper.

DEFINITION 2.1. *For an assembly sequence $\vec{A}$ we define the following sets of directions for $\forall i, j \in \mathbb{Z}$, letting $t = A(i, j)$:*

- $inputsides^{\vec{A}}(t) = \{D \in \mathcal{D} \text{ s.t. } t = t_n \text{ and } \Gamma_D^{A_n}(t_n) > 0\}$,
- $propsides^{\vec{A}}(t) = \{D \in \mathcal{D} \text{ s.t. } D^{-1} \in inputsides^{\vec{A}}(A(D(pos(t))))\}$, *and*
- $termsides^{\vec{A}}(t) = \mathcal{D} - inputsides^{\vec{A}}(t) - propsides^{\vec{A}}(t)$.

Intuitively, *inputsides* are the sides with which the tile initially binds in the process of self-assembly; these sides determine its identity. *propsides* propagate information by being the sides to which neighboring tiles bind. *termsides* are sides that do neither. Note that by definition *empty* tiles have four *termsides*.

DEFINITION 2.2. *A finite assembly sequence $\vec{A}$ of $\mathbf{T} = (T, t_s, g, \tau)$ is called* locally deterministic *if $\forall i, j \in \mathbb{Z}$, letting $t = A(i, j)$,*

(1) $\Gamma^A_{inputsides^{\vec{A}}(t)}(t) \leq \tau$, *and*

(2) $\forall t'$ *s.t.* $type(t') \in T$, $pos(t') = pos(t)$ *but* $type(t') \neq type(t)$,

$$\Gamma^A_{\mathcal{D}-propsides^{\vec{A}}(t)}(t') < \tau.$$

We allow the possibility of $<$ in property (1) in order to account for the seed and *empty* tiles. Intuitively, the first property says that when a new tile binds to

[5] Additionally, assemblies satisfying the deterministic-RC property must have no strength-0 interactions between neighboring nonempty tiles. However, such interactions are used in our construction.

a growing assembly, it binds "just barely." The second property says that nothing can grow from nonpropagating sides except "as desired." We say that $\mathbf{T}$ is locally deterministic if there exists a locally deterministic assembly sequence for it.

It is clear that if $\vec{A}$ is a locally deterministic assembly sequence of $\mathbf{T}$, then $A \in Term(\mathbf{T})$. Otherwise, the *empty* tile in the position where a new (nonempty) tile can be added to A would violate the second property. However, the existence of a locally deterministic assembly sequence leads to the following much stronger conclusion.

THEOREM 2.3. *If there exists a locally deterministic assembly sequence $\vec{A}$ of $\mathbf{T}$, then $\mathbf{T}$ uniquely produces A.*

Proof. See Appendix A. □

3. Arbitrarily scaled shapes and their complexity. In this section, we introduce the model for the output of the self-assembly process used in this paper. Let S be a finite set of locations on $\mathbb{Z} \times \mathbb{Z}$. The adjacency graph $G(S)$ is the graph on S defined by the adjacency relation where two locations are considered adjacent if they are directly north/south or east/west of one another. We say that S is a *coordinated shape* if $G(S)$ is connected.[6] The *coordinated shape of assembly* A is the set $S_A = \{pos(t) \text{ s.t. } t \in A\}$. Note that S_A is a coordinated shape because A constitutes a single connected component.

For any set of locations S, and any $c \in \mathbb{Z}^+$, we define a *c-scaling of* S as

$$S^c = \{(i, j) \text{ s.t. } (\lfloor i/c \rfloor, \lfloor j/c \rfloor) \in S\}.$$

Geometrically, this represents a "magnification" of S by a factor c. Note that a scaling of a coordinated shape is itself a coordinated shape: every node of $G(S)$ gets mapped to a c^2-node connected subgraph of $G(S^c)$, and the relative connectivity of the subgraphs is the same as the connectivity of the nodes of $G(S)$. A parallel argument shows that if S^c is a coordinated shape, then so is S. We say that coordinated shapes S_1 and S_2 are *scale-equivalent* if $S_1^c = S_2^d$ for some $c, d \in \mathbb{Z}^+$. Two coordinated shapes are *translation-equivalent* if they can be made identical by translation. We write $S_1 \cong S_2$ if S_1^c is translation-equivalent to S_2^d for some $c, d \in \mathbb{Z}^+$. Scale-equivalence, translation-equivalence, and $\cong$ are equivalence relations (see Appendix B). This defines the equivalence classes of coordinated shapes under $\cong$. The equivalence class containing S is denoted $\tilde{S}$ and we refer to it as the *shape* $\tilde{S}$. We say that $\tilde{S}$ is the *shape of assembly* A if $S_A \in \tilde{S}$. The view of computation performed by the self-assembly process espoused here is the production of a shape as the "output" of the self-assembly process, with the understanding that the scale of the shape is irrelevant. Physically, this view may be appropriate to the extent that a physical object can be constructed from arbitrarily small pieces. However, the primary reason for this view is that there does not seem to be a comprehensive theory of complexity of coordinated shapes akin to the theory we develop here for shapes ignoring scale.

Having defined the notion of shapes, we turn to their descriptional complexity. As usual, the Kolmogorov complexity of a binary string x with respect to a universal TM U is $K_U(x) = \min\{|p| \text{ s.t. } U(p) = x\}$. (See the exposition of Li and Vitanyi [13] for an in-depth discussion of Kolmogorov complexity.) Let us fix some "standard" universal machine U. We call the Kolmogorov complexity of a coordinated shape S

[6] We say "coordinated" to make explicit that a fixed coordinate system is used. We reserve the unqualified term "shape" for when we ignore scale and translation.

to be the size of the smallest program outputting it as a list of locations:[7,8]

$$K(S) = \min\{|s| \text{ s.t. } U(s) = \langle S\rangle\}.$$

The Kolmogorov complexity of a shape $\tilde{S}$ is

$$K(\tilde{S}) = \min\left\{|s| \text{ s.t. } U(s) = \langle S\rangle \text{ for some } S \in \tilde{S}\right\}.$$

We define the *tile-complexity* of a coordinated shape S and shape $\tilde{S}$, respectively, as

$$K_{sa}(S) = \min\left\{\begin{array}{l} n \text{ s.t. } \exists \text{ a tile system } \mathbf{T} \text{ of } n \text{ tile types that uniquely produces} \\ \text{assembly } A \text{ and } S \text{ is the coordinated shape of } A \end{array}\right\},$$

$$K_{sa}(\tilde{S}) = \min\left\{\begin{array}{l} n \text{ s.t. } \exists \text{ a tile system } \mathbf{T} \text{ of } n \text{ tile types that uniquely produces} \\ \text{assembly } A \text{ and } \tilde{S} \text{ is the shape of } A \end{array}\right\}.$$

4. Relating tile-complexity and Kolmogorov complexity. The essential result of this paper is the description of the relationship between the Kolmogorov complexity of any shape and the number of tile types necessary to self-assemble it.

THEOREM 4.1. *There exist constants a_0, b_0, a_1, b_1 such that for any shape $\tilde{S}$,*

$$a_0 K(\tilde{S}) + b_0 \leq K_{sa}(\tilde{S}) \log K_{sa}(\tilde{S}) \leq a_1 K(\tilde{S}) + b_1. \tag{4.1}$$

Note that since any tile system of n tile types can be described by $O(n \log n)$ bits, the theorem implies that there is a way to construct a tiling system such that asymptotically at least a constant fraction of these bits is used to "describe" the shape rather than any other aspect of the tiling system.

Proof of Theorem 4.1. To see that $a_0 K(\tilde{S}) + b_0 \leq K_{sa}(\tilde{S}) \log K_{sa}(\tilde{S})$, realize that there exists a constant size program p_{sa} that, given a binary description of a tile system, simulates its self-assembly, making arbitrary choices where multiple tile additions are possible. If the self-assembly process terminates, p_{sa} outputs the coordinated shape of the terminal assembly as the binary encoding of the list of locations in it. Any tile system $\mathbf{T}$ of n tile types with any diagonal strength function and any threshold τ can be represented[9] by a string $d_{\mathbf{T}}$ of $4n\lceil \log 4n\rceil + 16n$ bits: for each tile type, the first of which is assumed to be the seed, specify the bond types on its four sides. There are no more than $4n$ bond types. In addition, for each tile type t specify for which of the 16 subsets $L \subseteq \mathcal{D}$, $\sum_{D\in L} g(bond_D(t)) \geq \tau$. If $\mathbf{T}$ is a tile system uniquely producing an assembly that has shape $\tilde{S}$, then $K(\tilde{S}) \leq |p_{sa} d_{\mathbf{T}}|$. The left inequality in (4.1) follows with the multiplicative constant $a_0 = 1/4 - \varepsilon$ for arbitrary $\varepsilon > 0$.

We prove the right inequality in (4.1) by developing a construction (section 5) showing how, for any program s s.t. $U(s) = \langle S\rangle$, we can build a tile system $\mathbf{T}$ of

[7] Note that $K(S)$ is within an additive constant of $K_U(x)$ where x is some other effective description of S, such as a computable characteristic function or a matrix. Since our results are asymptotic, they are independent of the specific representation choice. One might also consider invoking a two-dimensional computing machine, but it is not fundamentally different for the same reason.

[8] Notation $\langle\cdot\rangle$ indicates some standard binary encoding of the object(s) in the brackets. In the case of coordinated shapes, it means an explicit binary encoding of the set of locations. Integers, tuples, or other data structures are similarly given simple explicit encodings.

[9] Note that this representation could also be used in the case that negative bond strengths are allowed so long as the strength function is diagonal.

$15\frac{|p|}{\log|p|} + b$ tile types, where b is a constant and p is a string consisting of a fixed program p_{sb} and s (i.e., $|p| = |p_{sb}| + |s|$), that uniquely produces an assembly whose shape is $\tilde{S}$. Program p_{sb} and constant b are both independent of S. The right inequality in (4.1) follows with the multiplicative constant $a_1 = 15 + \varepsilon$ for arbitrary $\varepsilon > 0$. □

Our result can be used to show that the tile-complexity of shapes is uncomputable.

COROLLARY 4.2. *K_{sa} of shapes is uncomputable. In other words, the following language is undecidable: $\tilde{L} = \{(l, n)$ s.t. $l = \langle S\rangle$ for some S and $K_{sa}(\tilde{S}) \leq n\}$.*

Language $\tilde{L}$ should be contrasted with $L = \{(l, n)$ s.t. $l = \langle S\rangle$ and $K_{sa}(S) \leq n\}$ which is decidable (but hard to compute in the sense of NP-completeness [1]).

Proof of Corollary 4.2. We essentially parallel the proof that Kolmogorov complexity is uncomputable. If $\tilde{L}$ were decidable, then we could make a program that computes $K_{sa}(\tilde{S})$ and subsequently uses Theorem 4.1 to compute an effective lower bound for $K(\tilde{S})$. Then we can construct a program p that, given n, outputs some coordinated shape S (as a list of locations) such that $K(\tilde{S}) \geq n$ by enumerating shapes and testing with the lower bound, which we know must eventually exceed n. But this results in a contradiction since $p\langle n\rangle$ is a program outputting $S \in \tilde{S}$ and so $K(\tilde{S}) \leq |p| + \lceil \log n\rceil$. But for large enough n, $|p| + \lceil \log n\rceil < n$. □

5. The programmable block construction.

5.1. Overview. The uniquely produced terminal assembly A of our tile system logically will consist of square "blocks" of $c \times c$ tiles. There will be one block for each location in S. Consider the coordinated shape in Figure 5.1(a). An example assembly A is graphically represented in Figure 5.1(b), where each square represents a block containing c^2 tiles. Self-assembly initiates in the *seed block*, which contains the seed tile, and proceeds according to the arrows illustrated between blocks. Thus if there is an arrow from one block to another, it indicates that the growth of the second block (a *growth block*) is initiated from the first. A terminated arrow indicates that the block does not initiate the self-assembly of an adjacent block in that direction—in fact, the boundary between such blocks consists of strength-0 interactions (i.e., mismatches). Figure 5.1(c) describes our nomenclature: an arrow comes into a block on its input side, arrows exit on propagating output sides, and terminated arrows indicate terminating output sides. The seed block has four output sides, which can be either propagating or terminating. Each growth block has one input and three output sides, which are also either propagating or terminating. The overall pattern of bonding of the finished target assembly A is as follows. Tiles on terminal output sides are not bound to the tiles on the adjacent terminal output side (i.e., there is no bonding along the dotted lines in Figure 5.8(a)), but all other neighboring tiles are bound. We will program the growth such that terminating output sides abut only other terminating output sides or *empty* tiles, and input sides exclusively abut propagating output sides, and vice versa.

The input/output connections of the blocks form a spanning tree rooted at the seed block. During the progress of the self-assembly of the seed block, a computational process determines the input/output relationships of the rest of the blocks in the assembly. This information is propagated from block to block during self-assembly (along the arrows in Figure 5.1(b)) and describes the shape of the assembly. By following the instructions each growth block receives in its input, the block decides where to start the growth of the next block and what information to pass to it in turn. The scaling factor c is set by the size of the seed block. The computation in

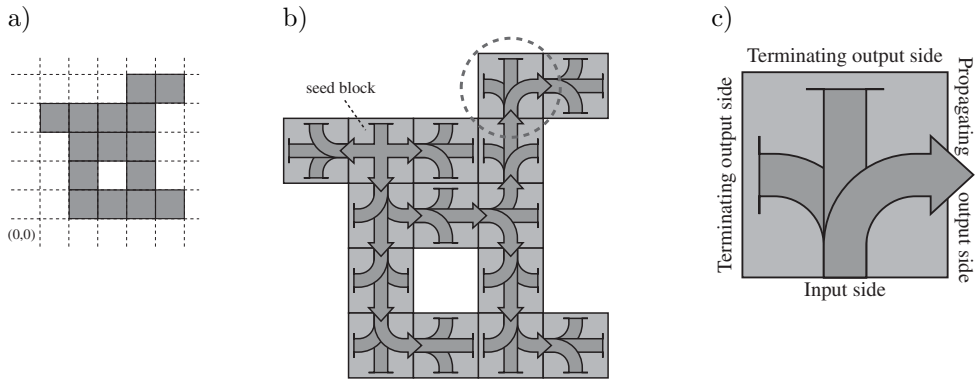


FIG. 5.1. *Forming a shape out of blocks:* (a) *A coordinated shape S.* (b) *An assembly composed of $c \times c$ blocks that grow according to transmitted instructions such that the shape of the final assembly is $\tilde{S}$ (not drawn to scale). Arrows indicate information flow and order of assembly. The seed block and the circled growth block are schematically expanded in Figure* 5.2. (c) *The nomenclature describing the types of block sides.*

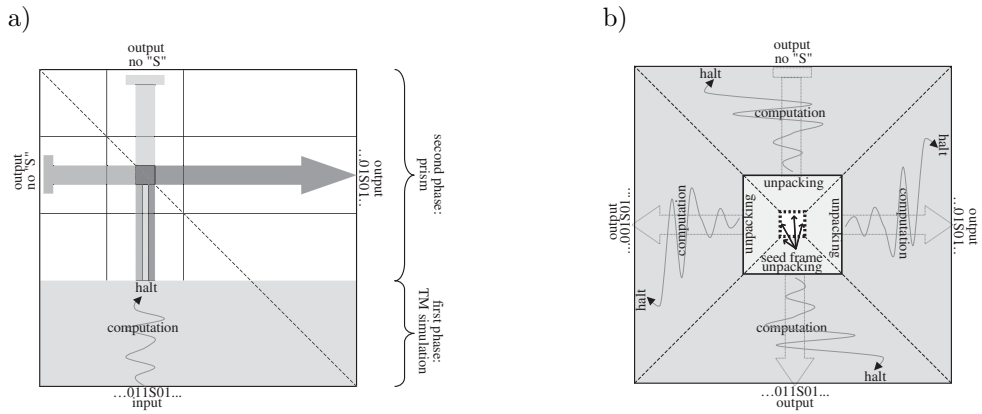


FIG. 5.2. *Internal structure of a growth block* (a) *and seed block* (b).

the seed block ensures that c is large enough so that there is enough space to do the necessary computation within the other blocks.

We present a general construction that represents a Turing-universal way of guiding large-scale self-assembly of blocks based on an input program p. In the following section, we describe the architecture of seed and growth blocks on which arbitrary programs can be executed. In section 5.3 we describe how program p can be encoded using few tile types. In section 5.4 we discuss the programming of p that is required to grow the blocks in the form of a specific shape and bound the scaling factor c. In section 5.5 we demonstrate that the target assembly A is *uniquely* produced.

5.2. Architecture of the blocks.

5.2.1. Growth blocks. There are four types of growth blocks depending upon where the input side is, which will be labeled by $\uparrow$, $\rightarrow$, $\downarrow$, or $\leftarrow$. The internal structure of a $\uparrow$ growth block is schematically illustrated in Figure 5.2(a). The other three types of growth block are rotated versions of the $\uparrow$ block. The specific tile types used for a

↑ growth block are shown in Figure 5.3, and a simple example is presented in Figure 5.4. The first part is a TM simulation, which is based on [18, 11]. The machine simulated is a universal TM that takes its input from the propagating output side of the previous block. This TM has an output alphabet $\{0, 1, S\}^3$ and an input alphabet $\{(000), (111)\}$ on a two-way tape (with λ used as the blank symbol). The output of the simulation, as 3-tuples, is propagated until the diagonal. The diagonal propagates each member of the 3-tuples crossing it to one of the three output sides, like a prism separating the colors of the spectrum. This allows the single TM simulation to produce three separate strings targeted for the three output sides. The "S" symbol in the output of the TM simulation is propagated like the other symbols. However, it acts in a special way when it crosses the boundary tiles at the three output sides of the block, where it starts a new block. The output sides that receive the "S" symbol become propagating output sides, and the output sides that do not receive it become terminating output sides. In this way, the TM simulation decides which among the three output sides will become propagating output sides, and what information they should contain, by outputting appropriate tuples. Subsequent blocks will use this information as a program, as discussed in section 5.4.

5.2.2. Seed block. The internal structure of the seed block is schematically shown in Figure 5.2(b). It consists of a small square containing all the information pertaining to the shape to be built (the seed frame), a larger square in which this information is unpacked into usable form, and finally four TM simulations whose computations determine the size of the seed block and the information transmitted to the growth blocks. For simplicity we first present a construction without the unpacking process (the *simple* seed block) and then explain the unpacking process separately and show how it can be used to create the full construction. The tile types used for the simple seed block are presented in Figure 5.5, and an example is given in Figure 5.6. While growth blocks contain a single TM simulation that outputs a different string to each of the three output sides, the seed block contains four *identical* TM simulations that output different strings to each of the four output sides. This is possible because the border tile types transmit information selectively: the computation in the seed block is performed using 4-tuples as the alphabet in a manner similar to that of the growth blocks, but on each side of the seed block only one of the elements of the 4-tuple traverses the border. As with growth blocks, if the transmitted symbol is "S," the outside edge initiates the assembly of the adjoining block. The point of having four identical TM simulations is to ensure that the seed block is square: while a growth block uses the length of its input side to set the length of its output sides (via the diagonal), the seed block does not have any input sides. (Remember that it is the seed block that sets the size of all the blocks.)

The initiation of the TM simulations in the seed block is done by tile types encoding the program p that guides the block construction. The natural approach to providing this input is using four rows (one for each TM) of unique tiles encoding one bit per tile, as illustrated in Figures 5.5 and 5.6. However, this method does not result in an asymptotically optimal encoding.

5.3. The unpacking process. To encode bits much more effectively we follow Adleman et al. [3] and encode on the order of $\log n / \log \log n$ bits per tile, where n is the length of the input. This representation is then unpacked into a one-bit-per-tile representation used by the TM simulation. The method of Adleman et al. requires $O(n/\log n)$ tiles to encode n bits, leading to the asymptotically optimal result of Theorem 4.1.

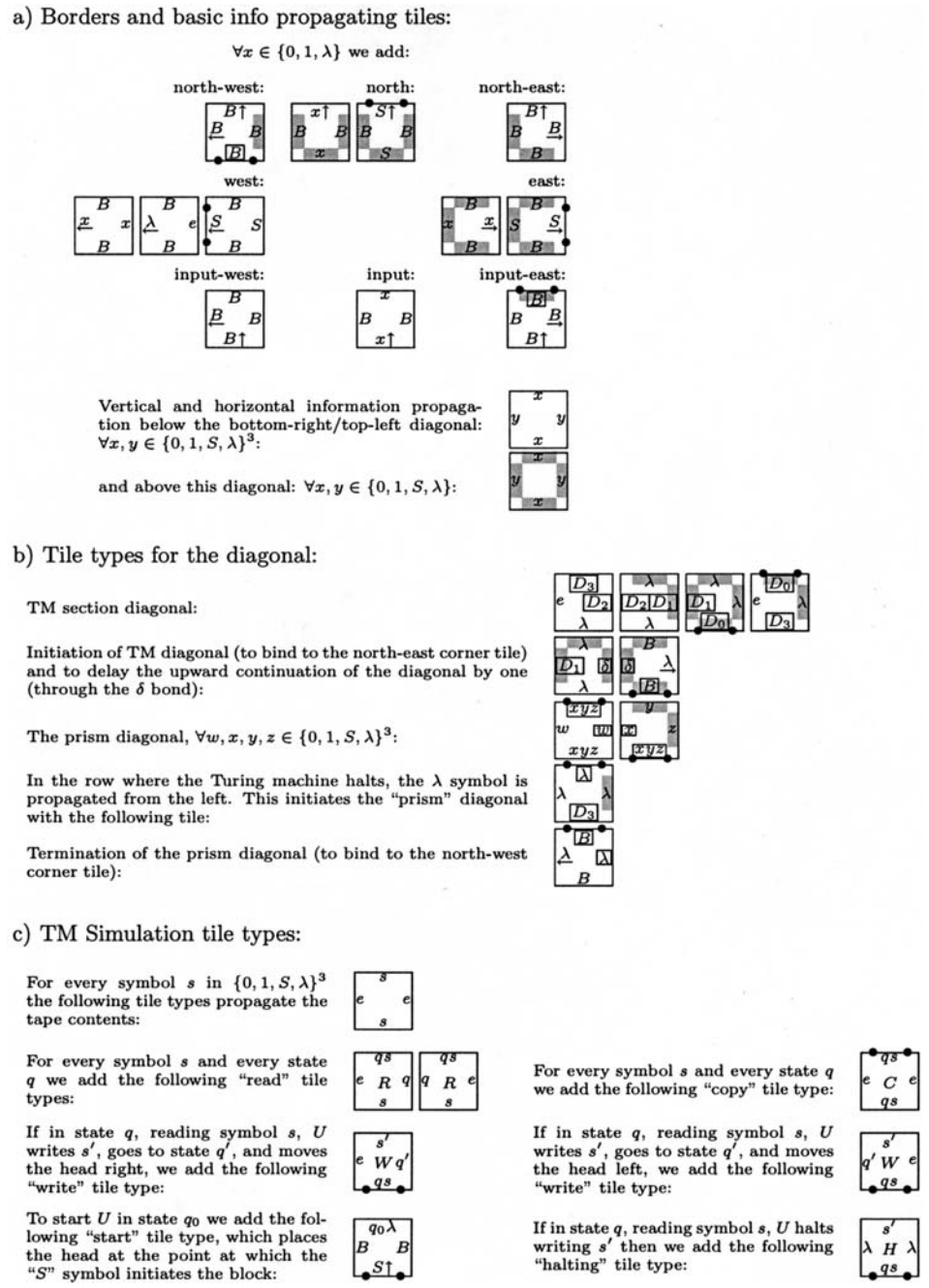


FIG. 5.3. *Growth block $\uparrow$ tile types. All bond types in which a block type symbol is omitted have the block type symbol "$\uparrow$" to prevent inadvertent incorporation of tiles from a different block type. We assume that in bond types above, a single symbol $x \in \{0,1,S,\lambda\}$ is the same as the tuplet (xxx). The tile types for other growth block types are formed by $90°, 180°$, and $270°$ rotations of the tile types of the $\uparrow$ block where the block type symbols $\{\uparrow,\downarrow,\leftarrow,\rightarrow\}$ are replaced by a corresponding $90°, 180°$, and $270°$ rotation of the symbol: i.e.,*  *($\uparrow$ growth block) $\Rightarrow$*  *($\rightarrow$ growth block). Looking at the border tile types, note that external sides of tiles on output sides of blocks have block type symbols compatible with the tiles on an input side of a block. However, tiles on output sides cannot bind to the tiles on an adjacent output side because of mismatching block type symbols.*

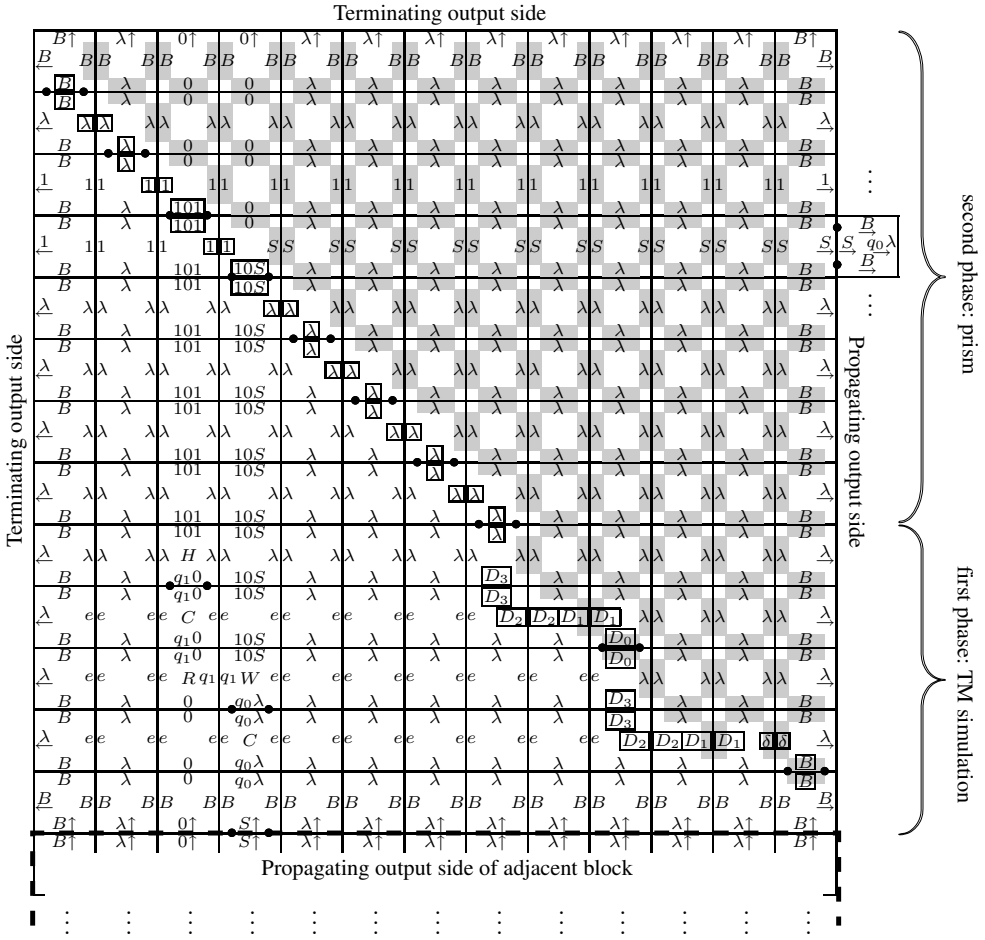


FIG. 5.4. *A trivial example of a ↑ growth block. Here, the TM makes one state transition and halts. All bond types in which a block type symbol is omitted have the block type symbol "↑." We assume that in bond types above, a single symbol $x \in \{0, 1, S, \lambda\}$ is the same as the tuplet (xxx). The natural assembly sequence to consider is adding tiles row by row from the south side (in which a new row is started by the strength-2 bond).*

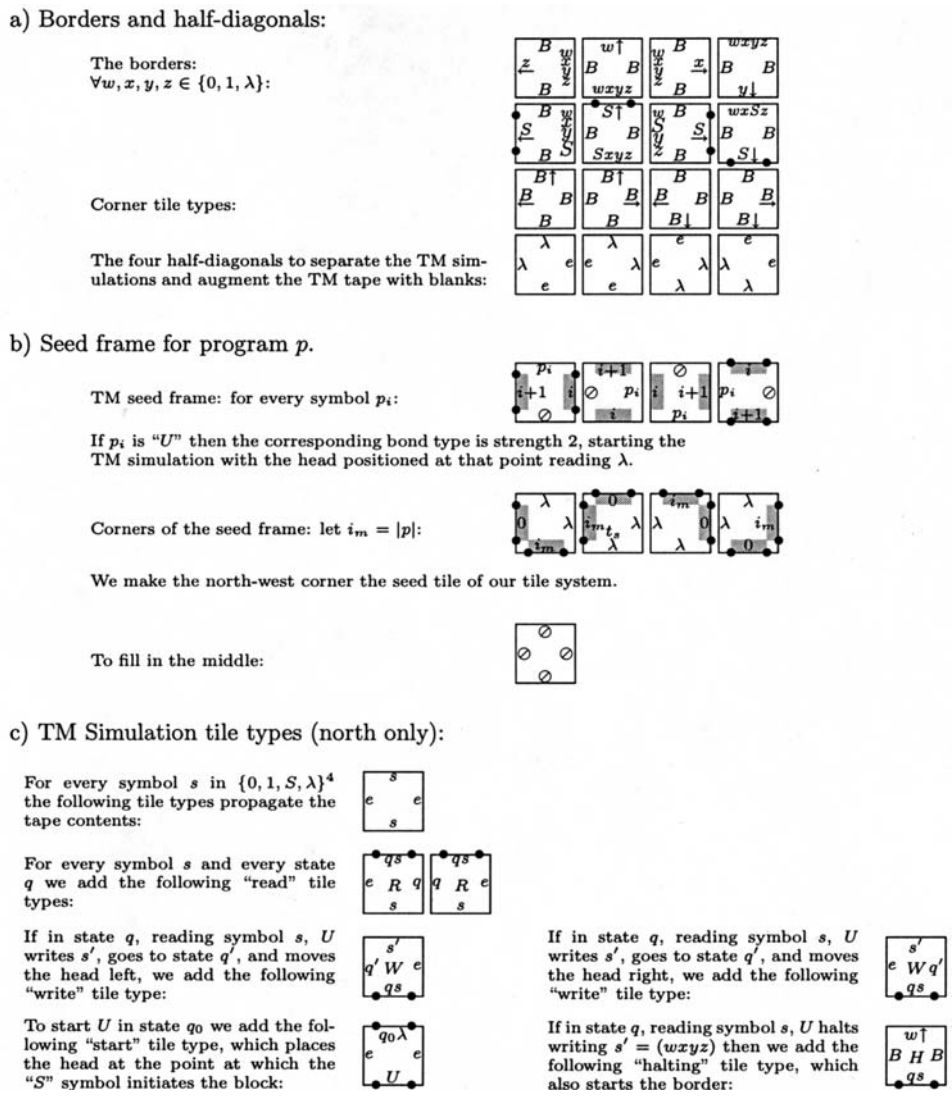


FIG. 5.5. *Seed block tile types without unpacking. All bond types in which a block type symbol is omitted have the block type symbol "☆" to prevent inadvertent incorporation of tiles from a different block type. We assume that in bond types above, a single symbol $x \in \{0, 1, S, \lambda\}$ is the same as the tuplet $(xxxx)$. Note that as with output sides of growth blocks, the external sides of seed block border tiles have block type symbols compatible with the tiles on an input side of a growth block. The three other TM simulations consist of tile types that are rotated versions of the north TM simulation shown. The halting tile types propagate one of the members of the tuple on which the TM halts, analogous to the border tile types. The bond types of TM tile types have a symbol from $\mathcal{D}$ which indicates which simulation they belong to (omitted above).*

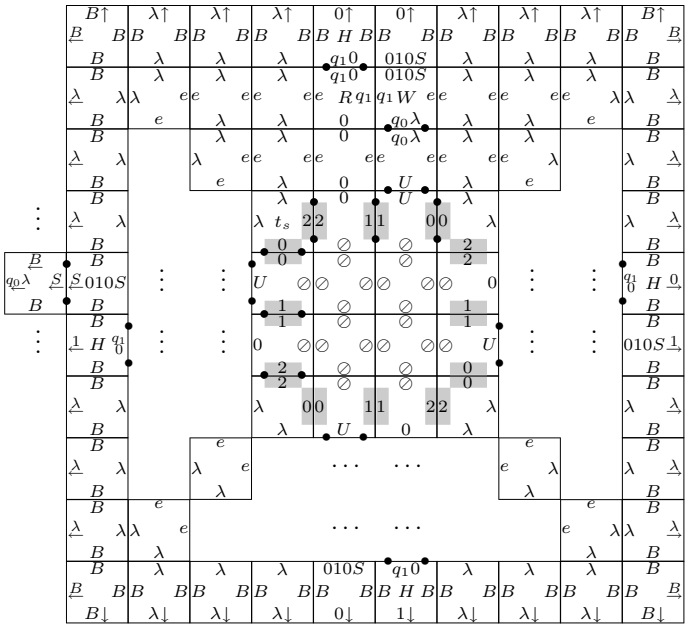


FIG. 5.6. *A simple seed block without unpacking showing the north TM simulation and the selective transmission of information through the borders. As shown, only the west side is a propagating output side; the other three sides are terminating output sides. All bond types in which a block type symbol is omitted have the block type symbol "☆." We assume that in bond types above, a single symbol $x \in \{0, 1, S, \lambda\}$ is the same as the tuplet $(xxxx)$. The natural assembly sequence to consider is growing the seed frame first band and then adding tiles row by row from the center (where a new row is started by the strength-2 bond).*

Our way of encoding information is based on Adleman et al. [3] but modified to work in a $\tau = 2$ tile system (with strength function ranging over $\{0, 1, 2\}$) and to fit our construction in its geometry. We express a length-n binary string using a concatenation of $\lceil n/k \rceil$ binary substrings of length k, padding with 0's if necessary.[10] We choose k such that it is the least integer satisfying $\frac{n}{\log n} \leq 2^k$. Clearly, $2^k < \frac{2n}{\log n}$. See Figure 5.7 for the tile types used in the unpacking for the north TM simulation and for a simple unpacking example (which for the sake of illustration uses $k = 4$).

Let us consider the number of tile types used to encode and unpack the n-bit input string for a single TM simulation (i.e., north). There are $2\lceil n/k \rceil \leq 2\lceil \frac{n}{\log \frac{n}{\log n}} \rceil = 2\lceil \frac{n}{\log n - \log\log n} \rceil$ unique tile types in each seed row. This implies that there exists a constant h such that $2\lceil n/k \rceil \leq \frac{3n}{\log n} + h$ for all n. We need at most $2^k + 2^{k-1} + \cdots + 4 < 2^{k+1}$ "extract bit" tile types and $2^{k-1} + 2^{k-2} + \cdots + 4 < 2^k$ "copy remainder" tile types. To initiate the unpacking of new substrings we need 2^k tile types. To keep on copying substrings that are not yet unpacked we need $2(2^k)$ tile types. The quantity of the other tile types is independent of n, k. Thus, in total, to unpack the n-bit input string for a single TM simulation we need no more than $\frac{3n}{\log n} + h + 2^{k+1} + 2^k + 2^k + 2(2^k) \leq 15\frac{n}{\log n} + O(1)$ tile types. Since there are 4 TM simulations in the seed block, we need $60\frac{n}{\log n} + O(1)$ tile types to encode and unpack the n-bit input string.

If the seed block requires only one propagating output side, then a reduced construction using fewer tile types can be used: only one side of the seed frame is specified, and only one direction of unpacking tiles are used. A constant number of additional tile types are used to fill out the remaining three sides of the square. These additional tile types must perform two functions. First, they must properly extend the diagonal on either side of the unpacking and TM simulation regions. In the absence of the other three unpacking and TM simulation processes, this requires adding strength-2 bonds that allow the diagonal to grow to the next layer. Second, the rest of the square must be filled in to the correct size. This can be accomplished by adding tiles that extend one diagonal to the other side of the seed frame (using the same logic as a construction in [11]). Altogether, a seed block with only one propagating output side requires only $15\frac{n}{\log n} + O(1)$ tile types. We will see in the next section that this is sufficient for growing any shape.

5.4. Programming blocks and the value of the scaling factor *c*. In order for our tile system to produce some assembly whose shape is $\tilde{S}$, instructions encoded in p must guide the construction of the blocks by deciding on which side of which block a new block begins to grow and what is encoded on the edge of each block. For our purposes, we take $p = p_{sb}\langle s \rangle$ (i.e., p_{sb} takes s as input), where s is a program that outputs the list of locations in the shape S. p_{sb} runs s to obtain this list and plans out a spanning tree t over these locations (it can just do a depth-first search) starting from some arbitrarily chosen location that will correspond to the seed block.[11] The information passed along the arrows in Figure 5.1(b) is $p_{gb}\langle t, (i, j) \rangle$, which is the concatenation of a program p_{gb} to be executed within each growth block, and an encoding of the tree t and the location (i, j) of the block into which the arrow is heading. When executed, $p_{gb}\langle t, (i, j) \rangle$ evaluates to a 3-tuple encoding of $p_{gb}\langle t, D(i, j) \rangle$ together with symbol "S" for each propagating output side D. Thus, each growth

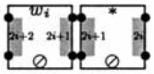
[10] We can assume that our universal TM U treats trailing 0's just as λ's.

[11] We can opt to always choose a leaf, in which case the seed block requires only one propagating output side. In this case the multiplicative factor a_1 is $15 + \varepsilon$, although the tile set used will depend upon the direction of growth from the leaf.

a) Unpacking tile types for the north side of the seed frame:

We use n/k coding tiles in the input row, each encoding a binary substring (w_i) of length k. These tiles are interspersed with buffer tiles holding the symbol "$*$". $\forall 0 \geq i \geq k/n - 1$:

The last tile of the seed row has symbol "U" which indicates the end of the input string.

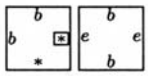


To initiate the unpacking of new substrings: $\forall x \in \{0,1\}^{k-1}$, $b \in \{0,1\}$:

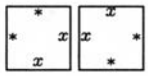
The following "extract bit" tile types perform the actual unpacking: $\forall j \in \{1,\ldots,k-1\}, \forall x \in \{0,1\}^j, b \in \{0,1\}$:

The following "copy remainder" tile types pass the remaining bits to the next extraction: $\forall j \in \{2,\ldots,k-1\}, \forall x \in \{0,1\}^j$:

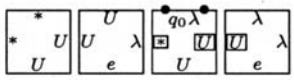
To copy a single bit in the last step of the unpacking of a substring and after unpacking every bit: $b \in \{0,1\}$:



These tile types keep on copying substrings that are not yet being unpacked: $\forall x \in \{0,1\}^k$:



Finally, the following tile types propagate the symbol "U", which indicates the end of the input string, and initiate the TM simulation once the unpacking process finishes:



b) North unpacking example:

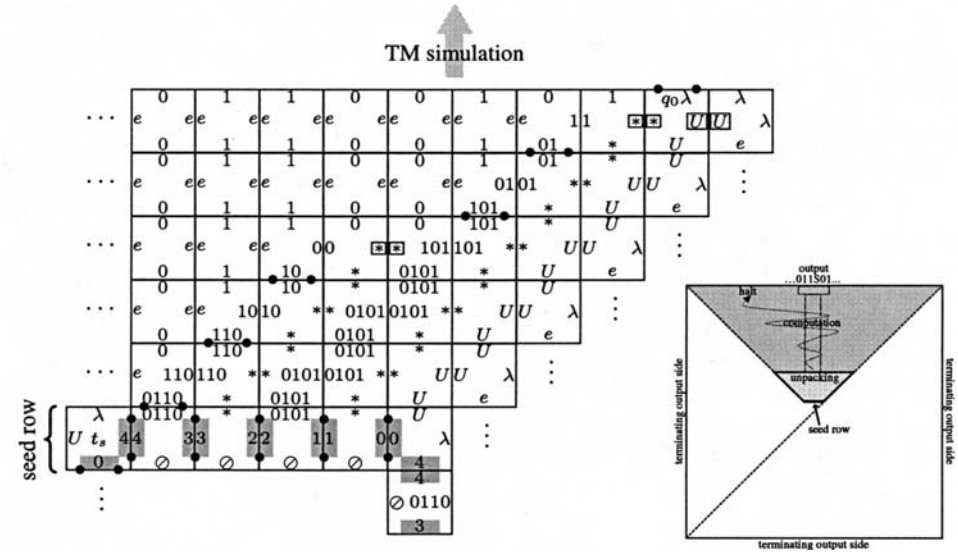


FIG. 5.7. *The unpacking for the north side of the seed frame.* (a) *The tile types used.* (b) *An example showing the unpacking of the string* 01100101 *if* $k = 4$ *for a seed block with up to four propagating output sides. Note that the unpacking process can be inserted immediately prior to the TM simulation without modifying other tile types. The inset shows the internal structure of a seed block with only one propagating output side.*

block passes $p_{gb}\langle t, D(i,j)\rangle$ to its Dth propagating output side as directed by t. Note that program p_{sb} in the seed tile must also run long enough to ensure that c is large enough that the computation in the growth blocks has enough space to finish without running into the sides of the block or into the diagonal. Nevertheless, the scaling factor c is dominated by the building of t in the seed block, as the computation in the growth blocks takes only $poly(|S|)$.[12] Since the building of t is dominated by the running time of s, we have $c = poly(time(s))$.

5.5. Uniqueness of the terminal assembly. By Theorem 2.3 it is enough to demonstrate a locally deterministic assembly sequence ending in our target terminal assembly to be assured that this terminal assembly is uniquely produced. Consider the assembly sequence $\vec{A}$ in which the assembly is constructed *block by block* such that every block is finished before the next one is started and each block is constructed by the natural assembly sequence described in the captions of Figures 5.4 and 5.6. It is enough to confirm that in this natural assembly sequence every tile addition satisfies the definition of local determinism (Definition 2.2). It is easy to confirm that every tile not adjacent to a terminal output side of a block indeed satisfies these conditions. Other than on a terminal output side of a block (and on *null* tiles) there are no *termsides*: every side is either an *inputside* or a *propside*. In our construction, each new tile binds through either a single strength-2 bond or two strength-1 bonds (thus condition 1 is satisfied since $\tau = 2$) such that no other tile type can bind through these *inputsides* (condition 2 is satisfied if the tile has no *termsides*). Note that inadvertent binding of a tile type from a different block type is prevented by the block type symbols.

Now let us consider *termsides* around the terminal output sides of blocks (Figure 5.8(a)). Here block type symbols come to the rescue again and prevent inadvertent binding. Let $t \in A$ be a tile with a *termside* (t can be *null*). We claim that $\forall t'$ s.t. $type(t') \in T$ and $pos(t') = pos(t)$, if $\Gamma^A_{termsides^{\vec{A}}(t)}(t') > 0$, then $\Gamma^A_{\mathcal{D}-propsides^{\vec{A}}(t)}(t') < \tau = 2$. In other words, if t' binds on a *termside* of t, then it cannot bind strongly enough to violate local determinism, implying we can ignore *termsides*. Figure 5.8(a) shows in dotted lines the *termsides* that could potentially be involved in bonding. These *termsides* cannot have a strength-2 bond because symbol "S" is not propagated to terminal output sides of blocks. Thus t' binding only on a single *termside* of t is not enough. Can t' bind on two *termsides* of t? To do so, it must be in a corner between two blocks, binding two terminal output sides of different blocks. But to bind in this way would require t' to bond to the block type symbol pattern[13] shown in Figure 5.8(b) (or its rotation), which none of the tile types in our tile system can do. Can t' bind on one *termside* and one *inputside* of t? Say the *termside* of t that t' binds on is the west side (Figure 5.8(c)). The tile to the west of t must be on the east terminal output side of a block, and thus it has symbol "$\rightarrow$" on its east side. So t' must have "$\rightarrow$" on the west, and depending on the type of block t is in, one of the other block type symbols as shown in Figure 5.8(c). But again none of the tile types in our tile system has the necessary block type symbol pattern.

[12] Note that fewer than n rows are necessary to unpack a string of length n (section 5.3). Since we can presume that p_{sb} reads its entire input and the universal TM needs to read the entire input program to execute it, the number of rows required for the unpacking process can be ignored with respect to the asymptotics of the scaling factor c.

[13] The block type symbol pattern of a tile type consists of the block type symbols among its four bond types. For instance, the tile type 

has block type symbol pattern 

. If two bond types do not have matching block type symbols, then obviously they cannot bind.

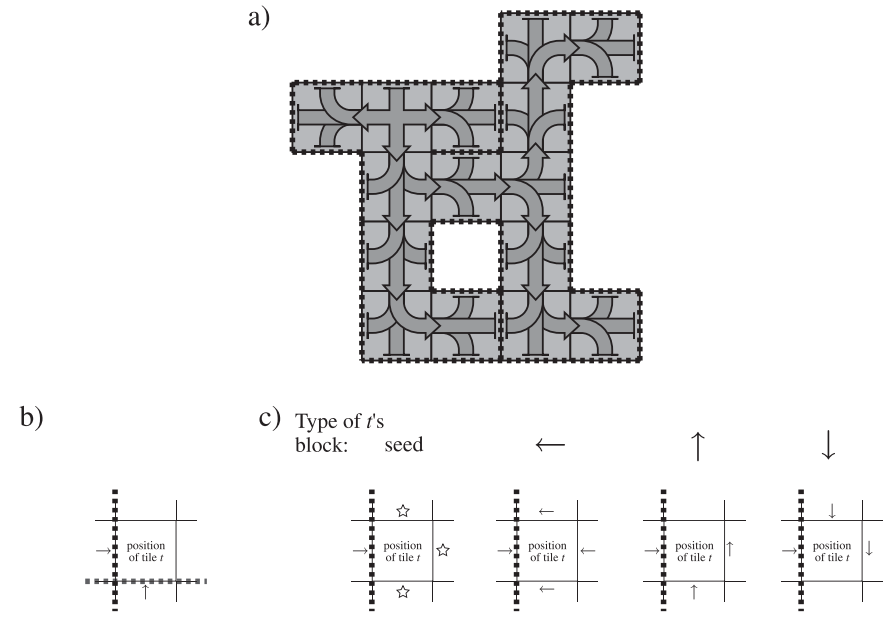


FIG. 5.8. (a) *The target terminal assembly with the dotted lines indicating the edges that have termsides with nonnull bonds.* (b) *The block type symbols of adjacent tiles on two termsides of t (west and south in this case).* (c) *The block type symbols of adjacent tiles on a termside (west side in this case) and an inputside of t. If t is in the seed block or ← growth block, then the north, east, and south sides may be the inputsides. If t is in a ↑ block, then the east and south sides may be the inputsides. If t is in a ↓ block, then the north and east sides may be the inputsides.*

6. Generalizations of shape complexity. In this work we have established both upper and lower bounds relating the descriptional complexity of a shape to the number of tile types needed to self-assemble the shape within the standard tile assembly model. The relationship is dependent upon a particular definition of shape that ignores its size. Disregarding scale in self-assembly appears to play a role similar to that of disregarding time in theories of computability and decidability. Those theories earned their universal standing by being shown to be identical for all "reasonable" models of computation. To what extent do our results depend on the particular model of self-assembly? Can one define a complexity theory for families of shapes in which the absolute scale is the critical resource being measured? In this section we discuss the generality and limitations of our result.

6.1. Optimizing the main result (section 4). Since the Kolmogorov complexity of a string depends on the universal TM chosen, the complexity community adopted a notion of additive equivalence, where additive constants are ignored. However, Theorem 4.1 includes multiplicative constants as well, which are not customarily discounted. It might be possible to use a more clever method of unpacking (section 5.2) and a seed block construction that reduces the multiplicative constant a_1 of Theorem 4.1. Correspondingly, there might be a more efficient way to encode any tile system than that described in the proof of the theorem, and thereby increase a_0.

Recall that s is the program for U producing the target coordinated shape S as a list of locations. For cases where our results are of interest, the scaling factor $c = poly(time(s))$ is extremely large since $|S|$ is presumably enormous and s must output every location in S. The program s' that, given (i, j), outputs 0/1, indicating whether S contains that location, may run much faster than s for large shapes. Can

our construction be adapted to use s' in each block rather than s in the seed block to obtain smaller scale? The problem with doing this directly is that the scale of the blocks, which sets the maximum allowed running time of computation in each block, must be set in the seed block. As a result, there must be some computable time bound on s' that is given to the seed block.

For any particular shape, there must be a range of achievable parameters: the number of tile types and the scaling factor. We know that we can obtain scaling factor 1 by using a unique tile type for each location. On the other extreme is our block construction which allows us to obtain an asymptotically optimal number of tile types at the expense of an enormous scaling factor. Presumably there is a gradual tradeoff between the number of tile types and the scale that can be achieved by a range of tile systems. The characterization of this tradeoff is a topic for future study.

In this vein, an important open problem remains of determining lower bounds on the scales of shapes produced by tile systems with an asymptotically optimal number of tile types. As an initial result of this kind, consider the following proof that an arbitrarily large scaling factor may need to be used if we stick to asymptotically optimal tile systems. Consider the coordinated shape that is a rectangle of width m and height 1. Clearly, it is an instance of the following shape $\tilde{S}$: a long, thin rectangle that is m times as long as it is high. According to Aggarwal et al. [4], the number of tile types required to self-assemble a long, thin rectangle that is n tiles long and k tiles high is $\Omega(\frac{n^{1/k}}{k})$. This implies that to produce any coordinated instance of $\tilde{S}$ at scale c requires $|T| = \Omega(\frac{(mc)^{1/c}}{c})$ tile types. Now we can define what an asymptotically optimal tile system means for us by choosing a_1, b_1 and requiring that the number of tile types $|T|$ satisfies $|T| \log |T| \leq a_1 K(\tilde{S}) + b_1$. Since $K(\tilde{S}) = O(\log m)$, it follows through simple algebra that no matter what a_1, b_1 are, for large enough m, the scaling factor c needs to get arbitrarily large to avoid a contradiction.

6.2. Strength functions. In most previous works on self-assembly, as in this work, strength functions are restricted by the following properties: (1) the effect that one tile has on another is equal to the effect that the other has on the first (i.e., g is *symmetric*: $g(\sigma, \sigma') = g(\sigma', \sigma)$); (2) the lack of an interaction is normalized to zero (i.e., $g(\sigma, null) = 0$); (3) there are no "adverse" interactions counteracting other interactions (i.e., g is *nonnegative*); (4) only sides with matching bond types interact (i.e., g is *diagonal*: $g(\sigma, \sigma') = 0$ if $\sigma \neq \sigma'$).

Properties 1 and 2 seem natural enough. Our results are independent of property 3 because the encoding used for the lower bound of Theorem 4.1 is valid for strength functions taking on negative values. Property 4, which reflects the roots of the tile assembly model in the Wang tiling model, is essential for the quantitative relationship expressed in Theorem 4.1: recent work by Aggarwal et al. [4] shows that permitting nondiagonal strength functions allows information to be encoded more compactly. Indeed, if property 4 is relaxed, then replacing our unpacking process with the method of encoding used in that work and using the lower bound of Aggarwal et al. leads to the following form of Theorem 4.1: assuming the maximum threshold τ is bounded by a constant, there exist constants a_0, b_0, a_1, b_1 such that for any shape $\tilde{S}$,

$$a_0 K(\tilde{S}) + b_0 \leq \left(K_{sa}^{nd}(\tilde{S})\right)^2 \leq a_1 K(\tilde{S}) + b_1,$$

where K_{sa}^{nd} is the tile-complexity when nondiagonal strength functions are allowed. It is an open question whether the constant bound on τ can be relaxed.

6.3. Wang tiling versus self-assembly of shapes. Suppose one is solely concerned with the existence of a configuration in which all sides match, and not with the process of assembly. This is the view of classical tiling theory [7]. Since finite tile sets can enforce uncomputable tilings of the plane [8, 16], one might expect greater computational power when the existence, rather than production, of a tiling is used to specify shapes. In this section we develop the notion of shapes in the Wang tile model [20] and show that results almost identical to the tile assembly model hold. One conclusion of this analysis is that making a shape "practically constructible" (i.e., in the sense of the tile assembly model) does not necessitate an increase in tile-complexity.

We translate the classic notion of the origin-restricted Wang tiling problem[14] as follows. An *(origin-restricted) Wang tiling system* is a pair (T, t_s), where T is a set of tile types and t_s is a *seed tile* with $type(t_s) \in T$. A configuration A is a valid tiling if all sides match and it contains the seed tile. Formally, A is a *valid tiling* if $\forall (i,j) \in \mathbb{Z}^2, D \in \mathcal{D}$, (1) $type(A(i,j)) \in T$, (2) $t_s \in A$, and (3) $bond_D(A(i,j)) = bond_{D^{-1}}(A(D(i,j)))$.

Since valid tilings are infinite objects, how can they define finite coordinated shapes? For tile sets containing the *empty* tile type, we can define shapes analogously to the tile assembly model. However, we cannot simply define the coordinated shape of a valid tiling to be the set of locations of nonempty tiles. For one thing, the set of nonempty tiles can be disconnected, unlike in self-assembly where any produced assembly is a single connected component. So we take the coordinated shape S_A of a valid tiling A to be the smallest region of nonempty tiles containing t_s that can be extended to infinity by *empty* tiles. Formally, S_A is the coordinated shape of the smallest subset of A that is a valid tiling containing t_s. If S_A is finite, then it is the *coordinated shape of valid tiling* A.[15] Shape $\tilde{S}$ is the *shape of a valid tiling* A if $S_A \in \tilde{S}$.

Produced assemblies of a tile system (T, t_s, g, τ) are not necessarily valid tilings of Wang tiling system (T, t_s) because the tile assembly model allows mismatching sides. Further, valid tilings of (T, t_s) are not necessarily produced assemblies of (T, t_s, g, τ). Even if one considers only valid tilings that are connected components, there might not be any sequence of legal tile additions that assembles these configurations. Nonetheless, if a tile system uniquely produces a valid tiling A, then all valid tilings of the corresponding Wang tile system agree with A and have the same coordinated shape as A.

LEMMA 6.1. *If* $empty \in T$ *and the tile system* $\mathbf{T} = (T, t_s, g, \tau)$ *uniquely produces assembly* A *such that* A *is a valid tiling of the Wang tiling system* (T, t_s)*, then for all valid tilings* A'*, it holds that* (1) $\forall (i,j) \in \mathbb{Z}^2$*,* $type(A(i,j)) \neq empty \Rightarrow A'(i,j) = A'(i,j)$*, and* (2) $S_{A'} = S_A$.

Proof. Consider an assembly sequence $\vec{A}$ of $\mathbf{T}$ ending in assembly A and let A' be a valid tiling of (T, t_s). Suppose t_n is the first tile added in this sequence such that $t' = A'(pos(t_n)) \neq t_n$. Since A' is a valid tiling, t' must match on all sides, including $inputsides^{\vec{A}}(t_n)$. But this implies that two different tiles can be added in the same location in $\vec{A}$, which means that A is not uniquely produced. This implies part (1) of the lemma. Now, to be a valid tiling, all exposed sides of assembly A must be null. Thus if A' and A agree on all places where A is nonempty, then $S_{A'} = S_A$, and part (2) of the lemma follows. □

[14] The *unrestricted* Wang tile model does not have a seed tile [20, 5, 18].

[15] S_A can be finite only if $empty \in T$ because otherwise no configuration containing an *empty* tile can be a valid tiling.

Define the *tile-complexity* K_{wt} of a shape $\tilde{S}$ in the origin-restricted Wang tiling model as the minimal number of tile types in a Wang tiling system with the property that a valid tiling exists and there is a coordinated shape $S \in \tilde{S}$ such that for every valid tiling A, $S_A = S$.

THEOREM 6.1. *There exist constants a_0, b_0, a_1, b_1 such that for any shape $\tilde{S}$,*

$$a_0 K(\tilde{S}) + b_0 \leq K_{wt}(\tilde{S}) \log K_{wt}(\tilde{S}) \leq a_1 K(\tilde{S}) + b_1.$$

Proof sketch. The left inequality follows in a manner similar to the proof of Theorem 4.1. Suppose every valid tiling of our Wang tile system has coordinated shape S. Any Wang tiling system of n tile types can be represented using $O(n \log n)$ bits. Making use of this information as input, we can use a constant-size program to find, through exhaustive search, the smallest region containing t_s surrounded by *null* bond types in some valid tiling. Thus, $O(n \log n)$ bits are enough to compute an instance of $\tilde{S}$. To prove the right inequality, our original block construction almost works, except that there are mismatches between a terminal output side of a block and the abutting terminal output side of the adjacent block or the surrounding *empty* tiles (i.e., along the dotted lines in Figure 5.8(a)). Consequently, the original construction does not yield a valid tiling. Nonetheless, a minor variant of our construction overcomes this problem. Instead of relying on mismatching bond type symbols to prevent inadvertent binding to terminal output sides of blocks, we can add an explicit capping layer that covers the terminal output sides with *null* bond types but propagates information through propagating output sides. This way, the terminal output sides of blocks are covered by *null* bond types and match the terminal output sides of the adjacent block and *empty* tiles. These modifications can be made preserving local determinism, which, by Lemma 6.1, establishes that the coordinated shape of any valid tiling is an instance of $\tilde{S}$. □

There may still be differences in the computational power between Wang tilings and self-assembly processes. For example, consider the smallest Wang tiling system and the smallest self-assembly tile system that produce instances of $\tilde{S}$. The instance produced by the Wang tiling system might be much smaller than the instance produced by self-assembly. Likewise, there might be *coordinated* shapes that can be produced with significantly fewer tile types by a Wang tiling system than by a self-assembly system.

Keep in mind that the definition we use for saying when a Wang tiling system produces a shape was chosen as a natural parallel to the definition used for self-assembly, but alternative definitions may highlight other interesting phenomena specific to Wang tilings. For example, one might partition tiles into two subsets, "solution" and "substance" tiles, and declare shapes to be connected components of substance tiles within valid tilings. In such tilings—reminiscent of "vicinal water" in chemistry—the solution potentially can have a significant (even computational) influence that restricts possible shapes of the substance, and hence the size of produced shapes need not be so large as to contain the full computation required to specify the shape.

6.4. Sets of shapes. Any coordinated shape S can be trivially produced by a self-assembly tile system or by a Wang tiling of $|S|$ tile types. Interesting behavior occurs only when the number of tile types is somehow restricted and the system is forced to perform some nontrivial computation to produce a shape. Previously in this paper, we restricted the number of tile types in the sense that we ask what is the minimal number of tile types that can produce a given shape. We saw that ignoring scale in this setting allows for an elegant theory. In the following two sections the restriction on the number of tile types is provided by the infinity of shapes they must

be able to produce. Here we will see as well that ignoring scale allows for an elegant theory.

Adleman [2] asks, "What are the 'assemblable [sic] shapes' - (analogous to what are the 'computable functions')?" While this is still an open question for coordinated shapes, our definition of a shape ignoring scale and translation leads to an elegant answer. A set of binary strings $\tilde{L}$ is a language of shapes if it consists of (standard binary) encodings of lists of locations that are coordinated shapes in some set of shapes: $\tilde{L} = \{\langle S \rangle \text{ s.t. } S \in \tilde{S} \text{ and } \tilde{S} \in R\}$ for some set of shapes R. Note that every instance of every shape in R is in this language. The language of shapes $\tilde{L}$ is recursively enumerable if there exists a TM that halts upon receiving $\langle S \rangle \in \tilde{L}$ and does not halt otherwise. We say a tile system $\mathbf{T}$ produces the language of shapes $\tilde{L}$ if $\tilde{L} = \{\langle S \rangle \text{ s.t. } S \in \tilde{S_A} \text{ for some } A \in Term(\mathbf{T})\}$. We may want $\tilde{L}$ to be *uniquely produced* in the sense that the $A \in Term(\mathbf{T})$ is unique for each shape. Further, to prevent infinite spurious growth we may also require $\mathbf{T}$ to satisfy the following *noncancerous* property: $\forall B \in Prod(\mathbf{T}), \exists A \in Term(\mathbf{T}) \text{ s.t. } B \rightarrow_{\mathbf{T}}^{*} A$. The following lemma is valid whether or not these restrictions are made.

LEMMA 6.2. *A language of shapes $\tilde{L}$ is recursively enumerable if and only if it is (uniquely) produced by a (noncancerous) tile system.*

Proof sketch. First of all, for any tile system $\mathbf{T}$ we can make a TM that, given a coordinated shape S as a list of locations, starts simulating all possible assembly sequences of $\mathbf{T}$ and halts if and only if it finds a terminal assembly that has shape $\tilde{S}$. Therefore, if $\tilde{L}$ is produced by a tile system, $\tilde{L}$ is recursively enumerable. In the other direction, if $\tilde{L}$ is recursively enumerable, then there is a program p that, given n, outputs the nth shape from $\tilde{L}$ (in some order) without repetitions. Our programmable block construction can be modified to execute a nondeterministic universal TM in the seed block by having multiple possible state transitions. We make a program that nondeterministically guesses n, feeds it to p, and proceeds to build the returned shape. Note that since every computation path terminates, this tile system is noncancerous, and since p enumerates without repetitions, the language of shapes is uniquely produced. □

Note that the above lemma does not hold for languages of coordinated shapes, defined analogously. Many simple recursively enumerable languages of coordinated shapes cannot be produced by any tile system. For example, consider the language of equilateral width-1 crosses centered at $(0, 0)$. No tile system produces this language. Scale-equivalence is crucial because it allows arbitrary amounts of information to be passed between different parts of a shape; otherwise, the amount of information is limited by the width of a shape.

The same lemma can be attained for the Wang tiling model in an analogous manner using the construction from section 6.3. Let us say a Wang tiling system (T, t_s) produces the language of shapes $\tilde{L}$ if $\tilde{L} = \{\langle S \rangle \text{ s.t. } S \in \tilde{S_A} \text{ for some valid tiling } A \text{ of } (T, t_s)\}$. Analogously to tile systems, we may require the *unique production* property that there is exactly one such A for each shape. Likewise, corresponding to the noncancerous property of tile systems, we may also require the tiling system to have the *noncancerous* property that every valid tiling has a coordinated shape (i.e., is finite). Again, the following lemma is true whether or not these restrictions are made.

LEMMA 6.3. *A language of shapes $\tilde{L}$ is recursively enumerable if and only if it is (uniquely) produced by a (noncancerous) Wang tiling system.*

6.5. Scale complexity of shape functions. Expanding upon the notion of a shape being the output of a universal computation process as mentioned in the in-

troduction, let us consider tile systems effectively computing a function from binary strings to shapes. The universal "programmable block" constructor presented in section 5 may be taken as an example of such a tile set if the full seed block is considered as an initial seed assembly rather than as part of the tile set per se. In this case, the remaining tile set is of constant size and will construct an arbitrary algorithmic shape when presented with a seed assembly containing the relevant program. The universal constructor tile set's efficiency, then, can be measured in terms of the scale of the produced shape. Similarly, other "programmable" tile sets may produce a limited set of shapes, but potentially with greater efficiency. (Such tile sets can be thought to produce a language of shapes (section 6.4) such that the choice of the produced shape can be deterministically specified.) For tile systems outputting shapes in this manner, we can show that the total number of tiles (not tile *types*) in the produced shape is closely connected to the time complexity of the corresponding function from binary strings to shapes in terms of TMs. The equivalent connection can be made between nondeterministic TMs and the size of valid tilings in the Wang tiling model.

Let f be a function from binary strings to shapes. We say that a TM M computes this function if for all x, $f(x) = \tilde{S} \Leftrightarrow \exists S \in \tilde{S}$ s.t. $M(x) = \langle S \rangle$. The standard notion of time complexity applies: $f \in TIME_{TM}(t(n))$ if there is a TM computing it running in time bounded by $t(n)$, where n is the size of the input. In section 5.2.2 we saw how binary input can be provided to a tile system via a seed frame wherein all four sides of a square present the bit string. Let us apply this convention here.[16] Extending the notion of the seed in self-assembly to the entire seed frame and using this as the input for a computation [17], we say a tile system computes f if the following holds: [starting with the seed frame encoding x the tile system uniquely produces an assembly of shape $\tilde{S}$] if and only if $f(x) = \tilde{S}$. We say that $f \in TILES_{SA}(t(n))$ if there is a tile system computing it and the size of coordinated shapes produced (in terms of the number of nonempty locations) for inputs of size n is upper bounded by $t(n)$. Similar definitions can be made for nondeterministic TMs (NDTMs) and Wang tiling systems. We say that an NDTM N computes f if the following holds: [every computation path of N on input x ending in an accept state (as opposed to a reject state) outputs $\langle S \rangle$ for some $S \in \tilde{S}$] if and only if $f(x) = \tilde{S}$. For NDTMs, $f \in TIME_{NDTM}(t(n))$ if there is an NDTM computing f such that every computation path halts after $t(n)$ steps. Extending the notion of the seed for Wang tilings to the entire seed frame as well, we say a Wang tiling system computes f if all valid tilings containing the seed frame have a coordinated shape and this coordinated shape is the same for all such valid tilings, and it is an instance of the shape $f(x)$. We say that $f \in TILES_{WT}(t(n))$ if there is a tiling system computing it and the size of coordinated shapes produced for inputs of size n is upper bounded by $t(n)$.

THEOREM 6.4.

(a) *If* $f \in TILES_{SA}(t(n))$, *then* $f \in TIME_{TM}(O(t(n)^4))$.

(b) *If* $f \in TIME_{TM}(t(n))$, *then* $f \in TILES_{SA}(O(t(n)^3))$.

(c) *If* $f \in TILES_{WT}(t(n))$, *then* $f \in TIME_{NDTM}(O(t(n)^4))$.

(d) *If* $f \in TIME_{NDTM}(t(n))$, *then* $f \in TILES_{WT}(O(t(n)^3))$.

Proof sketch. (a) Let $\mathbf{T}$ be a tile system computing f such that the total number of tiles used on an input of size n is $t(n)$. A TM with a two-dimensional tape can simulate the self-assembly process of $\mathbf{T}$ with an input of size n in $O(t(n)^2)$ time: for each of the $t(n)$ tile additions, it needs to search $O(t(n))$ locations for the next addition. This

[16] Any other similar method would do. For the purposes of this section, it does not matter whether we use the one-bit-per-tile encoding or the encoding requiring unpacking (section 5.3).

two-dimensional TM can be simulated by a regular TM with a quadratic slowdown.[17]

(b) Let M be a deterministic TM that computes f and runs in time $t(n)$. Instead of simulating a universal TM in the block construction, we simulate a TM M' which runs M on input x encoded in the seed frame and acts as program p_{sb} in section 5.4. Then the scale of each block is $O(t(n))$, which implies that each block consists of $O(t(n)^2)$ tiles. Now the total number of blocks cannot be more than the running time of M since M outputs every location that corresponds to a block. Thus the total number of tiles is $O(t(n)^3)$.

(c) An argument similar to (a) applies to the Wang tiling system with the following exception. A Wang tiling system can simulate an NDTM and still be able to output a unique shape. The tiling system can be designed such that if a reject state is reached, the tiling cannot be a valid tiling. For example, the tile representing the reject state can have a bond type that no other tile matches. Thus all valid tilings correspond to accepting computations.

(d) Simulation of Wang tiling systems can, in turn, be done by an NDTM as follows. Suppose every valid tiling of our Wang tile system has coordinated shape S. The simulating NDTM acts in a manner similar to that of the TM simulating self-assembly above, except that every time two or more different tiles can be added in the same location, it nondeterministically chooses one. If the NDTM finds a region containing the seed frame surrounded by *null* bond types, it outputs the shape of the smallest such region and enters an accept state. Otherwise, at some point no compatible tile can be added, and the NDTM enters a reject state. The running time of accepting computations is $O(t(n)^2)$ via the same argument as for (b). □

If, as is widely believed, NDTMs can compute some functions in polynomial time that require exponential time on a TM, then it follows that there exist functions from binary strings to shapes that can be computed much more efficiently by Wang tiling systems than by self-assembly, where efficiency is defined in terms of the size of the coordinated shape produced.

The above relationship between $TIME$ and $TILES$ may not be the tightest possible. As an alternative approach, very small-scale shapes can be created as Wang tilings by using an NDTM that recognizes tuples (i, j, x), rather than one that generates the full shape. This will often yield a compact construction. As a simple example, this approach can be applied to generating circles with radius x at scale $O(n^2)$, where $n = O(\log x)$. It remains an open question how efficiently circles can be generated by self-assembly.

6.6. Other uses of programmable growth. The programmable block construction is a general way of guiding the large-scale growth of the self-assembly process and may have applications beyond those explored so far. For instance, instead of constructing shapes, the block construction can be used to simulate other tile systems in a scaled manner using fewer tile types. It is easy to reprogram it to simulate, using few tile types, a large deterministic $\tau = 1$ tile system for which a short algorithmic description of the tile set exists. We expect that a slightly extended version of the

[17] The rectangular region of the two-dimensional tape previously visited by the two-dimensional head (the arena) is represented row by row on a one-dimensional tape separated by special markers. The current position of the two-dimensional head is also represented by a special marker. If the arena is $l \times m$, a single move of the two-dimensional machines which does not escape the current arena requires at most $O(m^2)$ steps, while a move that escapes it in the worst case requires an extra $O(ml^2)$ steps to increase the arena size. We have $m, l = O(t(n))$, and the number of times the arena has to be expanded is at most $O(t(n))$.

block construction can also be used to provide compact tile sets that simulate other $\tau = 2$ tile systems that have short algorithmic descriptions.

To self-assemble a circuit, it may be that the shape of the produced complex is not the correct notion. Rather one may consider finite patterns, where each location in a shape is "colored" (e.g., resistor, transistor, wire). Further, assemblies that can grow arbitrarily large may be related to infinite patterns. What is the natural way to define the self-assembly complexity of such patterns? Do our results (section 4) still hold?

Appendix A. Local determinism guarantees unique production: Proof of Theorem 2.3.

LEMMA A.1. *If $\vec{A}$ is a locally deterministic assembly sequence of $\mathbf{T}$, then for every assembly sequence $\vec{A}'$ of $\mathbf{T}$ and for every tile $t' = t'_n$ added in $\vec{A}'$, the following conditions hold, where $t = A(pos(t'))$:*

(i) $inputsides^{\vec{A}'}(t') = inputsides^{\vec{A}}(t)$.

(ii) $t' = t$.

Proof. Suppose $t' = t'_n$ is the first tile added that fails to satisfy one of the above conditions. Consider any $D \in inputsides^{\vec{A}'}(t')$. Tile $t_D = A'(D(pos(t')))$ must have been added before t' in $\vec{A}'$ and so $D^{-1} \notin inputsides^{\vec{A}'}(t_D) = inputsides^{\vec{A}}(t_D)$. This implies $D \notin propsides^{\vec{A}}(t)$ and thus,

$$inputsides^{\vec{A}'}(t') \cap propsides^{\vec{A}}(t) = \emptyset. \tag{A.1}$$

Now, $\forall D, \Gamma_D^{A'_n}(t') \leq \Gamma_D^A(t')$ because A'_n has no more tiles than A and except at $pos(t)$ they all agree. Equation (A.1) implies

$$\Gamma^A_{inputsides^{\vec{A}'}(t')}(t') \leq \Gamma^A_{\mathcal{D}-propsides^{\vec{A}}(t)}(t').$$

Therefore,

$$\Gamma^{A'_n}_{inputsides^{\vec{A}'}(t')}(t') \leq \Gamma^A_{\mathcal{D}-propsides^{\vec{A}}(t)}(t').$$

So by property (2) of Definition 2.2, no tile of type $\neq type(t)$ could have been sufficiently bound here by $inputsides^{\vec{A}'}(t')$ and thus $t' = t$. Therefore, t' cannot fail the second condition (ii).

Now, suppose t' fails the first condition (i). Because of property (1) of Definition 2.2, this can happen only if $\exists D \in inputsides^{\vec{A}'}(t') - inputsides^{\vec{A}}(t')$. Since $D \notin inputsides^{\vec{A}}(t')$, t_D must have been added after t' in $\vec{A}$. So since t_D binds t', $D^{-1} \in inputsides^{\vec{A}}(t_D)$, and so $D \in propsides^{\vec{A}}(t)$. But by (A.1) this is impossible. Thus we conclude $A' \subseteq A$. ∎

Lemma A.1 directly implies that if there exists a locally deterministic assembly sequence $\vec{A}$ of $\mathbf{T}$, then $\forall A' \in Prod(\mathbf{T}), A' \subseteq A$. Theorem 2.3 immediately follows: if there exists a locally deterministic assembly sequence $\vec{A}$ of $\mathbf{T}$, then $\mathbf{T}$ uniquely produces A.

Since local determinism is a property of the *inputsides* classification of tiles in a terminal assembly, Lemma A.1 also implies the following corollary.

COROLLARY A.2. *If there exists a locally deterministic assembly sequence $\vec{A}$ of $\mathbf{T}$, then every assembly sequence ending in A is locally deterministic.*

Appendix B. Scale-equivalence and "≅" are equivalence relations. Translation-equivalence is clearly an equivalence relation. Let us write $S_0 \overset{tr}{=} S_1$ if the two coordinated shapes are translation-equivalent.

LEMMA B.1. *If $S = S_0^d$ and $S_0 = S_m^k$, then $S = S_m^{dk}$.*

Proof. $S(i,j) = S_0(\lfloor i/d \rfloor, \lfloor j/d \rfloor) = S_m(\lfloor \lfloor i/d \rfloor / k \rfloor, \lfloor \lfloor j/d \rfloor / k \rfloor) = S_m(\lfloor i/dk \rfloor, \lfloor j/dk \rfloor)$. □

LEMMA B.2. *If $S_0 \overset{tr}{=} S_1$, then $S_0^d \overset{tr}{=} S_1^d$.*

Proof. $S_0^d(i,j) = S_0(\lfloor i/d \rfloor, \lfloor j/d \rfloor) = S_1(\lfloor i/d \rfloor + \Delta i, \lfloor j/d \rfloor + \Delta j) = S_1(\lfloor \frac{i+d\Delta i}{d} \rfloor, \lfloor \frac{j+d\Delta j}{d} \rfloor) = S_1^d(i + d\Delta i, j + d\Delta j)$. □

To show that scale-equivalence is an equivalence relation, the only nontrivial property is transitivity. Suppose $S_0^c = S_1^d$ and $S_1^{d'} = S_2^{c'}$ for some $c, c', d, d' \in \mathbb{Z}^+$. $(S_1^d)^{d'} = (S_1^{d'})^d = S_1^{d'd}$ by Lemma B.1. Thus, $S_1^{d'd} = (S_0^c)^{d'} = (S_2^{c'})^d$, and by Lemma B.1, $S_0^{cd'} = S_2^{c'd}$.

To show that "≅" is an equivalence relation, again only transitivity is nontrivial. Suppose $S_0 \cong S_1$ and $S_1 \cong S_2$. In other words, $S_0^c \overset{tr}{=} S_1^d$ and $S_1^{d'} \overset{tr}{=} S_2^{c'}$ for some $c, c', d, d' \in \mathbb{Z}^+$. By Lemma B.2, $(S_0^c)^{d'} \overset{tr}{=} (S_1^d)^{d'}$ and $(S_1^{d'})^d \overset{tr}{=} (S_2^{c'})^d$. Then by Lemma B.1, $S_0^{cd'} \overset{tr}{=} S_1^{d'd}$ and $S_1^{d'd} \overset{tr}{=} S_2^{c'd}$, which implies $S_0^{cd'} \overset{tr}{=} S_2^{c'd}$ by the transitivity of translation-equivalence. In other words, $S_0 \cong S_2$.

Acknowledgments. We thank Len Adleman, members of his group, Ashish Goel, and Paul Rothemund for fruitful discussions and suggestions. We also thank Rebecca Schulman and David Zhang for useful and entertaining conversations about descriptional complexity of tile systems, and an anonymous reviewer for a very careful reading of this paper and helpful comments.

REFERENCES

[1] L. ADLEMAN, Q. CHENG, A. GOEL, M.-D. HUANG, D. KEMPE, P. MOISSET DE ESPANES, AND P. W. K. ROTHEMUND, *Combinatorial optimization problems in self-assembly*, in Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing, ACM, New York, 2002, pp. 23–32.

[2] L. M. ADLEMAN, *Toward a Mathematical Theory of Self-Assembly* (extended abstract), Technical report, University of Southern California, Los Angeles, 1999.

[3] L. ADLEMAN, Q. CHENG, A. GOEL, AND M.-D. HUANG, *Running time and program size for self-assembled squares*, in Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing, ACM, New York, 2001, pp. 740–748.

[4] G. AGGARWAL, Q. CHENG, M. H. GOLDWASSER, M.-Y. KAO, P. MOISSET DE ESPANES, AND R. T. SCHWELLER, *Complexities for generalized models of self-assembly*, SIAM J. Comput., 34 (2005), pp. 1493–1515.

[5] R. BERGER, *The undecidability of the domino problem*, Mem. Amer. Math. Soc., 66 (1966).

[6] M. COOK, P. W. K. ROTHEMUND, AND E. WINFREE, *Self-assembled circuit patterns*, in DNA Computing, Lecture Notes in Comput. Sci. 2943, Springer-Verlag, Berlin, 2004, pp. 91–107.

[7] B. GRUNBAUM AND G. SHEPHARD, *Tilings and Patterns*, W. H. Freeman, New York, 1986.

[8] W. HANF, *Nonrecursive tilings of the plane* I, J. Symbolic Logic, 39 (1974), pp. 283–285.

[9] P. W. K. ROTHEMUND, *Theory and Experiments in Algorithmic Self-Assembly*, Ph.D. thesis, University of Southern California, Los Angeles, 2001.

[10] P. W. K. ROTHEMUND, N. PAPADAKIS, AND E. WINFREE, *Algorithmic self-assembly of DNA Sierpinski triangles*, PLoS Biology, 2 (2004), e424.

[11] P. W. K. ROTHEMUND AND E. WINFREE, *The program-size complexity of self-assembled squares*, in Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, ACM, New York, 2000, pp. 459–468.

[12] T. H. LABEAN, H. YAN, J. KOPATSCH, F. LIU, E. WINFREE, J. H. REIF, AND N. C. SEEMAN, *Construction, analysis, ligation, and self-assembly of DNA triple crossover complexes*, J. Am. Chem. Soc., 122 (2000), pp. 1848–1860.

[13] M. Li and P. Vitanyi, *An Introduction to Kolmogorov Complexity and Its Applications*, 2nd ed., Springer, New York, 1997.
[14] C. Mao, T. H. LaBean, J. H. Reif, and N. C. Seeman, *Logical computation using algorithmic self-assembly of DNA triple-crossover molecules*, Nature, 407 (2000), pp. 493–496.
[15] C. Mao, W. Sun, and N. C. Seeman, *Designed two-dimensional DNA Holliday junction arrays visualized by atomic force microscopy*, J. Am. Chem. Soc., 121 (1999), pp. 5437–5443.
[16] D. Myers, *Nonrecursive tilings of the plane* II, J. Symbolic Logic, 39 (1974), pp. 286–294.
[17] J. H. Reif, *Local parallel biomolecular computation*, in DNA-Based Computers III, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 48, AMS, Providence, RI, 1999, pp. 217–254.
[18] R. M. Robinson, *Undecidability and nonperiodicity of tilings of the plane*, Invent. Math., 12 (1971), pp. 177–209.
[19] J. von Neumann, *The Theory of Self-Reproducing Automata*, A. W. Burks, ed., University of Illinois Press, Urbana, IL, 1966.
[20] H. Wang, *Proving theorems by pattern recognition.* II, Bell Sys. Tech. J., 40 (1961), pp. 1–41.
[21] E. Winfree, *Simulations of Computing by Self-Assembly*, Caltech CS Technical report, 1998.22, California Institute of Technology, Pasadena, CA.
[22] E. Winfree, *Algorithmic Self-Assembly of DNA*, Ph.D. thesis, California Institute of Technology, Pasadena, CA, 1998.
[23] E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman, *Design and self-assembly of two dimensional DNA crystals*, Nature, 394 (1998), pp. 539–544.

SIAM J. COMPUT.
Vol. 36, No. 6, pp. 1570–1599

FIRST-ORDER LANGUAGES EXPRESSING CONSTRUCTIBLE SPATIAL DATABASE QUERIES*

BART KUIJPERS†, GABRIEL KUPER‡, JAN PAREDAENS§, AND LUC VANDEURZEN†

Abstract. The research presented in this paper is situated in the framework of constraint databases introduced by Kanellakis, Kuper, and Revesz in their seminal paper of 1990, specifically, the language with real polynomial constraints ($\mathsf{FO} + \mathsf{poly}$). For reasons of efficiency, this model is implemented with only *linear* polynomial constraints, but this limitation to linear polynomial constraints has severe implications on the expressive power of the query language. In particular, when used for modeling spatial data, important queries that involve Euclidean distance are not expressible. The aim of this paper is to identify a class of two-dimensional constraint databases and a query language within the constraint model that go beyond the linear model and allow the expression of queries concerning distance. We seek inspiration in the Euclidean constructions, i.e., constructions by ruler and compass. We first present a programming language that captures exactly the first-order ruler-and-compass constructions that are expressible in a first-order language with real polynomial constraints. If this language is extended with a **while** operator, we obtain a language that is complete for all ruler-and-compass constructions in the plane. We then transform this language in a natural way into a query language on finite point databases, but this language turns out to have the same expressive power as $\mathsf{FO} + \mathsf{poly}$ and is therefore too powerful for our purposes. We then consider a *safe* fragment of this language and use this to construct a query language that allows the expression of Euclidean distance without having the full power of $\mathsf{FO} + \mathsf{poly}$.

Key words. constraint databases, real algebraic geometry, first-order logic, query languages, ruler-and-compass constructions

AMS subject classifications. 03B70, 14P10, 51M04, 57R05, 68P15

DOI. 10.1137/S0097539702407199

1. Introduction and motivation.

Kanellakis, Kuper, and Revesz [28, 29] (see also [30]) introduced the framework of *constraint databases* which provides a rather general model for spatial databases [32]. Spatial database systems [1, 7, 10, 21, 22, 37] are concerned with the representation and manipulation of data that have a geometrical or topological interpretation. In the context of the constraint model, a spatial database, although conceptually viewed as a possibly infinite set of points in the real space, is represented as a finite union of systems of polynomial equations and inequalities. For example, the spatial database consisting of the set of points on the northern hemisphere together with the points on the equator of the unit sphere in the three-dimensional space $\mathbf{R}^3$ can be represented by the formula $x^2+y^2+z^2 = 1 \wedge z \geq 0$. The set $\{(x, y) \mid (y-x^2)(x^2-y+1/2) > 0\}$ of points in the real plane lying strictly above the parabola $y = x^2$ and strictly below the parabola $y = x^2 + 1/2$ is an example of a two-dimensional database in the constraint model. These are called *semi-algebraic* sets [6].

Several query languages on databases using the constraint model have been stud-

*Received by the editors May 7, 2002; accepted for publication (in revised form) August 28, 2006; published electronically February 20, 2007.
http://www.siam.org/journals/sicomp/36-6/40719.html
†Theoretical Computer Science Group, Hasselt University and Transnational University of Limburg, B-3590 Diepenbeek, Belgium (bart.kuijpers@uhasselt.be, luc.vandeurzen@groept.be).
‡Dipartimento Informatica e Telecommunicazioni, Università di Trento, I-38050 Povo, Trento, Italy (kuper@acm.org).
§Department of Mathematics and Computer Science, University of Antwerp, Middelheimlaan 1, B-2020 Antwerp, Belgium (jan.paredaens@ua.ac.be).

ied. One such query language is obtained by extending the relational calculus with polynomial inequalities [32]. This language is usually referred to as $\mathsf{FO} + \mathsf{poly}$. The query deciding whether the two-dimensional database S is a straight line, for instance, can be expressed in this language by the sentence

$$\exists a \exists b \exists c \big(\neg(a = 0 \wedge b = 0) \wedge \forall x \forall y (S(x, y) \leftrightarrow ax + by + c = 0) \big) .$$

Although variables in such expressions range over the real numbers, queries expressed in this calculus can still be computed effectively. In particular, the closure property holds: Any $\mathsf{FO} + \mathsf{poly}$ query, when evaluated on a spatial database in the constraint model, yields a spatial database in the constraint model. This follows immediately from Tarski's quantifier-elimination procedure for the first-order theory of real closed fields [38].

However, Tarski's quantifier-elimination procedure is computationally very expensive. Since the 1970s various more efficient algorithms have been proposed, including Cylindrical Algebraic Decomposition [8, 9], which are still unsuitable for practice. The best known algorithms were proposed in the 1990s [4, 34], and, by the use of alternative data structures [14], in the best case, quantifier elimination is exponential in the number of quantifier blocks. (A recent textbook on this matter is [36], and for a discussion on the influence of data structures we refer the reader to [25]). Due to this complexity it seems to be infeasible for real spatial database applications to rely on quantifier elimination (we discuss this further below). In existing implementations of the constraint model, such as the DEDALE system [15, 16, 17], the constraints are restricted to *linear* polynomial constraints, and the sets definable in this restricted model are called *semi-linear*. It is argued that linear polynomial constraints provide a sufficiently general framework for spatial database applications [17, 40]. Indeed, in one of the main application domains, geographical information systems, linear approximations are used to model geometrical objects (for an overview of this field since the early '90s, see [1, 7, 10, 21, 22, 37]).

When we extend the relational calculus with linear polynomial inequalities, we obtain an effective language which has the same closure property as above but this time with respect to linear databases. We refer to this language as $\mathsf{FO} + \mathsf{lin}$, and therefore an $\mathsf{FO} + \mathsf{lin}$ query evaluated on a linear constraint database yields a linear constraint database.

We return to the complexity of query evaluation by quantifier elimination. Although for both $\mathsf{FO} + \mathsf{poly}$ and $\mathsf{FO} + \mathsf{lin}$ the cost of quantifier elimination grows exponentially with the number of blocks of quantifiers to be eliminated, an argument in favor of the language $\mathsf{FO} + \mathsf{lin}$ is that there exists a conceptually "easier" way (Fourier's method [27, 31]) to eliminate quantifiers for this language which makes an effective implementation feasible [15, 16, 17]. This algorithm has the same asymptotic complexity as the quantifier-elimination procedure for $\mathsf{FO} + \mathsf{poly}$, though there is a slight gain in data complexity: Grumbach and Su have shown that the data complexity for $\mathsf{FO} + \mathsf{lin}$ is NC^1, while it is NC for $\mathsf{FO} + \mathsf{poly}$ (for certain restrictions of $\mathsf{FO} + \mathsf{lin}$, namely ℓ-bounded instances, an AC^0 bound is obtained) [19]. From the practical point of view, a more significant advantage of the linear model is the existence of numerous efficient algorithms for geometrical operations [33].

There are, however, a number of serious disadvantages to the restriction to linear polynomial constraints, related to the limited expressive power of the query language $\mathsf{FO} + \mathsf{lin}$. The expressive power of the language $\mathsf{FO} + \mathsf{lin}$ has been extensively studied (see, e.g., [2, 3, 18, 20, 40, 41] and references therein). Among the limitations of

$\mathsf{FO} + \mathsf{lin}$, one of the most important is that the language is incapable of expressing queries that involve Euclidean distance, betweenness, and collinearity. A query like "Return all cities in Belgium that are further than 100 km away from Brussels" is, however, a query that is of importance in spatial database applications. The goal of this paper is to overcome these limitations of $\mathsf{FO} + \mathsf{lin}$ for the special case of two-dimensional spatial databases.

We note that languages whose expressive power on semi-linear databases is strictly between that of $\mathsf{FO} + \mathsf{lin}$ and $\mathsf{FO} + \mathsf{poly}$ have already been studied. Vandeurzen, Gyssens, and Van Gucht [40, 41] have shown that, even though $\mathsf{FO} + \mathsf{lin}$ extended with a primitive for collinearity yields a language with the complete expressive power of $\mathsf{FO} + \mathsf{poly}$, a "careful" extension with a collinearity operator yields a language whose expressive power is strictly between that of $\mathsf{FO} + \mathsf{lin}$ and that of $\mathsf{FO} + \mathsf{poly}$ on semi-linear databases. However, even this extension does not allow the expression of queries involving distance.

In this paper, we define a new query language, $\mathsf{SafeEuQL}^{\uparrow}$, and a class of two-dimensional constraint databases on which this language is closed, called *semi-circular* databases. The language $\mathsf{SafeEuQL}^{\uparrow}$ allows the expression of queries concerning distance, betweenness, and collinearity. The class of semi-circular databases obviously is a strict superclass of the class of linear databases, since $\mathsf{SafeEuQL}^{\uparrow}$ allows for the definition of data by means of distance. Semi-circular databases are describable by means of polynomial equalities, inequalities that involve linear polynomials, and polynomials that define circles. The language $\mathsf{SafeEuQL}^{\uparrow}$ is strictly more powerful on linear databases than $\mathsf{FO} + \mathsf{lin}$, and on semi-circular databases is strictly less powerful than $\mathsf{FO} + \mathsf{poly}$.

To define this language, we have turned, for inspiration, to the *Euclidean constructions*, i.e., the constructions by ruler and compass that we know from high-school geometry. These constructions were first described in the 4th century B.C. by Euclid of Alexandria in the 13 books of his *Elements* [24]. Of the 465 propositions to be found in these volumes only 60 are concerned with ruler-and-compass constructions. Most of these constructions belong to the mathematical folklore and are known to all of us. "Construct the perpendicular from a given point on a given line" or "construct a regular pentagon" are well-known examples. Since the 19th century, we also know that a certain number of constructions are *not* performable by ruler and compass, e.g., the trisection of an arbitrary angle or the squaring of the circle. For a 20th century description of these constructions and the main results concerning them, we refer the reader to [26].

An alternative to considering languages based on ruler and compass constructions would be to use a constraint language based on the field of the constructible real numbers. It follows from a result by Ziegler that this theory is undecidable [42], however. Ziegler showed, among other results, that any finitely axiomatizable subtheory of the reals with addition, multiplication, and order is undecidable (as conjectured by Tarski).

We define and study in the current paper three languages for ruler-and-compass constructions.

First, we define a programming language that describes Euclidean constructions. We will refer to this procedural language as EuPL (short for Euclidean Programming Language). Engeler [11, 12] studied a similar programming language in the '60s, but his language contains a while-loop and therefore goes beyond first-order logic-based languages. His language also differs from ours in that EuPL also contains a choice

statement. This statement corresponds to choosing arbitrary points, which satisfy some conditions, in the plane, something that is often done in constructions with ruler and compass on paper. We claim that EuPL captures exactly the planar geometrical constructions, i.e., the first-order expressible ruler-and-compass constructions. We show that the choice statement, at least for deterministic programs, can be omitted. We also prove a number of useful decidability properties of EuPL programs: that equivalence and satisfiability of EuPL programs are decidable, and that it is decidable whether a program is deterministic.

We then transform the programming language EuPL into a query language for finite point databases, called EuQL (short for Euclidean Query Language). It turns out that this calculus can express nonconstructible queries—in fact, we show that EuQL has the same expressive power on finite point databases as FO + poly. It is therefore too powerful for our purposes.

We then study a *safe* fragment of EuQL, in which all queries are constructible. In particular, a SafeEuQL query returns constructible finite point relations from given finite point relations.

SafeEuQL is the key ingredient in our query language SafeEuQL$^{\uparrow}$ for semi-circular databases. Since SafeEuQL works on finite point databases, we interpret these queries to work on intensional representations of semi-circular databases. We then give FO + poly-definable mappings from databases to their representation and back. Using these mappings, we can "lift" SafeEuQL to a query language on semi-circular databases. This "lifting" technique has also been used by Benedikt and Libkin [5] and Gyssens, Vandeurzen, and Van Gucht [23].

We then compare the expressive power of SafeEuQL$^{\uparrow}$ with the expressive power of FO + poly on semi-circular databases, and show that on semi-linear databases FO + poly is more expressive than SafeEuQL$^{\uparrow}$. Finally, we compare the expressive power of SafeEuQL$^{\uparrow}$ and FO + lin on semi-linear databases.

Overview of the query languages. The following scheme gives an overview of the different languages. A horizontal arrow indicates that a language is closed on the given class of databases. A subscheme of the form

$$\begin{array}{ccc} B & \xrightarrow{\mathcal{L}_1} & B \\ & \subsetneq \uparrow & \\ A & \xrightarrow{\mathcal{L}_2} & A \end{array}$$

means that on databases in the class A, the language $\mathcal{L}_1$, mapping databases in class A to databases in class A, is more expressive than the language $\mathcal{L}_2$, mapping databases in class B to databases in class B (where B is a larger class of databases than A). The top part of Figure 1, for instance, expresses that SafeEuQL$^{\uparrow}$ is strictly more expressive than FO + poly on semi-circular databases.

Organization of the paper. In the next section, we define FO + poly and FO + lin. In section 3, we introduce the class of semi-circular databases and describe a complete and lossless representation of them by means of finite point databases. We devote the next three sections to the study of the three languages for ruler-and-compass constructions: EuPL, EuQL, and SafeEuQL. The query language for semi-circular databases is given in section 7, where we show that it is closed and compare its expressive power with that of FO + lin on semi-linear databases and FO + poly on semi-circular databases.

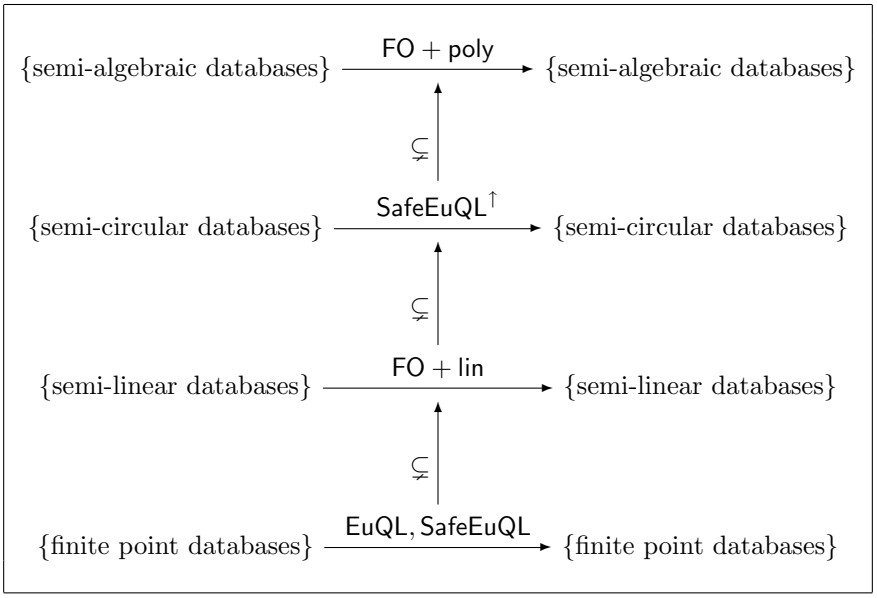


FIG. 1. *Comparison of the expressive power of the different languages.*

2. Constraint-based database models. In this section, we provide the necessary background for the polynomial and linear constraint database models and formally define two query languages, FO + poly and FO + lin, for the polynomial and the linear database model, respectively. Since the linear database model is a submodel of the polynomial database model, we start with the latter. We denote the set of the real numbers by $\mathbf{R}$.

2.1. The polynomial database model. A *polynomial formula* is a well-formed first-order logic formula over the theory of the real numbers, i.e., over $(+, \times, <, 0, 1)$. In other words, a polynomial formula is built with the logical connectives $\wedge$, $\vee$, and $\neg$ and the quantifiers $\exists$ and $\forall$ from atomic formulas of the form $p(x_1, \ldots, x_n) > 0$, where $p(x_1, \ldots, x_n)$ is a polynomial with real algebraic coefficients and real variables $x_1, \ldots, x_n$.

Every polynomial formula $\varphi(x_1, \ldots, x_n)$ with n free variables $x_1, \ldots, x_n$ defines a point set

$$\{(x_1, \ldots, x_n) \in \mathbf{R}^n \mid \varphi(x_1, \ldots, x_n)\}$$

in the n-dimensional Euclidean space $\mathbf{R}^n$ in the standard manner. Point sets defined by a polynomial formula are called *semi-algebraic sets*. We shall also refer to them as *semi-algebraic relations*, since they can be seen as n-ary relations over the real numbers.

We remark that, by the quantifier-elimination theorem of Tarski [38], it is always possible to represent a semi-algebraic set by a quantifier-free formula. The same theorem also guarantees the decidability of the equivalence of two polynomial formulas.

A *polynomial database* is a finite set of *semi-algebraic relations*, and a query in the polynomial database model is a computable mapping from m-tuples of semi-algebraic relations to a semi-algebraic relation.

The most natural query language for the polynomial data model is the relational

calculus augmented with polynomial equalities and inequalities, i.e., the first-order language which contains as atomic formulas polynomial inequalities and formulas of the form $R_i(y_1, \ldots, y_n)$, where R_i $(i = 1, \ldots, m)$ are semi-algebraic relation names for the input parameters of the query, and $y_1, \ldots, y_n$ are real variables. In the literature, this query language is commonly referred to as $\mathsf{FO} + \mathsf{poly}$ [30].

Example 2.1. The $\mathsf{FO} + \mathsf{poly}$ formula

$$R(x, y) \wedge \forall\varepsilon(\varepsilon \leq 0 \vee \exists v \exists w(\neg R(v, w) \wedge (x - v)^2 + (y - w)^2 < \varepsilon))$$

has x and y as free variables. For a given binary semi-algebraic relation R, it computes the set of points with coordinates (x, y) that belong to the intersection of R and its topological border.

Tarski's quantifier-elimination procedure ensures that every $\mathsf{FO} + \mathsf{poly}$ query is effectively computable and yields a polynomial database as result [28, 29] (this property is commonly referred to as "closure").

2.2. The linear database model. Polynomial formulas built from atomic formulas that contain only linear polynomials with real algebraic coefficients are called *linear formulas*. Point sets defined by linear formulas are called *semi-linear sets* or *semi-linear relations*.

We remark that there is also a quantifier-elimination property for the linear model: Any linear formula that contains quantifiers can be converted to an equivalent quantifier-free linear formula. There is a conceptually easy algorithm, usually referred to as Fourier's method, for eliminating quantifiers in the linear model (this method is described in [27, 31]).

A *linear database* is a finite set of *semi-linear relations*. As in the polynomial model, queries in the linear model are defined as mappings from m-tuples of semi-linear relations to a semi-linear relation. A very appealing query language for the semi-linear data model, called $\mathsf{FO} + \mathsf{lin}$, is obtained by restricting the polynomial formulas in $\mathsf{FO} + \mathsf{poly}$ to contain only linear polynomials. Using algebraic computation techniques for the elimination of variables, one can see that the result of every $\mathsf{FO} + \mathsf{lin}$ query is a semi-linear relation [27, 31, 30].

Example 2.2. The $\mathsf{FO} + \mathsf{lin}$ formula

$$R(x, y) \wedge \forall\varepsilon(\varepsilon \leq 0 \vee \exists v \exists w(\neg R(v, w) \wedge x - \varepsilon < v < x + \varepsilon \wedge y - \varepsilon < w < y + \varepsilon))$$

has two free variables: x and y. For a given binary semi-linear relation R, it computes the set of points with coordinates (x, y) that belong to the intersection of R and its topological border. In fact, this formula is equivalent to the one in Example 2.1, even though it makes use of a different metric to compute the topological border. It should be clear that not every $\mathsf{FO} + \mathsf{poly}$ formula has an equivalent $\mathsf{FO} + \mathsf{lin}$ formula.

3. Semi-circular relations. We now describe a class of planar relations in the constraint model that can be described by linear polynomials and those quadratic polynomials that describe circles, and then describe an encoding of these relations as finite relations of points. This encoding will be *complete*, meaning that every such relation has an encoding, and *lossless*, meaning that the original relation can be recovered from the encoding (even in $\mathsf{FO} + \mathsf{poly}$).

DEFINITION 3.1. *We call a subset of* $\mathbf{R}^2$ *a* semi-circular set *or* semi-circular relation *if and only if it can be defined as a Boolean combination of sets of the form*

$$\{(x, y) \mid ax + by + c\ \theta\ 0\},$$

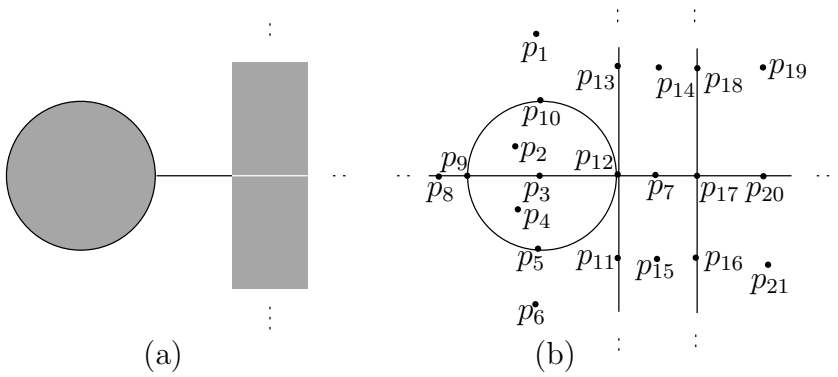


FIG. 2. *A semi-circular relation* (a) *and its carrier* (b).

where a, b, and c are real algebraic numbers with $a \neq 0$ or $b \neq 0$, and θ is either $\geq$ or $>$; or

$$\{(x, y) \mid (x - a)^2 + (y - b)^2 \; \theta \; c^2\} \, ,$$

where a, b, and c are real algebraic numbers with $c \neq 0$, and θ is either $\geq$ or $>$.

As far as planar figures are concerned, the class of semi-circular relations clearly contains the class of semi-linear relations.

Example 3.1. Figure 2(a) shows an example of a semi-circular relation. It is the set

$$\{(x, y) \mid x^2 + y^2 \leq 1 \vee (y = 0 \wedge 1 \leq x < 2) \vee (x > 2 \wedge \neg y = 0)\} \, .$$

Given such a semi-circular database, we consider all of the sets of the form $\{(x, y) \mid p(x, y) = 0\}$ for each polynomial $p(x, y)$ that occurs in the definition of the semi-circular relation.

For the semi-circular relation of Figure 2(a), these sets are shown in Figure 2(b) and are defined by the equations $x^2 + y^2 - 1 = 0$, $y = 0$, $x - 1 = 0$, and $x - 2 = 0$. We refer to these lines and circles as *a carrier* of the semi-circular relation (or as *the carrier* of a particular representation of the semi-circular relation). The carrier in Figure 2(b) partitions the plane $\mathbf{R}^2$ into 21 classes, each of which belongs either entirely to the semi-circular relation or to its complement. In general, these partition classes can be disconnected. We then pick a representative of each of these classes, illustrated by points $p_1, \ldots, p_{21}$ in Figure 2(b). We can then represent a semi-circular relation R by a finite point database[1] that consists of three relations, L, P, and C, as follows:

1. L contains, for each line in the carrier of R, a pair of its points;
2. C contains, for each circle in the carrier of R, its center and one of its points;
3. P contains a representative of each class in the partition induced by the carrier of R that belongs to R.

The sets L and C are binary relations of points in the plane, while the set P is a unary relation of points.

[1]These points can be represented explicitly by their real algebraic coordinates, or implicitly by a real polynomial formula. Equality of such points can therefore be decided by Tarski's theorem [38].

We refer to such a finite representation of a semi-circular relation as an *intensional LPC-representation*. Clearly, a semi-circular relation can have more than one intensional LPC-representation, for example, because there may be more than one constraint formula describing it.

For the semi-circular database of Figure 2(a) we can have the following finite representation: L consists of the tuples (p_7, p_8), (p_{11}, p_{12}), and (p_{16}, p_{17}); C consists of the single tuple (p_3, p_5); and P consists of the points p_2, p_3, p_4, p_5, p_7, p_9, p_{10}, p_{12}, p_{19}, and p_{21}.

The complete plane $\mathbf{R}^2$ can be represented by $L = \emptyset$, $C = \emptyset$, and $P = \{p\}$ for any point p in $\mathbf{R}^2$.

We remark that an intensional LPC-representation of a semi-circular relation is lossless in the sense that the semi-circular relation can be reconstructed from the representation. In section 7, we show how to compute in $\mathsf{FO} + \mathsf{poly}$ an LPC-representation of a semi-circular relation and how to reconstruct, also in $\mathsf{FO} + \mathsf{poly}$, the semi-circular relation from its LPC-representation.

As an immediate consequence of the existence of quantifier-elimination algorithms for the real closed field, we get the following property.

PROPOSITION 3.1. *It is decidable whether two LPC-representations of semi-circular relations represent the same semi-circular relation.*

4. The language EuPL. We now define our first programming language, EuPL, for expressing Euclidean constructions. This language is modeled after the language of Engeler [11], with two key differences. The language of Engeler uses iteration, but we are interested only in first-order database query languages. We therefore first explore the consequences of defining a Euclidean programming language without iteration. An additional feature of EuPL is that it includes a nondeterministic choice operator. We decided to include this operator as it is used frequently in the Euclidean constructions that we want to model, but we shall show that this choice operator is redundant, since, under appropriate assumptions, it can be simulated by other operations.

EuPL has one basic type $\langle \text{var} \rangle$ which ranges over points in the Euclidean plane. We use p, $q, \ldots$ to denote variables. There is no basic notion of lines and circles, since lines are represented by pairs of points (p_1, p_2) and circles by triples (p_1, p_2, p_3), where (p_1, p_2) represents the line through the points p_1 and p_2 (assuming p_1 and p_2 are distinct) and (p_1, p_2, p_3) (assuming p_2 and p_3 are distinct) represents the circle with center p_1 and radius equal to $d(p_2, p_3)$, the distance between p_2 and p_3.[2]

Our language corresponds to one view of Euclidean constructions, as it is known that all ruler-and-compass constructions can be carried out on lines and circles that are represented by points (see [13]). The main reason that we have chosen to use a point representation in EuPL is to make the language similar to the database languages in the following sections, where such an encoding really is necessary. As far as EuPL is concerned, however, we could have defined a similar language with lines and circles as primitive notions—all of the results in the current section, apart from Theorem 4.2, would still hold.

The basic predicates in EuPL are as follows:

1. **defined**(p),
2. $p_1 = p_2$,
3. (i) p_1 **is on line** (p_2, p_3),

[2] We could also represent circles by pairs (p_1, p_2), where p_1 is the center and p_2 a point on the circle, or in other ways. Our choice is arbitrary, but tends to make actual constructions simpler.

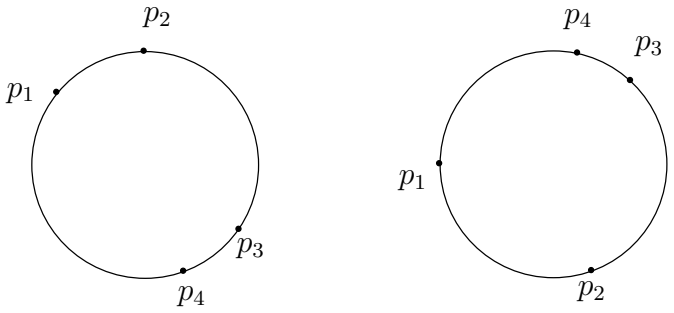


FIG. 3. *The two orientations for* **c-order**(p_1, p_2, p_3, p_4).

(ii) p_1 **is on circle** (p_2, p_3, p_4),
(iii) p_1 **is in circle** (p_2, p_3, p_4),
(iv) p_1 **is on the same side as** p_2 **of line** (p_3, p_4),

4. (i) **l-order**(p_1, p_2, p_3),
(ii) **c-order**(p_1, p_2, p_3, p_4).

The first condition means that the variable p represents a point. Such a test is needed, as a variable may be undefined if it is the result of an intersection of two disjoint objects, such as parallel lines.

Given our encoding of lines and circles, the meaning of the predicates in 3(a–d) should be clear. For example, p_1 **is in circle** (p_2, p_3, p_4) means that p_1 is in the circle with center p_2 and radius $d(p_3, p_4)$. The predicates in 4(a), 4(b) are order relations. The predicate **l-order**(p_1, p_2, p_3) (line-order) means that p_1, p_2, and p_3 are on the same line, and that p_2 is between p_1 and p_3. **c-order**(p_1, p_2, p_3, p_4) (circle-order) means that p_1, p_2, p_3, and p_4 are all on the same circle, in this order, in either the clockwise or the counterclockwise direction (see Figure 3). Note that whenever pairs (respectively, triples) of points do not define lines (respectively, circles), the corresponding predicates are false.

The basic operations in EuPL to compute intersections of objects correspond to the combinations line/line, line/circle, and circle/circle.

1. $q :=$ **l-l-crossing**(p_1, p_2, p_3, p_4);
2. $q_1, q_2 :=$ **l-c-crossing**$(p_1, p_2, p_3, p_4, p_5)$;
3. $q_1, q_2 :=$ **c-c-crossing**$(p_1, p_2, p_3, p_4, p_5, p_6)$.

The semantics of these operators in most cases should be clear, except that the choice of which intersection point to assign to q_1 and which to q_2 is arbitrary.[3] In the case of the intersection of two parallel lines (identical or not), q is undefined, and similarly for the intersection of two disjoint circles, or for the intersection of a disjoint circle and line. In the case of a line tangent to a circle or two circles that meet in a single point, q_1 and q_2 will be identical.

The choice operator, whose syntax is

$$\textbf{choose } p \textbf{ such that } \langle \text{condition} \rangle ,$$

assigns to p, in a nondeterministic manner, a point p that satisfies the given condition. When $\langle$condition$\rangle$ is unsatisfiable, p is undefined.

[3]This actually introduces an additional nondeterminism into the language. This can be handled by straightforward modifications of the proofs that follow and will be ignored from now on.

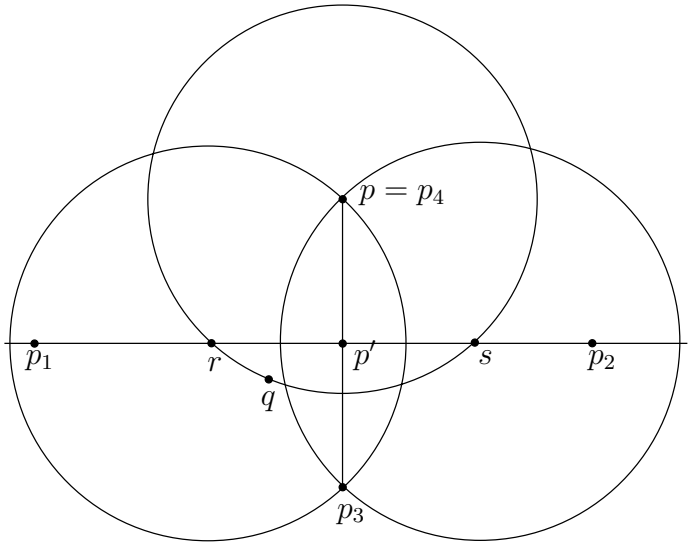


FIG. 4. *Construction of the perpendicular.*

The language also has a conditional statement **if** C **then** S_1 **else** S_2 with the usual semantics. A formal specification of the language appears in the appendix. The basic notion is that of a *multifunction* with n input variables and m output variables, representing points in the plane. Each multifunction is defined by a sequence of assignment statements and conditional statements, without looping, and its result is returned by a **result** statement.

We illustrate the language by several examples, showing how to express several Euclidean constructions in EuPL.

Example 4.1. Given a line (p_1, p_2) and a point p not on the line, construct the perpendicular (p, p') to the given line from p (see Figure 4).

multifunction perp$(p, p_1, p_2) = (p')$;
begin
choose q **such that not**(q **is on the same side as** p **of line** (p_1, p_2));
  $r, s :=$ **l-c-crossing**(p_1, p_2, p, p, q);
  $p_3, p_4 :=$ **c-c-crossing**(r, r, p, s, s, p);
  $p' :=$ **l-l-crossing**(p_1, p_2, p_3, p_4);
  result(p');
end

For another example, we show how to construct an arbitrary point on an ellipse. Given collinear points a, b, p, and q, we construct, for each r between a and b, points r' and r'' "corresponding" to r such that (a) r' and r'' are on the ellipse through a and b with foci p and q and (b) as r ranges from a to b all the points on this ellipse are constructed. Note that the ellipse itself is not constructible with ruler and compass and therefore, as we shall see in Theorem 4.3, cannot be defined by an EuPL program.

Example 4.2. Given the collinear points a, b, p, and q, with $d(a, p) = d(b, q)$, p between a and q, and q between p and b, r' is constructed as follows (see Figure 5):

multifunction put-ellipse$(a, b, p, q) = (r')$;
begin
choose r **such that** **l-order**(a, r, b);

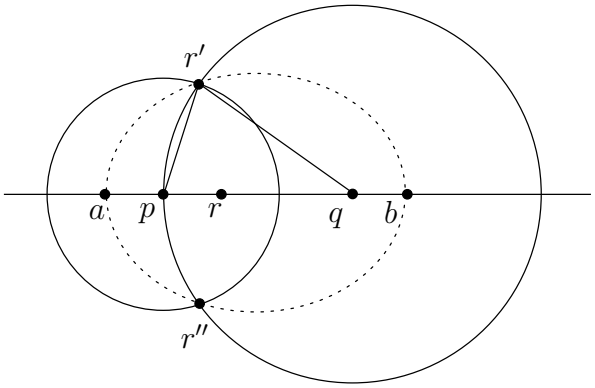


FIG. 5. *Construction of a point on an ellipse.*

$r', r'' :=$ **c-c-crossing**(p, a, r, q, b, r);
result(r');
end

The point r' is on the ellipse with foci p and q and major axis $d(a, b)$ since $d(p, r') + d(q, r') = d(a, r) + d(r, b) = d(p, a) + d(q, a) = d(p, b) + d(q, b)$.

4.1. Consequences of quantifier elimination: Representation independence. As pointed out above, EuPL is at least as powerful as a language with lines and circles as primitives. But EuPL is actually more powerful than desired. For example, if we are given a line represented by points (p_1, p_2), the EuPL program that returns p_1 would have no natural geometric interpretation.

This leads to the following problem: Given an EuPL program, does the result depend on the representation of the input lines and circles or not? The inputs and outputs of an EuPL program are just points, with no indication as to what they "really" represent. We therefore first need to impose an interpretation on these points, i.e., specify which of these points represent lines or circles.

For example, given a program with inputs $(p_1, p_2, p_3, p_4, p_5)$ and outputs (q_1, q_2), we could interpret (p_1, p_2) as a line, (p_3, p_4, p_5) as a circle, and (q_1, q_2) as a line. Other interpretations of the inputs and outputs of the program are also possible: Given a specific interpretation we shall refer to P as an *interpreted* EuPL program.

The formal definition of an interpreted program is very simple. As objects (points, lines, or circles) are represented by 1, 2, or 3 points, respectively, all we need are two equivalence relations on the input and output.

DEFINITION 4.1. *An* interpretation *of an* EuPL *program P is a pair of equivalence relations E_i and E_o on the input and output variables of P, such that each equivalence class has between* 1 *and* 3 *elements.*

Remark. Note that all of the language primitives are well defined, regardless of the interpretations of the variables. For example, q **is in circle** (p_1, p_2, p_3) is well defined even if (p_1, p_2) represents a line and (p_3, q) another line, though it is unlikely to have an intuitive result or be representation-independent. This means that we do not need to address the issue of an interpretation of the internal variables of a program.

We now address the issue of whether an EuPL program P is representation-dependent or not. It turns out that, given an interpretation of P, this is decidable.

First, we define representation dependence.

DEFINITION 4.2. *Let P be an* EuPL *program with input parameters $(p_1, \ldots, p_n)$ and output parameters $(q_1, \ldots, q_m)$ for which an interpretation is fixed. We call P* representation-independent *if for any two inputs values $(a_1, \ldots, a_n)$ and $(a'_1, \ldots, a'_n)$ that represent the same points, lines, and circles in the plane (taking into account the fixed interpretation), P returns outputs $(b_1, \ldots, b_m)$ and $(b'_1, \ldots, b'_m)$, respectively, that also represent the same points, lines, and circles in the plane (taking into account the fixed interpretation).*

The following theorem follows from the fact that the input-output transformations in EuPL can be expressed in first-order logic over the reals and from the fact that the truth of sentences in this logic can be decided (via quantifier elimination) [38].

THEOREM 4.1. *It is decidable whether the output of an interpreted* EuPL *program depends on the representation of its inputs or not.*

Proof. Let P be an EuPL multifunction, with inputs $\vec{\mathbf{p}} = (p_1, \ldots, p_n)$ and outputs $\vec{\mathbf{q}} = (q_1, \ldots, q_m)$. We shall write (p^x, p^y) for the x- and y-coordinates of a point p, and shall also write expressions such as $tp = q$ and $d(p, q) = t$ for $(tp^x = q^x \wedge tp^y = q^y)$ and $(p^x - q^x)^2 + (p^y - q^y)^2 = t^2 \wedge t \geq 0$, respectively.

We define two formulas, $\phi_P(\vec{\mathbf{p}}, \vec{\mathbf{q}}, \vec{\mathbf{r}})$ and $\psi_P(\vec{\mathbf{p}}, \vec{\mathbf{q}}, \vec{\mathbf{r}})$, over the theory of real closed fields, where $\vec{\mathbf{r}}$ is the list of internal variables of P. Intuitively, ϕ_P describes how $\vec{\mathbf{q}}$ depends on $\vec{\mathbf{p}}$, given $\vec{\mathbf{r}}$ as the results of the choice operations, while ψ_P describes the conditions that $\vec{\mathbf{r}}$ and $\vec{\mathbf{q}}$ must satisfy. In what follows, whenever ψ_S is not defined explicitly, it will be a tautology. The separation between ϕ and ψ is not really needed for the current theorem, but will be used later in Theorem 4.2. The basic idea is that for each intersection operation we add a conjunct that says when the corresponding objects are defined and have an intersection, while for each choice operation we add a conjunct that says that the condition is satisfiable. The construction of ϕ and ψ is by induction on the rules defining P. We omit some of the straightforward cases. As a first step, rename variables if needed, to ensure they are not assigned a value more than once.

1. P is the sequence of statements $S_1; S_2; \ldots; S_k$. ϕ_P is defined as $\phi_{S_1} \wedge \cdots \wedge \phi_{S_k}$ and ψ_P as $\psi_{S_1} \wedge \cdots \wedge \psi_{S_k}$.
2. Assignment statements:
   (i) S is $q := \textbf{l-l-crossing}(p_1, p_2, p_3, p_4)$. ϕ_S can be informally given as

   $$\exists! t \exists! t' (q = tp_1 + (1-t)p_2 \wedge q = t'p_3 + (1-t')p_4).$$

   Note that this formula is unsatisfiable whenever q is undefined.
   (ii) S is $q,\ q' := \textbf{l-c-crossing}(p_1, p_2, p_3, p_4, p_5)$. Let $\phi'_S(q, p_1, p_2, p_3, p_4, p_5)$ be

   $$\exists t (q = tp_1 + (1-t)p_2 \wedge d(q, p_3) = d(p_4, p_5)),$$

   which means that q is one of the intersection points. Then ϕ_S is

   $$\phi'_S(q, p_1, p_2, p_3, p_4, p_5) \wedge \phi'_S(q', p_1, p_2, p_3, p_4, p_5) \\ \wedge (q \neq q' \vee (q = q' \wedge \neg \exists q''(\phi'_S(q'', p_1, p_2, p_3, p_4, p_5) \wedge q'' \neq q)));$$

   i.e., q and q' are distinct intersection points if such exist, and both are equal to the unique intersection point if only one exists.
   (iii) S is $q,\ q' := \textbf{c-c-crossing}(p_1, p_2, p_3, p_4, p_5, p_6)$. This is similar to the previous case.

3. S is the conditional

$$\textbf{if } C \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ end}.$$

ϕ_S is

$$\phi_C \to \phi_{S_1} \wedge \phi_{\textbf{not } C} \to \phi_{S_2},$$

and ψ_S is

$$\phi_C \to \psi_{S_1} \wedge \phi_{\textbf{not } C} \to \psi_{S_2}.$$

4. S is the choice statement

$$\textbf{choose } v \textbf{ such that } C.$$

ϕ_S is a tautology, and ψ_S is equal to ϕ_C.

5. Conditionals. Most of the conditionals, such as $p_1 \textbf{ is on line } (p_2, p_3)$ are handled in a similar way to assignments. The most complicated one is when C is $\textbf{c-order}(p_1, p_2, p_3, p_4)$. Here ϕ_C first computes the center of the circle through p_1, p_2, and p_3 and tests whether p_4 is on this circle. If so, $\textbf{c-order}(p_1, p_2, p_3, p_4)$ is true when p_2 and p_4 are not on the same side of the line through p_1 and p_3.
6. C is **defined** ($\langle \text{var} \rangle$). ϕ_C is defined as the appropriate Boolean value.

Assume now that $\vec{\mathbf{p}}$ and $\vec{\mathbf{p}}'$ are two inputs to P that are equivalent with respect to the given interpretation. Let $\vec{\mathbf{q}}$ and $\vec{\mathbf{q}}'$ be the outputs of P on these inputs. From the definition of ϕ_P and the semantics of P, it follows that $\exists \vec{\mathbf{r}}(\phi_P(\vec{\mathbf{p}}, \vec{\mathbf{q}}, \vec{\mathbf{r}}) \wedge \psi_P(\vec{\mathbf{p}}, \vec{\mathbf{q}}, \vec{\mathbf{r}}))$ and $\exists \vec{\mathbf{r}}'(\phi_P(\vec{\mathbf{p}}', \vec{\mathbf{q}}', \vec{\mathbf{r}}') \wedge \psi_P(\vec{\mathbf{p}}', \vec{\mathbf{q}}', \vec{\mathbf{r}}'))$ hold.

Given the interpretation (E_i, E_o) we write formulas $\xi_i(\vec{\mathbf{p}}, \vec{\mathbf{p}}')$ and $\xi_o(\vec{\mathbf{q}}, \vec{\mathbf{q}}')$ that specify when the inputs and outputs are equivalent. For example, if $\{p_1, p_2\}$ is an equivalence class in E_i, then

$$\phi_{p_1'} \textbf{ is on line}_{(p_1, p_2)} \wedge \phi_{p_2'} \textbf{ is on line}_{(p_1, p_2)} \wedge p_1' = p_2'$$

is a conjunct in ξ_i, whereas if $\{q_1, q_2, q_3\}$ is in E_o, then

$$q_1' = q_1 \wedge d(q_2', q_3') = d(q_2, q_3)$$

is a conjunct in ξ_o.

The output of P is then independent of the representation if and only if the formula

$$\forall \vec{\mathbf{p}} \forall \vec{\mathbf{p}}' \forall \vec{\mathbf{q}} \forall \vec{\mathbf{q}}' \exists \vec{\mathbf{r}} \exists \vec{\mathbf{r}}' ((\xi_i(\vec{\mathbf{p}}, \vec{\mathbf{p}}') \wedge \phi_P(\vec{\mathbf{p}}, \vec{\mathbf{q}}, \vec{\mathbf{r}}) \wedge \psi_P(\vec{\mathbf{p}}, \vec{\mathbf{q}}, \vec{\mathbf{r}}) \\ \wedge \phi_P(\vec{\mathbf{p}}', \vec{\mathbf{q}}', \vec{\mathbf{r}}') \wedge \psi_P(\vec{\mathbf{p}}', \vec{\mathbf{q}}', \vec{\mathbf{r}}')) \to \xi_o(\vec{\mathbf{q}}, \vec{\mathbf{q}}'))$$

holds in the real numbers.

As this is a first-order formula over the theory of real closed fields, the result follows from Tarski's theorem. □

Using the formulas ϕ_P and ψ_P, given in the proof of Theorem 4.1, it is clear that we can write an $\mathsf{FO} + \mathsf{poly}$-sentence that expresses that two programs have the same input-output behavior. Therefore, we have the following corollary.

COROLLARY 4.1. *Equivalence of* EuPL *programs is decidable.*

Note that there were two different ways we could have interpreted the choice operator in the above theorem. Either, for equivalent inputs, we make the same choices, obtaining equivalent outputs, or we make different choices for both inputs, resulting in equivalent outputs. We have chosen the first approach above, but modifying the proof to use the latter approach is trivial. As discussed below, "good" programs should be choice-independent anyway, so that the distinction is not very important.

The result of the program in Example 4.1 (computation of a perpendicular) does not depend on the choice made by the choice operator. This is true for the classic Euclidean constructions as well as for all other "reasonable" EuPL programs. As with representation independence, the question whether the result of a program depends on the results of the choice operators is decidable.

The following result holds both for interpreted and for uninterpreted EuPL programs.

COROLLARY 4.2. *It is decidable whether the result of an* EuPL *program depends on the choices made by* **choose** *operators.*

COROLLARY 4.3. *It is decidable whether the result of an interpreted* EuPL *program depends on the representation of its inputs and on the results of the choice operations.*

We now show that given two fixed points and a representation- and choice-independent program P, the use of choice is actually redundant. This means that P can be converted (effectively) into an equivalent deterministic program.

We now consider a variant EuPL^{2c} of EuPL, which is just EuPL with two additional distinct constant points p_0 and p_0'.

THEOREM 4.2. *Every representation- and choice-independent* EuPL^{2c} *program P is equivalent to a program P' which does not use the* **choose** *operator.*

Proof. Let P be an EuPL^{2c} program that is representation- and choice-independent. The choice independence of P implies that the outcome of the program does not depend on the particular value of p that is chosen in any expression

$$\textbf{choose } p \textbf{ such that } \psi(p, p_1, \ldots, p_n)$$

appearing in P. Therefore, any expression **choose** p **such that** $\psi(p, p_1, \ldots, p_n)$ appearing in P may be replaced by a series of EuPL^{2c} statements, among which there is no choice statement, provided that the result is a point p satisfying $\psi(p, p_1, \ldots, p_n)$. We shall now construct such a sequence.

Note first that $\psi(p, p_1, \ldots, p_n)$ is a Boolean combination of basic choice predicates in EuPL^{2c}. We now show that there exists a formula $\psi'(p, p_1, \ldots, p_n)$, equivalent to $\psi(p, p_1, \ldots, p_n)$, such that ψ' is a Boolean combination of atomic predicates in all of which the variable p is the first variable. It will then follow that p belongs to an equivalence class of the plane determined by the lines and circles in these atomic predicates, as these predicates describe how p is located with respect to certain lines and circles determined by the points $p_1, \ldots, p_n$.

We show how to move the variable p to the first position for one specific case; the treatment of the other cases is similar. Consider the predicate

$$p_1 \textbf{ is on the same side as } p_2 \textbf{ of line } (p, p_3).$$

For points p, p_1, p_2, and p_3 that form a quadrangle, this expression is equivalent to $\neg((\varphi_1 \wedge \varphi_2) \vee (\neg\varphi_1 \wedge \neg\varphi_2))$, where φ_1 and φ_2 are, respectively,

$$p \textbf{ is on the same side as } p_1 \textbf{ of line } (p_2, p_3)$$

and

$$p \textbf{ is on the same side as } p_2 \textbf{ of line } (p_1, p_3).$$

We now define an EuPL^{2c} program that produces a set $S_{\psi'}$ containing at least one representative point of each of the equivalence classes of the plane determined by the lines and circles in the formula $\psi'(p, p_1, \ldots, p_n)$. We then replace

$$\textbf{choose } p \textbf{ such that } \psi(p, p_1, \ldots, p_n)$$

by a formula that computes these representative points, checks for each of them whether the condition $\psi'(p, p_1, \ldots, p_n)$ holds, and returns the first such point.

To start with, let $S_{\psi'}$ be the set $\{p_1, \ldots, p_n\}$. Then for each predicate $\textbf{c-order}(p, p_i, p_j, p_k)$ in ψ', construct the center of the circle through p_i, p_j, and p_k, and add it to $S_{\psi'}$. For each circle appearing in ψ' such that no point of $S_{\psi'}$ occurs in the circle, take the intersections of the circle and the line that connects the center to the fixed points p_0 or p_0', and them to $S_{\psi'}$. Next, construct all of the intersection points of the circles and lines in the formula and add them to $S_{\psi'}$. This deals with all equivalence classes that are single points.

Next, for every pair of points in $S_{\psi'}$ add their midpoints to $S_{\psi'}$; for all pairs of points on a circle then add the midpoints of the arc segments between them, and for each unbounded line segment add the intersection of this segment and a circle whose center is the start of the segment and whose radius is the distance between p_0 and p_0'. This deals with the one-dimensional equivalence classes.

Finally, for all triples of points in $S_{\psi'}$, add their centroids to $S_{\psi'}$ This deals with the two-dimensional equivalence classes and completes the proof. □

All of the languages that we shall discuss from now on are deterministic. We should point out that the discussion of choice operators in this section is designed to *motivate* the subsequent sections, not to apply directly to them. We have illustrated why a language without choice operators is appropriate as a language for modeling Euclidean constructions. This will still be the case for the database languages below, even though some of our current results (such as decidability) no longer hold in the presence of a database.

4.2. Euclidean constructions. We now compare the expressiveness of EuPL with the Euclidean constructions it is intended to model. Our first result in this direction follows directly from the definitions.

THEOREM 4.3. *All* EuPL *multifunctions are constructible with ruler and compass.*

What about the converse? The converse does not hold because our language models *first-order* ruler-and-compass constructions. For an example of a non–first-order ruler-and-compass construction, consider the following.

Example 4.3. Let p, q, r, and s be four different points as in Figure 6. Consider the following construction: First we construct the point q_1 on the line through p and q such that $d(q_1, q) = d(q, p)$ such that $q_1 \neq p$. Then we repeat this construction until we get to the other side of the line through r and s. The result will be the first point to the right of the vertical line (q_n with $n = 10$ in Figure 6).

The computation of this point requires iteration, as stated in the following lemma.

LEMMA 4.1. *The above construction cannot be expressed by an* EuPL *program.*

Proof. From the proof of Theorem 4.1 it follows that any EuPL program can be expressed in FO + poly. If the construction from Example 4.3 would be expressible

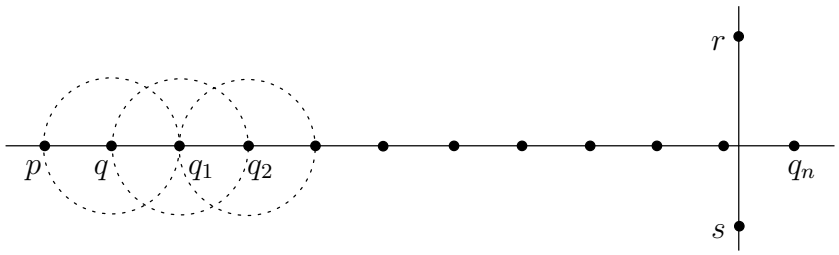


FIG. 6. *Non–first-order construction.*

by an EuPL-program, it would therefore be possible in FO + poly to test whether the distance from p to the vertical through r and s is a multiple of $d(p, q)$, which would allow integers to be definable in FO + poly. Results in [35] would imply the undecidability of FO + poly, a contradiction. □

5. The language EuQL. Our goal is to define a database query language for Euclidean geometry. In this section we describe an initial attempt, EuQL, at defining such a language. EuQL should be a declarative database language, so assignment statements are replaced by predicates. For example, the crossing-point operators become predicates rather than assignments. In addition, the **defined** predicate is not needed, as existential quantifiers can be used instead.

The relations of the input database are finite two-dimensional point relations, i.e., finite tuples of two-dimensional points, represented by real polynomial formulas. The relation R_i, of arity m_i, is an m_i-ary finite two-dimensional point relation, i.e., a $2m_i$-ary relation over the reals. An EuQL query over a schema $R_1, \ldots, R_n$ is of the form

$$Q(R_1, \ldots, R_n) = \{(v_1, \ldots, v_m) \mid \varphi(R_1, \ldots, R_n, v_1, \ldots, v_m)\} ,$$

where φ is a formula in the first-order logic with equality, database predicates, all constant points with real algebraic coordinates, and the following predicates:

1. ⟨var⟩ **is on line** (⟨var⟩, ⟨var⟩),
2. ⟨var⟩ **is on circle** (⟨var⟩, ⟨var⟩, ⟨var⟩),
3. ⟨var⟩ **is in circle** (⟨var⟩, ⟨var⟩, ⟨var⟩),
4. ⟨var⟩ **is on the same side as** ⟨var⟩ **of line** (⟨var⟩, ⟨var⟩),
5. **l-order**(⟨var⟩, ⟨var⟩, ⟨var⟩),
6. **c-order**(⟨var⟩, ⟨var⟩, ⟨var⟩, ⟨var⟩),
7. ⟨var⟩ **is l-l-crossing point of** (⟨var⟩, ⟨var⟩, ⟨var⟩, ⟨var⟩),
8. ⟨var⟩ **is l-c-crossing point of** (⟨var⟩, ⟨var⟩, ⟨var⟩, ⟨var⟩, ⟨var⟩),
9. ⟨var⟩ **is c-c-crossing point of** (⟨var⟩, ⟨var⟩, ⟨var⟩, ⟨var⟩, ⟨var⟩, ⟨var⟩),

EuQL has three constant points o, e_1, and e_2, with (o, e_1) perpendicular to (o, e_2). We shall refer to the line through o and e_1 as the *x-axis* and to the line through o and e_2 as the *y-axis*.

The semantics of EuQL are defined as a function

$$S(Q) : \mathcal{R}_1 \times \cdots \times \mathcal{R}_n \to \mathcal{R} ,$$

where $\mathcal{R}_i$ is the type of relation R_i and $\mathcal{R}$ the type of the result relation of Q. The interpretations of variables, logical connectives, etc., are standard. The other predicates are interpreted in the natural way.

For example, we have the following:

1. $S(v_1 \textbf{ is on line } (v_2, v_3))(r_1, \ldots, r_n)$ is the set of those tuples $(a_{v_1}, a_{v_2}, a_{v_3})$ for which a_{v_2} and a_{v_3} are distinct and a_{v_1}, a_{v_2}, and a_{v_3} are collinear.
2. $S(v_1 \textbf{ is on circle } (v_2, v_3, v_4))(r_1, \ldots, r_n)$ is the set of those tuples $(a_{v_1}, a_{v_2}, a_{v_3}, a_{v_4})$ for which a_{v_1} is on the circle with center a_{v_2} and radius $d(a_{v_3}, a_{v_4})$ and a_{v_3} and a_{v_4} are distinct.

The three special points are interpreted as a coordinate system.

Example 5.1. Given a binary relation R that consists of pairs of points, return the unary relation with the midpoints of each tuple of R:

$$\begin{aligned}\{(p) \mid \exists p_1 \exists p_2 ((&R(p_1,p_2) \wedge p_1 = p_2 \wedge p = p_1)\\ &\vee (R(p_1,p_2) \wedge \neg(p_1 = p_2) \wedge p \textbf{ is on line } (p_1,p_2)\\ &\wedge \exists p_3 \exists p_4 (p_3 \textbf{ is on circle } (p_1,p_1,p_2)\\ &\qquad \wedge p_3 \textbf{ is on circle } (p_2,p_2,p_1)\\ &\qquad \wedge p_4 \textbf{ is on circle } (p_1,p_1,p_2)\\ &\qquad \wedge p_4 \textbf{ is on circle } (p_2,p_2,p_1)\\ &\qquad \wedge \neg(p_3 = p_4) \wedge p \textbf{ is on line } (p_3,p_4))))\}.\end{aligned}$$

Unfortunately, it turns out that EuQL is too powerful. To show why, we define a query that constructs an ellipse, and thus show that EuQL expresses more than just the Euclidean constructions. The construction is similar to the construction of an arbitrary point on an ellipse in EuPL, but by using first-order quantifiers we can essentially simulate choice operators and iterate over all possible choices.

Example 5.2. Given a 4-ary relation of points, for each tuple t return the ellipse with foci t_1 and t_2, and major axis equal to $d(t_3, t_4)$:

$$\begin{aligned}\{(p) \mid \exists t_1 \exists t_2 \exists t_3 \exists t_4 \exists q\\ (R(t_1,t_2,t_3,t_4) \wedge t_2 \textbf{ is on circle } (t_4,t_1,t_3) \wedge \textbf{l-order}(t_3,t_1,t_2)\\ \wedge \textbf{l-order}(t_1,t_2,t_4) \wedge \neg(t_3 = t_4) \wedge \textbf{l-order}(t_3,q,t_4)\\ \wedge p \textbf{ is on circle } (t_1,t_3,q) \wedge p \textbf{ is on circle } (t_2,t_4,q))\}.\end{aligned}$$

THEOREM 5.1. EuQL *can express queries that are not constructible in Euclidean geometry.*

While this shows that EuQL does not match the intuition we had in mind, one might hope that it would still serve as a language between FO + lin and FO + poly. This is not the case, however, as the following result shows.

From Euclid, we know that multiplication can be performed with ruler and compass, and so the following theorem holds.

THEOREM 5.2. *On finite point databases,*[4] EuQL *has the same expressive power*[5] *as* FO + poly.

In order to obtain the desired language, we shall now restrict EuQL in an appropriate way.

[4]We define *finite point databases* as database instances over some database schema $R_1, \ldots, R_n$ in which the interpretation of each relation R_i is a finite set of points in $\mathbf{R}^2$.

[5]Let *can* be the canonical bijection mapping a point p of $\mathbf{R}^2$ to the pair (p^x, p^y) of its real coordinates. An EuQL query Q over an input schema $R_1, \ldots, R_n$ has the same expressive power as an FO + poly query Q' over the schema $R'_1, \ldots, R'_n$, where the arity of R'_i is double the arity of R_i if for any instance $A_1, \ldots, A_n$ over $R_1, \ldots, R_n$, $can(Q(A_1, \ldots, A_n)) = Q'(can(A_1), \ldots, can(A_n))$.

6. The language SafeEuQL. In this section we define a subset of EuQL, called SafeEuQL, that consists of queries whose results, on finite databases, are constructible in Euclidean geometry. This subset consists of those EuQL queries that satisfy a syntactically defined *safety* condition, whose intuition is to restrict the domain over which variables range to be finite, as soon as the input database is finite.

A ⟨disjunction⟩ is defined as a disjunction of ⟨conjunction⟩'s, and a ⟨conjunction⟩ is a conjunction of ⟨factor⟩'s. A ⟨factor⟩ is a ⟨term⟩ or a $\neg$ ⟨term⟩. Finally, a ⟨term⟩ is either $\exists$ ⟨var⟩ (⟨disjunction⟩) or is an EuQL primitive, including the three constant points. We say that an EuQL expression is in *safe-range normal form* if it can be defined as a ⟨disjunction⟩.

We now define the set of variables which are *safe* in an EuQL expression in safe-range normal form. Let R be a relation with attributes of type point. Denote the set of safe variables of an expression φ, with φ in safe-range normal form, by $\mathcal{S}v(\varphi)$. The set $\mathcal{S}v(\varphi)$ then is defined as follows:

1. $\mathcal{S}v(R(v_1, \ldots, v_p))$ equals $\{v_1, \ldots, v_p\}$.
2. For each of the EuQL primitives φ, $\mathcal{S}v(\varphi)$ equals the empty set.
3. $\mathcal{S}v(\exists v\varphi)$ equals $\mathcal{S}v(\varphi) - \{v\}$.
4. $\mathcal{S}v(\neg\varphi)$ equals the empty set.
5. $\mathcal{S}v(\varphi_1 \wedge \varphi_2)$ equals the smallest set S, with respect to $\subseteq$, such that the following properties hold:
   (i) if φ_i is the expression "$v_1 = v_2$" with v_1 or v_2 in S, then both v_1 and v_2 are in S;
   (ii) if φ_i is the expression "v_1 **is l-l-crossing point of** (v_2, v_3, v_4, v_5)" and the variables $v_2, \ldots, v_5$ are in S, then v_1 is in S;
   (iii) if φ_i is the expression "v_1 **is l-c-crossing point of** $(v_2, v_3, v_4, v_5, v_6)$" and the variables $v_2, \ldots, v_6$ are in S, then v_1 is in S;
   (iv) if φ_i is the expression "v_1 **is c-c-crossing point of** $(v_2, v_3, v_4, v_5, v_6, v_7)$" and the variables $v_2, \ldots, v_7$ are in S, then v_1 is in S; and
   (v) $\mathcal{S}v(\varphi_1) \cup \mathcal{S}v(\varphi_2)$ is a subset of S.
   All the above cases also hold for the appropriate variables when the remaining variables are constants. Showing existence of the set S is straightforward.
6. $\mathcal{S}v(\varphi_1 \vee \varphi_2)$ equals $\mathcal{S}v(\varphi_1) \cap \mathcal{S}v(\varphi_2)$.

DEFINITION 6.1. *An* EuQL *query* $\{(v_1, \ldots, v_m) \mid \varphi(R_1, \ldots, R_n, v_1, \ldots, v_m)\}$*, with* φ *in safe-range normal form, is called* safe *if*

1. *for each subformula of* φ *of the form* $\exists v\psi$*, it is the case that* $v \in \mathcal{S}v(\psi)$*, and*
2. *every free variable* v_i *of* φ *is in* $\mathcal{S}v(\varphi)$.

Example 6.1. Consider again the query which computes the midpoints of all tuples of a binary relation R. This query can be expressed with a safe EuQL query as follows:

$$\begin{aligned}\{(p) \mid\ & \exists p_1 \exists p_2 (p_1 = p_2 \wedge R(p_1, p_2) \wedge p = p_1) \\ & \vee \exists p_1 \exists p_2 \exists p_3 \exists p_4 (\neg(p_1 = p_2) \wedge \neg(p_3 = p_4) \wedge R(p_1, p_2) \\ & \quad \wedge p_3 \textbf{ is c-c-crossing point of } (p_1, p_1, p_2, p_2, p_1, p_2) \\ & \quad \wedge p_4 \textbf{ is c-c-crossing point of } (p_1, p_1, p_2, p_2, p_1, p_2) \\ & \quad \wedge p \textbf{ is l-l-crossing point of } (p_1, p_2, p_3, p_4))\}.\end{aligned}$$

The variables p_1 and p_2 are safe in both parts of the disjunction because of the EuQL term $R(p_1, p_2)$. The variables p_3 and p_4 in the second part of the disjunction are safe since they are the two intersection points of circles defined in terms of the safe variables p_1 and p_2. Finally, p is safe because it denotes the intersection point of two lines defined by safe variables.

To show that safety of an EuQL query is a syntactical requirement, consider the query that computes the midpoint of two points as given in Example 5.1. This time, the formula is not safe because p_3, p_4, and p are not safe.

The set of all safe EuQL queries will be called SafeEuQL. The following closure property holds.

THEOREM 6.1. *A* SafeEuQL *query applied to a finite point database yields a finite point database which can be constructed by ruler and compass from the input.*

Proof. Let B be a finite point database and φ a SafeEuQL expression. We show that, when φ is applied to B, every variable v in $\mathcal{S}v(\varphi)$ ranges over a finite domain.

First, let φ be quantifier-free. We prove the claim on the safe variable v in φ by induction on the length of φ, i.e., on the number of propositional connectives in φ.

For the basis, observe that the only SafeEuQL expressions with v as a safe variable are those of the form $R(\ldots, v, \ldots)$, $v = c_1$, v **is l-l-crossing point of** (c_1, c_2, c_3, c_4), v **is l-c-crossing point of** $(c_1, c_2, c_3, c_4, c_5)$, or v **is c-c-crossing point of** $(c_1, c_2, c_3, c_4, c_5, c_6)$ with $c_1, \ldots, c_6$ safe. By assumption the relation R is finite, and thus the claim holds.

Now assume that the claim holds for safe variables in quantifier-free SafeEuQL expressions of length at most k. Let v be a safe variable in the quantifier-free SafeEuQL expression φ of length $k+1$. There are two cases:

1. $\varphi \equiv \psi_1 \vee \psi_2$. From the definition of safety it follows that v is safe in both ψ_1 and ψ_2. By the induction hypotheses, we know that for ψ_1 and ψ_2 applied to B, v ranges over finite domains, say D_1 and D_2. Then when φ is applied to B, v must range over a domain contained in $D_1 \cup D_2$.
2. $\varphi \equiv \psi_1 \wedge \psi_2$. Denote by S the union of $\mathcal{S}v(\psi_1)$ and $\mathcal{S}v(\psi_2)$. By the induction hypothesis, when ψ_1 and ψ_2 are applied to B, each variable of S ranges over a finite domain. Let D be the union of all these domains. Repeat the following process until v is in S. Consider every SafeEuQL primitive in φ which does not occur in any $\neg\psi$, where $\neg\psi$ is a subformula of ψ_1 or ψ_2.
   If the primitive is of the form $v_1 = v_2$ with $v_1 \in S$, then add v_2 to S. If the primitive has the form v_1 **is l-l-crossing point of** (v_2, v_3, v_4, v_5), v_1 **is l-c-crossing point of** $(v_2, v_3, v_4, v_5, v_6)$, or v_1 **is c-c-crossing point of** $(v_2, v_3, v_4, v_5, v_6, v_7)$, where the points $v_2, \ldots, v_7$ are all in S, then add v_1 to S and let D' be the finite set of crossing-points obtained by letting the variables $v_2, \ldots, v_7$ range over the points of D. Add the points in D' to D. The resulting set is still finite, and every variable of S ranges over at most the points in D. Since, by assumption, v is safe in φ, it follows that this process terminates after a finite number of steps. Thus, for φ applied on B, v ranges over a finite set of points.

Note that the case $\varphi \equiv \neg\psi$ cannot occur because it has no safe variables.

Next, consider a SafeEuQL term of the form $\exists v\psi$ with ψ quantifier-free. By the definition of safety, v must be safe in ψ. As a consequence of the first part of the proof, when ψ is applied on B, v ranges over a finite domain, say D_v. Replace the formula $\exists v\psi$ in φ by $D_v(v) \wedge \psi$, which results in a SafeEuQL expression with the same result on B as φ. Since φ has only a finite number of quantifiers, we can repeat this process until we obtain a quantifier-free SafeEuQL expression with the same result as φ on the finite point database B. All (free) variables in this expression range over a finite domain, and thus the result of the expression will also be finite.

Finally, observe that every EuQL primitive can be simulated with ruler and compass. Since every variable in a SafeEuQL expression applied to a finite point database

ranges over a finite set of points, there exists a finite sequence of ruler-and-compass constructions which yields the same set of points as the SafeEuQL expression. Thus, for every SafeEuQL expression, the finite output database can be constructed with ruler and compass from the input database, which concludes the proof. □

THEOREM 6.2. *SafeEuQL has full arithmetical power on the coordinates of safe variables; i.e., we can subtract, add, multiply, and divide coordinates of such variables.*

Proof. Assume that p and q are safe variables. Using the three fixed points as a coordinate system, we write SafeEuQL queries to compute the points with coordinates $(p_1, 0)$, $(0, p_2)$, $(q_1, 0)$, and $(0, q_2)$, where p_1, p_2, q_1, and q_2 are the coordinates of the points p and q, respectively. Without loss of generality, we can therefore assume that p and q are safe variables with coordinates of the form $(p_1, 0)$ and $(q_1, 0)$. If we then consider the well-known ruler-and-compass constructions for multiplication and division, it is easy to see that they can be expressed as SafeEuQL queries. This concludes the proof. □

7. The main results. We now define two query languages which are closed on semi-circular relations. The first, $\mathsf{SafeEuQL}^{\uparrow}$, captures those first-order geometrical constructions that can be described by ruler and compass. The second captures all $\mathsf{FO} + \mathsf{poly}$ expressible queries that map semi-circular relations to semi-circular relations.

To define these languages, we lift the query language SafeEuQL, which is defined on finite point databases, to a language called $\mathsf{SafeEuQL}^{\uparrow}$, which is defined on semi-circular databases. This is done by interpreting these $\mathsf{SafeEuQL}^{\uparrow}$ queries to work on the intensional representations of semi-circular databases defined in section 3.

We use the following convention: R_{poly} refers to a two-dimensional semi-algebraic relation, R_{circ} to a semi-circular relation, and R_{lin} to a two-dimensional semi-linear relation.

Given an LPC-database which, by definition, consists of finite relations of points in the plane, there exists a database consisting of three relations containing the coordinates of the points in the relations L, P, and C, respectively. Indeed, for every point appearing in the relations L, P, or C, we can compute the coordinates of that point with respect to the coordinate system defined by the constant points o, e_1, and e_2, by constructing parallel lines with the line oe_2 (respectively, oe_1) through the points in the finite relations, and then taking the intersection of these lines with the line oe_1 (respectively, oe_2).

In the following, we shall not distinguish between the point and coordinate representation of an LPC-database; i.e., given L, P, and C relations, we will interpret them as points or coordinates depending on the context in which they are used.

7.1. The query language $\mathsf{SafeEuQL}^{\uparrow}$. Before defining $\mathsf{SafeEuQL}^{\uparrow}$, we need two lemmas. The first is straightforward.

LEMMA 7.1. *There exists an* $\mathsf{FO} + \mathsf{poly}$ *query* $Q_{(L,P,C)\to R_{\text{circ}}}$ *that maps the coordinate representation of every intensional LPC-representation of a semi-circular relation to the semi-circular relation it represents.*

LEMMA 7.2. *There exists an* $\mathsf{FO} + \mathsf{poly}$ *query*

$$Q_{R_{\text{circ}}\to(L,P,C)}$$

that maps any semi-circular relation to the coordinate representation of an intensional LPC-representation of this relation.

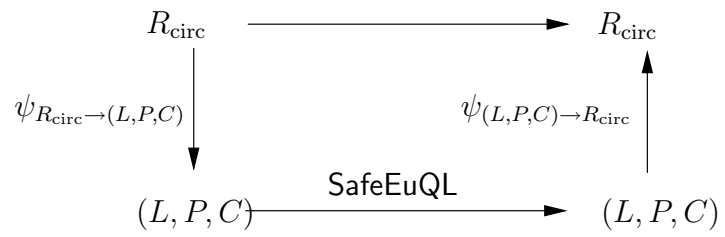


FIG. 7. *The query language* $\mathsf{SafeEuQL}^{\uparrow}$ *is closed on semi-circular relations.*

Proof. Let S be a semi-circular set. It is well known that the topological boundary of S, ∂S can be expressed in $\mathsf{FO}+\mathsf{poly}$ (e.g., using the first-order definition of ε-environments of points). The same is obviously true for the complement of S, S^c and therefore also for the boundary of the complement of S, ∂S^c.

Consider the following sets. Let L_S be the set of all triples (a,b,c) of $\mathbf{R}^3$ such that the line $ax+by+c=0$ has infinitely many points in common with ∂S or with ∂S^c. Let C_S be the set of all triples (a,b,r) of $\mathbf{R}^3$ such that the circle $(x-a)^2+(y-b)^2-r^2=0$ has infinitely many points in common with ∂S or with ∂S^c. Next, let us denote by $\partial^2 S$ the set consisting of all isolated points of ∂S and of ∂S^c and of all end points of half-lines, line segments, and circle segments on ∂S or ∂S^c (a point p is said to be an *end point* of a line segment l with carrier c if there exists an $\varepsilon>0$ and a point q such that $d(p,q)<\varepsilon$ and $q\in c\setminus l$; and end point of a circle segment is defined in a similar way). Let I_S be the set of all triples $(1,0,-a)$ and $(0,1,-b)$ of $\mathbf{R}^3$ such that (a,b) are the coordinates of a point of $\partial^2 S$.

It is clear that all lines and circles in a carrier of S are appearing in $L_S\cup I_S$, respectively C_S, albeit multiple times (except for the lines given by I_S). We can consider the first-order definable equivalence relation $\sim_L$ on $L_S\cup I_S$, defined as $(a,b,c)\sim_L(a',b',c')$ if and only if the equations $ax+by+c=0$ and $a'x+b'y+c'=0$ define the same line (i.e., if and only if $ac'=a'c$ and $bc'=b'c$). We can also consider the first-order definable equivalence relation $\sim_C$ on C_S, defined as $(a,b,r)\sim_L(a',b',r')$ if and only if the equations $(x-a)^2+(y-b)^2-r^2=0$ and $(x-a')^2+(y-b')^2-r'^2=0$ define the same circle (i.e., if and only if $a=a'$, $b=b'$, and $r=\pm r'$). By the definable choice property (see, e.g., Property 1.2 in Chapter 6 of [39]), representatives of each equivalence class can be first-order defined. Once this is done it is easy to obtain two representative points on each line in the L relation of S and the center and a representative point on each circle in the C relation of S.

It remains to be shown that also the P relation of S can be first-order defined. The sets $L_S\cup I_S$ and C_S partition $\mathbf{R}^2$ according to the following first-order definable equivalence relation $\sim_S$: $(x,y)\sim_S(x',y')$ if and only if for all $(a,b,c)\in L_S\cup I_S$, $ax+by+c$ and $ax'+by'+c$ have the same *sign* ($=0$, <0, or >0) and for all $(a,b,r)\in C_S$, $(x-a)^2+(y-b)^2-r^2$ and $(x'-a)^2+(y'-b)^2-r^2$ have the same sign.

Since $\sim_S$ is first-order definable, again by the definable choice property, representatives of each equivalence class of $\sim_S$ can be first-order defined and added to the P relation of S whenever these representatives belong to S. ∎

DEFINITION 7.1. $\mathsf{SafeEuQL}^{\uparrow}$ *is the set of all queries* Q *of the form*

$$Q_{(L,P,C)\to R_{\text{circ}}}\circ Q_{\mathsf{SafeEuQL}}\circ Q_{R_{\text{circ}}\to(L,P,C)},$$

where Q_{SafeEuQL} *is a* SafeEuQL *query (see Figure* 7*).*

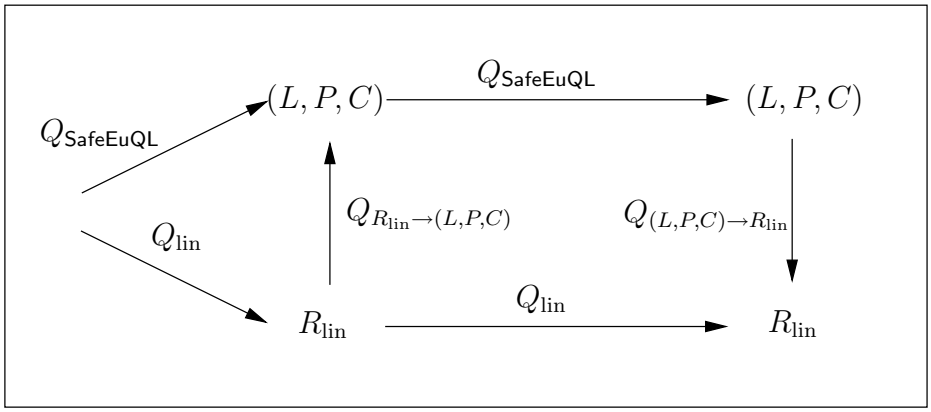


FIG. 8. *Any* $\mathsf{FO}+\mathsf{lin}$ *query on semi-linear relations can be simulated in* SafeEuQL *on the intensional level. The two arrows at the left denote the property that any semi-linear relation can be defined in the language* $\mathsf{FO}+\mathsf{lin}$ *and that any LPC-database can be defined in* SafeEuQL.

A $\mathsf{SafeEuQL}^{\uparrow}$ query is therefore a composition of three queries. First, the query maps a semi-circular relation to its *LPC*-representation. The point representation of this *LPC*-database then is the input of a SafeEuQL query which produces another *LPC*-database. Finally, the coordinate representation of this *LPC*-database is mapped to the semi-circular relation it represents. (A similar "lifting" idea is used in [5].)

The language $\mathsf{SafeEuQL}^{\uparrow}$ is closed on the class of semi-circular relations. This is illustrated in Figure 7. We thus have a syntactically defined subclass of $\mathsf{FO}+\mathsf{poly}$ that is closed on semi-circular relations.

7.2. On semi-linear relations the language $\mathsf{SafeEuQL}^{\uparrow}$ is more expressive than $\mathsf{FO}+\mathsf{lin}$. As discussed in section 3 and Lemma 7.2, every two-dimensional semi-linear relation can be intensionally represented as a finite *LPC*-database. We will show that every $\mathsf{FO}+\mathsf{lin}$ query on semi-linear relations can be simulated in SafeEuQL on the intensional level. Therefore, we can conclude that $\mathsf{SafeEuQL}^{\uparrow}$, on semi-linear databases, can express every $\mathsf{FO}+\mathsf{lin}$ query. On the other hand, it is clear that $\mathsf{SafeEuQL}^{\uparrow}$ is more expressive than $\mathsf{FO}+\mathsf{lin}$ on linear inputs, simply because the latter can output linear databases only, while the former can also produce semi-circular ones.

This is illustrated in Figure 8 and stated more precisely in the following theorem, whose proof follows later in this section.

THEOREM 7.1. *There exist* $\mathsf{FO}+\mathsf{poly}$ *queries* $Q_{R_{\mathrm{lin}}\to(L,P,C)} : R_{\mathrm{lin}} \mapsto (L,P,C)$ *and* $Q_{(L,P,C)\to R_{\mathrm{lin}}} : (L,P,C) \mapsto R_{\mathrm{lin}}$ *such that, for every* $\mathsf{FO}+\mathsf{lin}$ *query* $Q_{\mathrm{lin}} : R_{\mathrm{lin}} \mapsto R_{\mathrm{lin}}$*, there exists a* SafeEuQL *query* $Q_{\mathsf{SafeEuQL}} : (L,P,C) \mapsto (L,P,C)$ *such that*

$$Q_{\mathrm{lin}} = Q_{(L,P,C)\to R_{\mathrm{lin}}} \circ Q_{\mathsf{SafeEuQL}} \circ Q_{R_{\mathrm{lin}}\to(L,P,C)}.$$

The main difficulty is to prove the existence of the SafeEuQL query which simulates a given $\mathsf{FO}+\mathsf{lin}$ query. From Lemmas 7.1 and 7.2, it follows that there exist $\mathsf{FO}+\mathsf{poly}$ queries which translate a linear relation into the coordinate representation of a corresponding *LPC*-database, and vice versa. Moreover, an examination of the proofs of Lemmas 7.1 and 7.2 shows that the corresponding *LPC*-database for a linear relation has an empty *C*-relation; we shall refer to such an encoding as an *LP*-database.

The main difficulty in simulating an FO + lin expression by a SafeEuQL expression is that, in general, subformulas of FO + lin expressions operate in a higher dimensional space, due to the quantifiers that may be present in the expression. Therefore, the LP-representation technique for two-dimensional linear relations has to be generalized to allow the representation of higher dimensional linear relations. For any semi-linear set of $\mathbf{R}^n$, there exist a finite number of n-dimensional hyperplanes which partition $\mathbf{R}^n$ into topologically open, convex cells, such that a finite number of these cells constitute the given semi-linear set. These $(n-1)$-dimensional hyperplanes are finitely represented with n linearly independent points. An n-dimensional point p can be represented within SafeEuQL as a tuple of n two-dimensional points as $((p_1,0),\ldots,(p_n,0))$, where p_i is the ith coordinate of p. Based on this, each n-dimensional semi-linear set is represented in SafeEuQL by a pair of relations (H^n, P^n), where H^n is a $2n^2$-ary relation containing the representation of a finite number of $(n-1)$-dimensional hyperplanes, and where P^n is a $2n$-ary relation containing the representations of the representatives of the partition classes that constitute the semi-linear set. More precisely, the hyperplane in $\mathbf{R}^n$ through the points $\overline{p_i} = (p_{i,1},\ldots,p_{i,n}) \in \mathbf{R}^n$, $1 \le i \le n$, is stored in H^n by the $2n^2$-ary tuple $(((p_{1,1},0)\ldots,(p_{1,n},0)),\ldots,((p_{n,1},0)\ldots,(p_{n,n},0)))$. As an example, the semi-linear subset $\{(x,y,z) \in \mathbf{R}^3 \mid z > 0 \wedge y > 0\}$ of $\mathbf{R}^3$ could be given by (H^3, P^3), with $H^3 = \{((0,0),(0,0),(0,0),(1,0),(0,0),(0,0),(0,0),(1,0),(0,0),(0,0),(0,0),(0,0),(1,0),(0,0),(0,0),(0,0),(0,0),(1,0))\}$ and $P_3 = \{((1,0),(1,0),(1,0))\}$.

Before giving the proof of Theorem 7.1, we prove two lemmas.

The first follows immediately from the fact that we can do arithmetic with Euclidean constructions.

LEMMA 7.3. *Denote by H^n the $2n^2$-ary point relation containing the representation of a finite number of hyperplanes of the n-dimensional space. Assume that x and y are safe variables. There exists a* SafeEuQL *expression SameSide$(H^n; p, q)$ which decides whether the two n-dimensional points p and q are on the same side of each hyperplane of H^n.*

LEMMA 7.4. *Denote by H^n the $2n^2$-ary point relation containing the representation of a finite number of hyperplanes of the n-dimensional space. There exists a* SafeEuQL *expression which computes the relation P^n that contains at least one representative point for every partition class induced by the hyperplanes of H^n.*

Proof. First, add the representation of every coordinate-plane of the n-dimensional space to the relation H^n. The partition induced by the hyperplanes of this new relation H^n is a refinement of the partition induced by the old relation H^n. Therefore, a finite set of representatives of this new partition is also a set of representatives of the old partition.

Next, take n hyperplanes from the relation H^n which are linearly independent, i.e., if no pair is either parallel or equal. We can test this in SafeEuQL as follows. Let p be an arbitrary point on the first hyperplane and q on the second. If, for each point r on the first hyperplane, the point as $r+q-p$ belongs to the second hyperplane, the two hyperplanes are equal or parallel. Since p, q, and r have to be in H^n, they must be safe variables, and by Theorem 6.2 we obtain a SafeEuQL expression which computes $r+q-p$. From Lemma 7.3 we can test in SafeEuQL whether a point belongs to a given hyperplane, and so we can test linear independence of hyperplanes in SafeEuQL.

Every set of n linearly independent hyperplanes of the n-dimensional space intersects in exactly one point. Using Theorem 6.2 again, we construct a SafeEuQL expression for computing this intersection point. Denote by I the set of all of inter-

section points of all sets of n linearly independent hyperplanes of H^n. (I cannot be empty since each hyperplane of H^n intersects at least $n-1$ coordinate planes and therefore contributes at least one point to I.)

We now compute representative points of the *bounded* partition classes induced by the hyperplanes of H^n. Each bounded partition class is convex, since it is the intersection of a finite number of open half-planes. Therefore, the topological closure of a partition class can be written as the convex hull of a finite number of points, the *corner points*, which must be in I, and the barycenter of these corner points can be taken as a representative, which can be expressed in SafeEuQL.

For unbounded partition classes, the set I may not suffice to compute the representative points. To handle this case we use a "bounding box": Each partition class will have a nonempty intersection with this bounding box, and we can choose a representative of the intersection of the partition class with the bounding box.

We now show how to construct this bounding box. For each coordinate plane of the n-dimensional space, we compute, in SafeEuQL, two hyperplanes parallel with this coordinate plane such that all points of I are between these hyperplanes.

Denote by H_B the resulting set of $2n$, and let B be the open n-dimensional bounding box defined by the hyperplanes of H_B. We claim that B has a nonempty intersection with each partition class induced by the hyperplanes of H^n. Indeed, each unbounded partition class has at least one corner point: It intersects at least $n-1$ coordinate planes which were added to H^n. This corner point was obtained from intersections of hyperplanes of H^n and is therefore contained in B. Since B is open, there exists a neighborhood of the corner point which is completely contained within B. The corner point, however, is also in the topological closure of its partition class, and therefore, this neighborhood has a nonempty intersection with the partition class. Therefore B has a nonempty intersection with the partition class. Finally, bounded partition classes are completely contained within B, since their topological closure can be written as the convex hull of points of I.

Let I' be the set of all intersection points of n linearly independent hyperplanes of $H^n \cup H_B$, which can be computed in SafeEuQL. The finite set of points P^n containing the barycenter of each n-tuple of points from I' contains, for each partition class induced by the hyperplanes of H^n, a representative of the intersection of that partition class with B. Since each partition class has a nonempty intersection with B, the set P^n contains a representative for each partition class induced by the hyperplanes of H^n. □

Proof of Theorem 7.1. Assume that Q_{lin} is an FO + lin query defined by a formula φ of FO + lin. Let I_{lin} be an arbitrary two-dimensional linear relation, and let O_{lin} be the result of Q_{lin} applied on I_{lin}.

Denote the LP-representations for I_{lin} and O_{lin} by I_L, I_P and O_L, O_P, respectively, which can be computed in FO + poly (see Lemma 7.2). We now construct SafeEuQL queries Q_L and Q_P such that for any input relation I_{lin} with LP-representation I_L and I_P, $Q_L(I_L, I_P) = O_L$ and $Q_P(I_L, I_P) = O_P$, where O_L and O_P are an LP-representation of the output $O_{lin} = Q_{lin}(I_{lin})$.

We prove this by induction on the structure of φ. For each subformula of φ with n free variables, we construct two SafeEuQL queries that construct the relations H^n and P^n, corresponding to the two parts of the LP-representation of the result.

1. *Atomic formula of the form $I_{lin}(x, y)$.* For each tuple (p, q) of I_L, H^2 should contain a tuple of the form $((p_x, 0), (p_y, 0), (q_x, 0), (q_y, 0))$, where p_x, p_y, q_x, q_y are the coordinates of p and q. For each tuple (p) of I_P, P_I should contain

a tuple $((p_x, 0), (p_y, 0))$. This can easily be expressed in SafeEuQL.

2. *Atomic formula of the form* $\sum_{i=1}^{n} a_i x_i \ \theta \ 0$*, with* $\theta \in \{=, <, >\}$. There exist n linearly independent points $p_1, \ldots, p_n$ such that the smallest affine space containing $p_1, \ldots, p_n$ is precisely the hyperplane given by $\sum_{i=1}^{n} a_i x_i = 0$, and there also exists a point p satisfying $\sum_{i=1}^{n} a_i x_i \ \theta \ 0$. Furthermore, the coordinates of these points can be computed in SafeEuQL. From this, it follows immediately that H^n and P^n can be expressed in SafeEuQL.
3. $\varphi_1(x_1, \ldots, x_m) \vee \varphi_2(x_1, \ldots, x_n)$. If $m \neq n$, assume without loss of generality that $m < n$. Assume that we have already computed the sets (H_1^m, P_1^m) and (H_2^m, P_2^m) in SafeEuQL. We first convert (H_1^n, P_1^n) to a representation of the formula $\varphi_1(x_1, \ldots, x_m, \ldots, x_n)$ in n-dimensional space by padding the representation of each point with $n - m$ zeros.
   The representation (H^n, P^n) of $\varphi_1(x_1, \ldots, x_n) \vee \varphi_2(x_1, \ldots, x_n)$ is then computed as follows. H^n is the union of H_1^n and H_2^n. Let P be a set of representatives of all the partition cells induced by the hyperplanes represented by H^n. The set P^n is then obtained from P as

$$\{x \mid P(x) \wedge \exists y((P_1^m(y) \wedge \mathit{SameSide}(H_1^m; x, y)) \vee (P_2^m(y) \wedge \mathit{SameSide}(H_2^m; x, y)))\} .$$

4. $\neg\varphi_1(x_1, \ldots, x_m)$. In this case, $H^m = H_1^m$ and $P^m = \{x \mid P(x) \wedge \neg\exists y(P_1^m(y) \wedge \mathit{SameSide}(H_1^m; x, y))\}$.
5. $\exists x_i \varphi_1(x_1, \ldots, x_m)$. Let H^m and P^m be the representation of φ_1. For every two hyperplanes of H^m, compute a finite representation of their intersection, and project this representation onto the appropriate $m-1$ dimensions. If the projection of two points coincides, introduce an arbitrary new point, so that we obtain $(m-1)$ linearly independent points denoting a hyperplane in the ith coordinate plane, and add a tuple with these $(m-1)$ points to H^{m-1}. To compute P^{m-1}, let P be the set of all representatives of the partition induced by the hyperplanes in H^{m-1}. Let p be a point of P and q a point of P^m. Compute the intersection point r of the perpendicular to the ith coordinate plane through p with the hyperplane through q parallel with the ith coordinate plane. If q and r belong to the same partition class induced by the hyperplanes of H^m, add p to P^{m-1}. It is straightforward to verify that this can be computed in SafeEuQL.

We have obtained two SafeEuQL queries that compute the relations

$$\{((p_1, 0), (p_2, 0), (q_1, 0), (q_2, 0)) \mid O_L((p_1, p_2), (q_1, q_2))\}$$

and

$$\{((p_1, 0), (p_2, 0)) \mid O_P((p_1, p_2))\}.$$

From these, computing O_L and O_P is trivial. ∎

7.3. On both semi-circular and semi-linear relations, FO + poly is more expressive than SafeEuQL$^{\uparrow}$. We define the fragment of FO + poly that maps semi-circular relations to semi-circular relations. Later on, we will show that this language also allows for the formulation of "nonconstructible" queries and therefore is more powerful than SafeEuQL$^{\uparrow}$.

DEFINITION 7.2. *Let* FO + poly$_{\text{circ}}$ *be the set of* FO + poly *queries that map semi-circular relations to semi-circular relations.*

The following result follows immediately from Lemma 7.5 below.

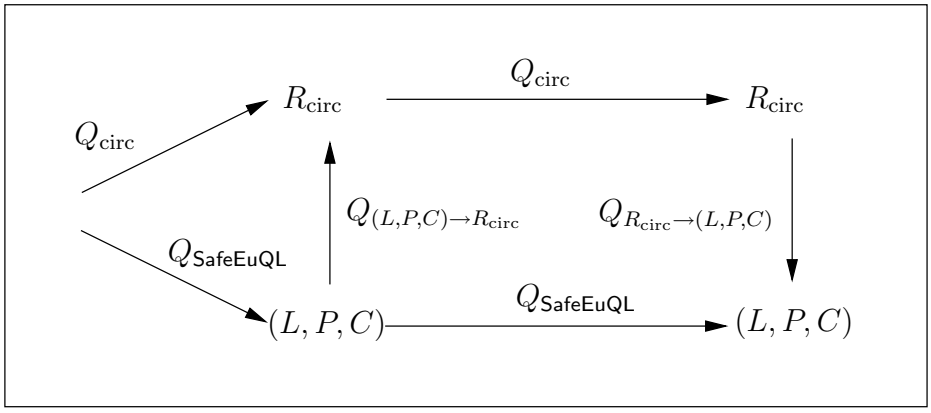


FIG. 9. *The query languages* $\mathsf{SafeEuQL}^{\uparrow}$ *and* $\mathsf{FO}+\mathsf{poly}_{\text{circ}}$. *Again, the arrows at the left denote which relations and databases can be defined in the respective languages.*

THEOREM 7.2. $\mathsf{SafeEuQL}^{\uparrow}$ *is a strict subset of* $\mathsf{FO}+\mathsf{poly}_{\text{circ}}$.

LEMMA 7.5 (Figure 9). *For every* SafeEuQL *query, there exists an* $\mathsf{FO}+\mathsf{poly}_{\text{circ}}$ *query* $Q_{\text{circ}} : R_{\text{circ}} \mapsto R_{\text{circ}}$ *such that*

$$Q_{\mathsf{SafeEuQL}} = Q_{R_{\text{circ}}\to(L,P,C)} \circ Q_{\text{circ}} \circ Q_{(L,P,C)\to R_{\text{circ}}},$$

but not conversely.

Proof. First, we show the existence of the $\mathsf{FO}+\mathsf{poly}_{\text{circ}}$ query Q_{circ}. From Theorem 5.2, it follows that every query expressible in EuQL can be simulated in $\mathsf{FO}+\mathsf{poly}$. The same holds for SafeEuQL, since it is a sublanguage of EuQL. Let $\tilde{Q}_{\text{circ}}$ be the $\mathsf{FO}+\mathsf{poly}$ query which simulates the SafeEuQL query Q_{SafeEuQL}; i.e., $\tilde{Q}_{\text{circ}}$ applied to the coordinate representation of an LPC-database has the same result as Q_{SafeEuql} applied to the LPC-database. Then let Q_{circ} be the query $Q_{(L,P,C)\to R_{\text{circ}}} \circ \tilde{Q}_{\text{circ}} \circ Q_{R_{\text{circ}}\to(L,P,C)}$. Clearly, Q_{circ} is an $\mathsf{FO}+\mathsf{poly}_{\text{circ}}$ query which satisfies the above conditions.

For the second part, consider the query that maps a semi-circular relation consisting of a line segment qr and a point p that is not collinear with q and r to the same relation augmented with two line segments ps and pt such that the angles $\angle pqs$, $\angle pst$, and $\angle ptr$ are equal. This query is expressible in $\mathsf{FO}+\mathsf{poly}$. Since the query maps every semi-circular relation to a semi-circular relation, it belongs to $\mathsf{FO}+\mathsf{poly}_{\text{circ}}$. However, it is not expressible in $\mathsf{SafeEuQL}^{\uparrow}$, since the trisection of an angle cannot be done with ruler and compass, and is therefore not expressible in SafeEuQL. □

We conclude with a remark on $\mathsf{FO}+\mathsf{poly}$, which is defined on R_{poly} relations. The richer class of 2-dimensional figures on which $\mathsf{FO}+\mathsf{poly}$ is defined allows us to express, for example, the construction of an ellipse. Once restricted to semi-circular relations, however, it follows immediately from the definitions that $\mathsf{FO}+\mathsf{poly}$ and $\mathsf{FO}+\mathsf{poly}_{\text{circ}}$ have the same expressive power.

8. Conclusion. Figure 10 summarizes our results.

1. On the bottom level of Figure 10, we have $\mathsf{FO}+\mathsf{lin}$ as a query language on semi-linear relations. Recall that queries concerning Euclidean distance are not expressible in this language. Not only does the data model only

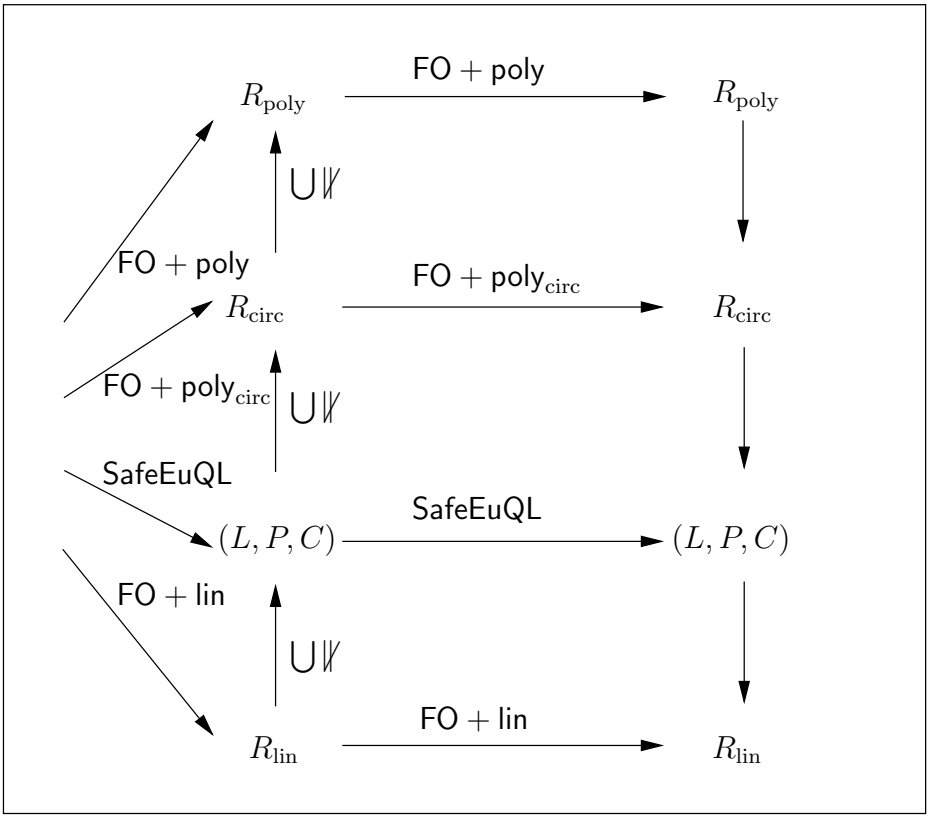


FIG. 10. *Comparison of the different query languages.*

allow semi-linear relations, but, moreover, there are FO + poly queries mapping semi-linear relations to semi-linear relations that are not expressible in FO + lin, for example, the transformation of a relation into its convex hull [41].

2. On the next level, we have more expressive power on (the intensional representation of) semi-linear relations. We can also express queries that involve Euclidean distance. The data model also supports a larger class of relations than the semi-linear ones. All queries expressible in SafeEuQL are constructible by ruler and compass. So, the trisection of a given angle, for instance, is *not* expressible in SafeEuQL.
3. In FO + $\text{poly}_{\text{circ}}$, we gain in expressive power compared to the previous level. For example, trisection of an angle is expressible in this language. The language FO + $\text{poly}_{\text{circ}}$ has the same expressive power as FO + poly on semi-circular relations.
4. On the top level, we have FO + poly. Here, the data model supports all relations definable with polynomial constraints, including queries (e.g., construction of an ellipse) that are not expressible in FO + $\text{poly}_{\text{circ}}$.

Appendix. Formal specification of EuPL. The formal specification of EuPL is as follows. The basic notion is that of a "multifunction," a function that takes a fixed number of input points and constructs a fixed (possibly more than one) number

of output points.

⟨multifunction⟩ →
multifunction ⟨name⟩ '(' ⟨var⟩ (, ⟨var⟩) ∗ ')'
= '(' ⟨type⟩ (, ⟨type⟩) ∗ ')';
begin
⟨statement⟩ (; ⟨statement⟩)∗
end

⟨choice-condition⟩ →
true | false |
⟨var⟩ = ⟨var⟩ |
⟨var⟩ **is on line** '(' ⟨var⟩ , ⟨var⟩ ')' |
⟨var⟩ **is on circle** '(' ⟨var⟩ , ⟨var⟩ , ⟨var⟩ ')' |
⟨var⟩ **is in circle** '(' ⟨var⟩ , ⟨var⟩ , ⟨var⟩ ')' |
⟨var⟩ **is on the same side as** ⟨var⟩ **of line** '(' ⟨var⟩ , ⟨var⟩ ')' |
l-order '(' ⟨var⟩ , ⟨var⟩ , ⟨var⟩ ')' |
c-order '(' ⟨var⟩ , ⟨var⟩ , ⟨var⟩ , ⟨var⟩ ')' |
⟨choice-condition⟩ **and** ⟨choice-condition⟩ |
⟨choice-condition⟩ **or** ⟨choice-condition⟩ |
not ⟨choice-condition⟩ |

Eu-conditions are those used in **if** clauses. They are slightly more general than the conditions used in choice statements.

⟨eu-condition⟩ →
⟨choice-condition⟩ |
defined (⟨var⟩) |
⟨eu-condition⟩ **and**⟨eu-condition⟩ |
⟨eu-condition⟩ **or**⟨eu-condition⟩ |
not⟨eu-condition⟩⟨statement⟩ →
⟨empty statement⟩ |
⟨assignment⟩ |
⟨conditional statement⟩ |
⟨choice⟩ |
⟨result⟩
⟨empty statement⟩ →
⟨assignment⟩ →
⟨var⟩ := **l-l-crossing**(⟨var⟩ , ⟨var⟩ , ⟨var⟩ , ⟨var⟩) |
⟨var⟩ , ⟨var⟩ := **l-c-crossing**(⟨var⟩ , ⟨var⟩ , ⟨var⟩ , ⟨var⟩ , ⟨var⟩) |
⟨var⟩ , ⟨var⟩ := **c-c-crossing**(⟨var⟩ , ⟨var⟩ , ⟨var⟩ , ⟨var⟩ , ⟨var⟩ , ⟨var⟩) |
⟨conditional statement⟩ →
if⟨eu-condition⟩
then⟨statement⟩(;⟨statement⟩)*
else⟨statement⟩(;⟨statement⟩)*
end
⟨choice⟩ →
choose ⟨var⟩**such that** ⟨choice-condition⟩
⟨result⟩ →
result⟨var⟩ (, ⟨var⟩)∗

REFERENCES

[1] D. Abel and B. C. Ooi, eds., *Proceedings of the Third International Symposium on Spatial Databases*, Lecture Notes in Comput. Sci. 692, Springer-Verlag, Berlin, 1993.

[2] F. Afrati, T. Andronikos, and T. Kavalieros, *On the expressiveness of first-order constraint languages*, in Proceedings of the First Workshop on Constraint Databases and Their Applications, Lecture Notes in Comput. Sci. 1034, G. Kuper and M. Wallace, eds., Springer-Verlag, Berlin, 1995, pp. 22–39.

[3] F. Afrati, S. Cosmadakis, S. Grumbach, and G. Kuper, *Linear versus polynomial constraints in database query languages*, in Proceedings of the Second International Workshop on Principles and Practice of Constraint Programming, Lecture Notes in Comput. Sci. 874, A. Borning, ed., Springer-Verlag, Berlin, 1994, pp. 181–192.

[4] S. Basu, R. Pollack, and M.-F. Roy, *On the combinatorial and algebraic complexity of quantifier elimination*, J. ACM, 43 (1996), pp. 1002–1046.

[5] M. Benedikt and L. Libkin, *Safe constraint queries*, SIAM J. Comput., 29 (2000), pp. 1652–1682.

[6] J. Bochnak, M. Coste, and M.-F. Roy, *Real Algebraic Geometry*, Springer-Verlag, Berlin, 1998.

[7] A. Buchmann, ed., *Proceedings of the First International Symposium on Spatial Databases*, Lecture Notes in Comput. Sci. 409, Springer-Verlag, Berlin, 1989.

[8] B. F. Caviness and J. R. Johnson, eds., *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Springer-Verlag, Wien, New York, 1998.

[9] G. E. Collins, *Quantifier elimination for real closed fields by cylindrical algebraic decomposition*, in Automata Theory and Formal Languages, Lecture Notes in Comput. Sci. 33, H. Brakhage, ed., Springer-Verlag, Berlin, 1975, pp. 134–183.

[10] M. J. Egenhofer and J. R. Herring, eds., *Proceedings of the Fourth International Symposium on Spatial Databases*, Lecture Notes in Comput. Sci. 951, Springer-Verlag, Berlin, 1995.

[11] E. Engeler, *Remarks on the theory of geometrical constructions*, in The Syntax and Semantics of Infinitary Languages, Lecture Notes in Math. 72, A. Dold and B. Echraun, eds., Springer-Verlag, Berlin, 1968, pp. 64–76.

[12] E. Engeler, *Foundations of Mathematics*, Springer-Verlag, Berlin, 1992.

[13] H. Eves, *College Geometry*, Jones and Barlett, Boston, 1995.

[14] M. Giusti, J. Heintz, J. E. Morais, J. Morgenstern, and L. M. Pardo, *Straight-line programs in geometric elimination theory*, J. Pure Appl. Algebra, 124 (1998), pp. 101–146.

[15] S. Grumbach, *Implementing linear constraint databases*, in Proceedings of the Second Workshop on Constraint Databases and Applications, Lecture Notes in Comput. Sci. 1191, V. Gaede, A. Brodsky, O. Günther, D. Srivastava, V. Vianu, and M. Wallace, eds., Springer-Verlag, Berlin, 1997, pp. 105–115.

[16] S. Grumbach, P. Rigaux, M. Scholl, and L. Segoufin, *DEDALE, a spatial constraint database*, in Proceedings of the Sixth International Workshop on Database Programming Languages, Lecture Notes in Comput. Sci. 1369, Springer-Verlag, Berlin, 1998, pp. 124–135.

[17] S. Grumbach and J. Su, *Towards practical constraint databases*, in Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, ACM Press, New York, 1996, pp. 28–39.

[18] S. Grumbach and J. Su, *Finitely representable databases*, J. Comput. System Sci., 55 (1997), pp. 273–298.

[19] S. Grumbach and J. Su, *Queries with arithmetical constraints*, Theoret. Comput. Sci., 173 (1997), pp. 151–181.

[20] S. Grumbach, J. Su, and C. Tollu, *Linear constraint query languages: Expressive power and complexity*, in Proceedings of the Logic and Computational Complexity Workshop, Lecture Notes in Comput. Sci. 960, D. Leivant, ed., Springer-Verlag, Berlin, 1994, pp. 426–446.

[21] O. Günther and H.-J. Schek, eds., *Proceedings of the Second International Symposium on Spatial Databases*, Lecture Notes in Comput. Sci. 525, Springer-Verlag, Berlin, 1991.

[22] R. H. Güting, ed., *Advances in Spatial Databases—6th International Symposium (SSD '99)*, Lecture Notes in Comput. Sci. 1651, Springer-Verlag, Berlin, 1999.

[23] M. Gyssens, L. Vandeurzen, and D. Van Gucht, *An expressive language for linear spatial database queries*, in Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, ACM Press, New York, 1998, pp. 109–118.

[24] T. Heath, *The Thirteen Books of Euclid's Elements*, Dover, New York, 1956.

[25] J. Heintz and B. Kuijpers, *Constraint databases, data structures and efficient query evaluation*, in Proceedings of the First International Symposium on Applications of Constraint Databases (CDB'04), Lecture Notes in Comput. Sci. 3074, B. Kuijpers and P. Revesz, eds., Springer-Verlag, Berlin, 2004, pp. 1–24.

[26] D. Hilbert, *Grundlagen der Geometrie*, Teubner, Leipzig, 1899.

[27] J. E. Hopcroft and J. D Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison–Wesley, Reading, MA, 1979.

[28] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz, *Constraint query languages*, in Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Nashville, TN, 1990, pp. 299–213.

[29] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz, *Constraint query languages*, J. Comput. System Sci., 51 (1995), pp. 26–52.

[30] G. Kuper, L. Libkin, and J. Paredaens, eds., *Constraint Databases*, Springer-Verlag, Berlin, 2000.

[31] J. L. Lassez, *Querying constraints*, in Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, ACM Press, New York, 1990, pp. 288–298.

[32] J. Paredaens, J. Van den Bussche, and D. Van Gucht, *Towards a theory of spatial database queries*, in Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, ACM Press, New York, 1994, pp. 279–288.

[33] M. F. Preparata and M. I. Shamos, *Computational Geometry*, Springer-Verlag, New York, 1985.

[34] J. Renegar, *On the computational complexity and geometry of the first-order theory of the reals*, J. Symbolic Comput., 13 (1989), pp. 255–352.

[35] J. Robinson, *Definability and decision problems in arithmetic*, J. Symbolic Logic, 14 (1949), pp. 98–114.

[36] M.-F. Roy, S. Basu, and R. Pollack, *Algorithms in Real Algebraic Geometry*, Algorithms Comput. Math. 10, Springer-Verlag, Berlin, 2003.

[37] M. Scholl and A. Voisard, eds., *Proceedings of the Fifth International Symposium on Spatial Databases*, Lecture Notes in Comput. Sci. 1262, Springer-Verlag, Berlin, 1997.

[38] A. Tarski, *A Decision Method for Elementary Algebra and Geometry*, University of California Press, Berkeley, CA, 1951.

[39] L. van den Dries, *Tame Topology and O-minimal Structures*, Cambridge University Press, Cambridge, UK, 1998.

[40] L. Vandeurzen, M. Gyssens, and D. Van Gucht, *On the desirability and limitations of linear spatial query languages*, in Proceedings of the Fourth International Symposium on Spatial Databases, Lecture Notes in Comput. Sci. 951, M. J. Egenhofer and J. R. Herring, eds., Springer-Verlag, Berlin, 1995, pp. 14–28.

[41] L. Vandeurzen, M. Gyssens, and D. Van Gucht, *On query languages for linear queries definable with polynomial constraints*, in Proceedings of the Second International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Comput. Sci. 1118, E. C. Freuder, ed., Springer-Verlag, Berlin, 1996, pp. 468–481.

[42] M. Ziegler, *Einige unentscheidbare Körpertheorien*, Enseign. Math. (2), 28 (1982), pp. 269–280.

SIAM J. COMPUT.
Vol. 36, No. 6, pp. 1600–1630

QUICKEST FLOWS OVER TIME*

LISA FLEISCHER† AND MARTIN SKUTELLA‡

Abstract. Flows over time (also called dynamic flows) generalize standard network flows by introducing an element of time. They naturally model problems where travel and transmission are not instantaneous. Traditionally, flows over time are solved in time-expanded networks that contain one copy of the original network for each discrete time step. While this method makes available the whole algorithmic toolbox developed for static flows, its main and often fatal drawback is the enormous size of the time-expanded network. We present several approaches for coping with this difficulty. First, inspired by the work of Ford and Fulkerson on maximal s-t-flows over time (or "maximal dynamic s-t-flows"), we show that static length-bounded flows lead to provably good multicommodity flows over time. Second, we investigate "condensed" time-expanded networks which rely on a rougher discretization of time. We prove that a solution of arbitrary precision can be computed in polynomial time through an appropriate discretization leading to a condensed time-expanded network of polynomial size. In particular, our approach yields fully polynomial-time approximation schemes for the NP-hard quickest min-cost and multicommodity flow problems. For single commodity problems, we show that storage of flow at intermediate nodes is unnecessary, and our approximation schemes do not use any.

Key words. network flows, flows over time, dynamic flows, quickest flows, earliest arrival flows, approximation algorithms

AMS subject classifications. 90B06, 90B10, 90B20, 90C27, 90C35, 90C59, 68Q25, 68W25

DOI. 10.1137/S0097539703427215

1. Introduction.

While standard network flows are useful to model a variety of optimization problems, they fail to capture a crucial element of many routing problems: routing occurs over time. In their seminal paper on the subject, Ford and Fulkerson [12, 13] introduced flows with transit times to remedy this and described a polynomial-time algorithm to solve the maximum flow over time, also called the maximum dynamic flow problem.[1] In addition to the normal input for classical network flow problems, each arc also has a transit time. The *transit time* is the amount of

*Received by the editors May 7, 2003; accepted for publication (in revised form) July 21, 2006; published electronically February 20, 2007. Different parts of this work have appeared in a preliminary form in *The quickest multicommodity flow problem*, in Integer Programming and Combinatorial Optimization, Lecture Notes in Comput. Sci. 2337, W. J. Cook and A. S. Schulz, eds., Springer, Berlin, 2002, pp. 36–53, and in *Minimum cost flows over time without intermediate storage*, in Proceedings of the 14th Annual ACM–SIAM Symposium on Discrete Algorithms, Baltimore, MD, 2003, pp. 66–75.

http://www.siam.org/journals/sicomp/36-6/42721.html

†Department of Computer Science, 6211 Sudikoff, Dartmouth College, Hanover, NH 03755 (lkf@cs.dartmouth.edu). The work of this author was supported in part by IBM and by the NSF through grants CCR-0049071 and INT-8902663.

‡Universität Dortmund, Fachbereich Mathematik, 44221 Dortmund, Germany (martin.skutella@uni-dortmund.de). The work of this author was supported in part by the EU Thematic Networks APPOL I+II, Approximation and Online Algorithms, IST-1999-14084 and IST-2001-30012, and by the DFG Focus Program 1126, "Algorithmic Aspects of Large and Complex Networks," grants SK 58/4-1 and SK 58/5-3.

[1]Earlier work on this topic referred to the problems as *dynamic flow* problems. Recently the term *dynamic* has been used in many algorithmic settings to refer to problems with input data that arrives online or changes over time, and the goal of the algorithms described is to modify the current solution quickly to handle the slightly modified input. For the problem of dynamic flows, the input data is available at the start. The solution to the problem involves describing how the optimal flow changes over time. For these reasons, we use the term "flows over time" instead of "dynamic flows" to refer to these problems.

time it takes for the flow to travel from the tail to the head of that arc. In contrast to the classical case of static flows, a *flow over time* in such a network specifies a flow rate entering an arc for each point in time. In this setting, the capacity of an arc limits the rate of flow into the arc at each point in time. In order to get an intuitive understanding of flows over time, one can associate arcs of the network with pipes in a pipeline system for transporting some kind of fluid.[2] The length of each pipeline determines the transit time of the corresponding arc while the width determines its capacity. A precise definition of flows over time is given later in section 2.

Flows over time may be applied to various areas of operations research and have many real-world applications such as traffic control, evacuation plans, production systems, communication networks (e.g., the Internet), and financial flows. Examples and further applications can be found in the survey articles of Aronson [2] and Powell, Jaillet, and Odoni [33]. However, flows over time are most likely significantly harder than their standard flow counterparts. For example, both minimum cost flows over time and fractional multicommodity flows over time are NP-hard [19, 25], even for very simple series-parallel networks.

1.1. Results from the literature.

Maximum flows over time. Ford and Fulkerson [12, 13] consider the problem of sending the maximal possible amount of flow from a source node s to a sink node t within a given time T. This problem can be solved efficiently using one min-cost flow computation on the given network. Ford and Fulkerson show that an optimal solution to this min-cost flow problem can be turned into a maximal flow over time by first decomposing it into flows on paths. The corresponding flow over time starts to send flow on each path at time zero and continues to send flow on each path so long as there is enough time left in the T time units for the flow along the path to arrive at the sink. A flow over time featuring this structure is called *temporally repeated*.

Quickest flows. A problem closely related to the problem of computing a maximal s-t-flow over time is the *quickest s-t-flow problem*: Send a given amount of flow from the source to the sink in the shortest possible time. This problem can be solved in polynomial time by incorporating the algorithm of Ford and Fulkerson in a binary search framework. Using Megiddo's method of parametric search [27], Burkard, Dlaska, and Klinz [3] present a faster algorithm which solves the quickest s-t-flow problem in strongly polynomial time.

Earliest arrival flows. An *earliest arrival flow* is an s-t-flow over time which simultaneously maximizes the amount of flow arriving at the sink before time θ for all $\theta \in [0, T)$. Gale [14] observes that these flows exist, and Wilkinson [35] and Minieka [28] give equivalent pseudo-polynomial-time algorithms to find them. These algorithms essentially use the successive shortest path algorithm (where the transit times are interpreted as arc lengths) in order to find a static flow which is then turned into a flow over time similar to Ford and Fulkerson's algorithm. The resulting solution is also a *latest departure flow*, i.e., a flow over time which simultaneously maximizes the amount of flow departing from the source after time θ for all $\theta \in [0, T)$ (subject to the constraint that the flow is finished by time T). A flow over time which is both an earliest arrival flow and a latest departure flow is called *universally maximal flow over time*. Hoppe and Tardos [23, 22] describe a polynomial-time approximation scheme for the universally maximal flow problem that routes a $1-\varepsilon$ fraction of the maximum

[2] We take a purely macroscopic point of view which does not involve any fluid dynamics.

possible flow that can reach the sink t by time θ for all $0 \leq \theta < T$. Problems with time-dependent arc capacities have been considered by Ogier [29] and Fleischer [10].

Flows over time with costs. Natural generalization of the quickest flows and maximum flows over time can be defined on networks with costs on the arcs. The problem can be to find either a minimum cost flow with a given time horizon or a quickest flow within a given cost budget. Klinz and Woeginger [25] show that the search for a quickest or a maximum s-t-flow over time with minimal cost cannot be restricted to the class of temporally repeated flows. In fact, adding costs has also a considerable impact on the complexity of these problems. Klinz and Woeginger prove NP-hardness results even for the special case of series parallel graphs. Moreover, they show that the problem of computing a maximal temporally repeated flow with minimal cost is strongly NP-hard.

Orlin [30] describes a polynomial-time algorithm to compute an infinite horizon, minimum cost flow over time that maximizes throughput. The infinite horizon problem does not have specified demand and is not concerned with computing how a flow starts and stops, issues that are crucial when flow demands are changing over time.

Quickest transshipments. Another generalization of quickest flows is the quickest transshipment problem: Given a vector of supplies and demands at the nodes, the task is to find a flow over time that satisfies all supplies and demands within minimal time. Unlike the situation for standard (static) network flow problems, this multiple source, multiple sink, single commodity flow over time problem is not equivalent to an s-t maximum flow over time problem. Hoppe and Tardos describe the first polynomial-time algorithm to solve this problem [24, 22]. They introduce the use of *chain decomposable flows* which generalize the class of temporally repeated flows and can also be compactly encoded as a collection of paths. However, in contrast to temporally repeated flows, these paths may also contain backward arcs. Therefore, a careful analysis is necessary to show feasibility of the resulting flows over time. Moreover, the algorithm of Hoppe and Tardos is not practical as it requires a submodular function minimization oracle for a subroutine.

Quickest multicommodity flows over time. In many applications, there are several commodities that must be routed through the same network. While there is substantial literature on the static multicommodity flow problem, hardly any results on multicommodity flows over time are known. Only recently, Hall, Hippler, and Skutella [19] showed that, already in the setting without costs, multicommodity flows over time are NP-hard. Indeed, it is not known if there always exists an optimal solution that can be described in polynomial space.

Discrete vs. continuous time model. All results mentioned so far were originally developed for a discrete time model, i.e., time is discretized into steps of unit length. In each step, flow can be sent from a node v through an arc (v, w) to the adjacent node w, where it arrives $\tau_{(v,w)}$ time steps later; here, $\tau_{(v,w)}$ denotes the given integral transit time of arc (v, w). In particular, the time-dependent flow on an arc is represented by a time-indexed vector in this model. In contrast to this, in the continuous time model the flow on an arc e is a function $f_e : \mathbb{R}^+ \to \mathbb{R}^+$. Fleischer and Tardos [11] point out a strong connection between the two models. They show that many results and algorithms which have been developed for the discrete time model can be carried over to the continuous time model. Since in this paper we mainly concentrate on the continuous time model, we give a more detailed discussion of the interrelation of the two models in section 4.1.

Time-expanded networks. In the discrete time model, flows over time can be described and computed in *time-expanded networks* which were introduced by Ford and Fulkerson [12, 13]. Here we assume that all transit times are integral. A time-expanded network contains a copy of the node set of the underlying "static" network for every discrete time step. Moreover, for every arc e in the static network with transit time τ_e, there is a copy between each pair of time layers with distance τ_e in the time-expanded network. A precise description of time-expanded networks is given in section 4.1. Unfortunately, due to the time expansion, the size of the resulting network grows linearly in T. In the worst case, T is exponential in the input size of the problem. This difficulty has already been pointed out by Ford and Fulkerson.

On the other hand, the advantage of this approach is that it turns the problem of determining an optimal flow over time into a classical static network flow problem on the time-expanded network. This problem can then be solved by well-known network flow algorithms, an approach which is also used in practice to solve flow over time problems. Due to the linear dependency of the size of the time-expanded network on T, such algorithms are termed "pseudopolynomial" since the run time of the algorithm depends on T and not $\log T$. In general, the size of these networks makes the problem solution prohibitively expensive.

1.2. Contributions of this paper. We describe approximation algorithms for flow over time problems. All of our algorithms approximate the minimum time horizon of an optimal flow. Thus an α-approximate solution is a flow that solves the original problem and requires at most α times the optimal time horizon to complete. Different parts of this work have appeared in a preliminary form in [6, 7].

Temporally repeated solutions. Inspired by the work of Ford and Fulkerson, we show in section 3 that static, *length-bounded* flows in the underlying static network lead to provably good multicommodity flows over time that can also be computed efficiently. The resulting approximation algorithm computes temporally repeated solutions and has performance ratio 2. For the more general problem with bounded cost, this approach yields a $(2+\varepsilon)$-approximation algorithm. In this context it is interesting to remember that the problem of computing a quickest temporally repeated flow with bounded cost is strongly NP-hard [25] and therefore does not allow a fully polynomial-time approximation scheme (FPTAS), unless P=NP. The same hardness result holds for quickest multicommodity flows without intermediate node storage and simple flow paths [19]. Finally, since a temporally repeated flow does not use intermediate node storage, our result implies a bound of 2 on the "power of intermediate node storage," i.e., the makespan of a quickest multicommodity flow without intermediate node storage is at most twice as long as the makespan of a quickest flow that is allowed to store flow at intermediate nodes.

Approximation schemes. Another main contribution of this paper is to show that problems that can be solved exactly in the time-expanded network can be solved close to optimally by a static flow computation in a network with polynomial size. A straightforward idea is to reduce the size of time-expanded networks by replacing the time steps of unit length by larger steps. In other words, applying a sufficiently rough discretization of time leads to a *condensed* time-expanded network of polynomial size. However, there is a tradeoff between the necessity to reduce the size of the time-expanded network and the desire to limit the loss of precision of the resulting flow model since the latter results in a loss of quality of achievable solutions.

In section 4 we show that there is a satisfactory solution to this tradeoff problem. An appropriate choice of the step length leads to a condensed time-expanded

network of polynomial size that permits a solution completing within $(1+\varepsilon)$ times the completion of a comparable flow in the continuous-time model, any $\varepsilon > 0$. More precisely, a condensed time-expanded network achieving this precision has n/ε^2 time layers where n is the number of nodes in the given network. One can thus say that the cost of $(1+\varepsilon)$-approximate temporal dynamics for network flow problems is a factor of n/ε^2 in the size of the network.

This observation has potential applications for many problems involving flows over time. In particular, it yields an FPTAS for the NP-hard quickest multicommodity flow problem. Since costs can easily be incorporated into time-expanded networks, our approach can be generalized to yield FPTASs for quickest multicommodity flow problems with cost constraints. Notice that already quickest s-t-flows with bounded cost are NP-hard.

Apart from NP-hard problems, we believe that our result is also of interest for flow problems, like the quickest transshipment problem, which are known to be solvable in polynomial time. While the algorithm of Hoppe and Tardos [24, 22] for the quickest transshipment problem relies on submodular function minimization, the use of condensed time-expanded networks leads to an FPTAS which simply consists of a series of max-flow computations.

No storage. Flows over time raise issues that do not arise in standard network flows. One issue is storage at intermediate nodes. In most applications (such as, e.g., traffic routing, evacuation planning, and telecommunications), storage is limited, undesired, or even prohibited at intermediate nodes. For single commodity problems, most generally the transportation problem with costs, we prove that storage is unnecessary, and our FPTAS does not use any.

Earliest arrival flows. Finally, in section 5 we discuss a variant of time-expanded networks which are suitable for approximating earliest arrival flows. We address the following problem: Given a set of sources with supplies and a single sink, send the supplies to the sink so that the amount of flow arriving at the sink by time θ is $D_t^*(\theta)$, the maximum possible, for all $0 \le \theta$. Instead of using a uniform discretization of time, we introduce "geometrically condensed time-expanded networks" which rely on geometrically increasing time steps. We use this network to obtain a flow that sends $D_t^*(\theta)$ units of flow to the sink by time $\theta(1+\varepsilon)$ for all $0 \le \theta$.

2. Preliminaries. We consider routing problems on a network $\mathcal{N} = (V, A)$ with $n := |V|$ nodes and $m := |A|$ arcs. Each arc $e \in A$ has an associated integral *transit time* or *length* τ_e and a capacity u_e. In the setting with costs, each arc e also has a cost coefficient c_e, which determines the per unit cost for sending flow through the arc. An arc e from node v to node w is sometimes also denoted (v, w); in this case, we write $\text{head}(e) = w$ and $\text{tail}(e) = v$.

2.1. Static flows. We start with the definition of single-commodity flows: Let $S \subseteq V$ be a set of terminals which can be partitioned into a subset of sources S^+ and sinks S^-. Every source node $v \in S^+$ has a supply $D_v \ge 0$ and every sink $v \in S^-$ has a demand $D_v \le 0$ such that $\sum_{v \in S} D_v = 0$. We often consider the case with only one source $s \in V$ and one sink $t \in V$. In this case, we let $d := D_s = -D_t$.

A *static flow* x on $\mathcal{N}$ assigns every arc e a nonnegative flow value x_e such that the *flow conservation constraints*

$$\sum_{e \in \delta^+(v)} x_e \; - \sum_{e \in \delta^-(v)} x_e \; = \; 0 \qquad \text{for all } v \in V \setminus S$$

are obeyed. Here, $\delta^+(v)$ and $\delta^-(v)$ denote the set of arcs e leaving node v ($\text{tail}(e) = v$) and entering node v ($\text{head}(e) = v$), respectively. The static flow x *satisfies the supplies and demands* if

$$\sum_{e\in\delta^+(v)} x_e \;-\; \sum_{e\in\delta^-(v)} x_e \;=\; D_v \qquad\qquad \text{for all } v \in S.$$

For the case of a single source s and a single sink t we also use the term *s-t-flow*. An s-t-flow x satisfying supply $d = D_s = -D_t$ has *value* $|x| = d$. Finally, a flow x is called *feasible* if it obeys the *capacity constraints* $x_e \le u_e$ for all $e \in A$. The cost of a static flow x is defined as

$$c(x) \;:=\; \sum_{e\in A} c_e\, x_e.$$

In the multiple-commodity setting, there is a set of commodities $K = \{1, \dots, k\}$, each of which is defined by a set of terminals $S_i = S_i^+ \cup S_i^- \subseteq V$ and demands and supplies $D_{v,i}$ for $v \in S_i$ and $i \in K$. A *static multicommodity flow* x on $\mathcal{N}$ assigns every arc-commodity pair (e, i) a nonnegative flow value x_e^i such that $x^i := (x_e^i)_{e\in A}$ is a single-commodity flow as defined above for all $i \in K$. The multicommodity flow x satisfies the demands and supplies if x^i satisfies the demands and supplies $D_{v,i}$ for $v \in S_i$. Finally, x is called *feasible* if it obeys the *capacity constraints* $x_e := \sum_{i\in K} x_e^i \le u_e$ for all $e \in A$. In the setting with costs, the cost of a static multicommodity flow x is defined as

$$c(x) \;:=\; \sum_{e\in A}\sum_{i\in K} c_{e,i}\, x_e^i, \tag{1}$$

where $c_{e,i}$ is the cost coefficient associated with arc e and commodity i.

2.2. Flows over time. In many applications of flow problems, static routing of flow as discussed in section 2.1 does not satisfactorily capture the real structure of the problem since not only the amount of flow to be transmitted but also the time needed for the transmission plays an essential role.

A *(multicommodity) flow over time* f on $\mathcal{N}$ with time horizon T is given by a collection of Lebesgue-measurable functions $f_{e,i} : [0, T) \to \mathbb{R}^+$, where $f_{e,i}(\theta)$ determines the rate of flow (per time unit) of commodity i entering arc e at time θ. Transit times are fixed throughout so that flow on arc e progresses at a uniform rate. In particular, the flow $f_{e,i}(\theta)$ of commodity i entering arc e at time θ arrives at head(e) at time $\theta + \tau_e$. Thus, in order to obey the time horizon T, we require that $f_{e,i}(\theta) = 0$ for $\theta \in [T - \tau_e, T)$. In order to simplify notation, we sometimes use $f_{e,i}(\theta)$ for $\theta \notin [0, T)$, implicitly assuming that $f_{e,i}(\theta) = 0$ in this case.

With respect to flow conservation, there are two different models of flows over time. In the model with storage of flow at intermediate nodes, it is possible to hold inventory at a node which is neither a source nor a sink before sending it onward. Thus, the flow conservation constraints are integrated over time to prohibit deficit at any node:

$$\int_0^{\xi} \left(\sum_{e\in\delta^+(v)} f_{e,i}(\theta) \;-\; \sum_{e\in\delta^-(v)} f_{e,i}(\theta - \tau_e) \right) d\theta \;\le\; 0 \tag{2}$$

for all $i \in K$, $\xi \in [0, T)$, $v \in V \setminus S_i^+$. Moreover, we require that equality holds in (2) for $i \in K$, $\xi = T$, and $v \in V \setminus S_i$, meaning that no flow should remain in the

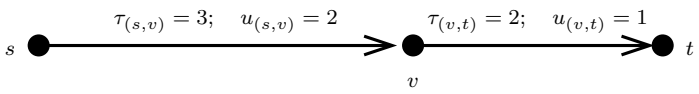


FIG. 1. *An instance of s-t-flows over time given by a network with transit times and capacities on the arcs.*

network after time T. In the model without storage of flow at intermediate nodes we additionally require that equality holds in (2) for all $i \in K$, $\xi \in [0, T)$, and $v \in V \setminus S_i$.

The flow over time f satisfies the supplies and demands if by time T the net flow out of each terminal $v \in S_i$ of commodity i equals its supply $D_{v,i}$:

$$\int_0^T \left(\sum_{e \in \delta^+(v)} f_{e,i}(\theta) \; - \sum_{e \in \delta^-(v)} f_{e,i}(\theta - \tau_e) \right) d\theta \; = \; D_{v,i} \tag{3}$$

for all $i \in K$ and $v \in S_i$. An *s-t-flow over time* is a single commodity flow from a single source s to a single sink t. An s-t-flow over time satisfying supply $d = D_s = -D_t$ has *value* $|f| = d$.

A flow over time f is *feasible* if it obeys the capacity constraints. Here, capacity u_e is interpreted as an upper bound on the rate of flow entering arc e, i.e., a capacity per time unit. Thus, the capacity constraints are $f_e(\theta) \le u_e$ for all $\theta \in [0, T)$ and $e \in A$, where $f_e(\theta) := \sum_{i \in K} f_{e,i}(\theta)$ is the total flow into arc e at time θ.

In the setting with costs, the cost of a flow over time f is defined as

$$c(f) \; := \; \sum_{e \in A} \sum_{i \in K} \int_0^T c_{e,i} \, f_{e,i}(\theta) \; d\theta. \tag{4}$$

Notice that we overload notation here since $c(x)$ is already used to denote the cost of a static flow x. This should not lead to any confusion in the following.

In Figure 1 we give a small illustrating example of s-t-flows over time. In order to send 2 units of flow from s to t in minimum time in the depicted network, one can choose between several alternatives. One is to send flow at rate 2 into the first arc during the time interval $[0, 1)$. Since the transit time of the first arc is 3, the two units of flow will arrive at the intermediate node v during the time interval $[3, 4)$. Thus, one can start to send flow at rate 1 into the second arc at time 3, and it will take 2 time units until time 5 before everything has been sent into the arc. Then, the flow finally arrives at the sink t within the time interval $[5, 7)$. The optimal time horizon is 7. Notice that in this solution flow is stored at the intermediate node v. An alternate solution also with time horizon 7 which avoids storing flow at v can be obtained by sending flow at rate 1 into the first arc during the time interval $[0, 2)$.

2.3. Maximum flows over time and quickest flows. Ford and Fulkerson [12, 13] show how to compute a maximum s-t-flow over time by reducing this problem to a static min-cost flow problem. More precisely, one can turn an optimal solution x to the static s-t-flow problem with objective function[3]

$$\max T\,|x| \; - \; \sum_e \tau_e \, x_e \tag{5}$$

[3]The objective function considered by Ford and Fulkerson is slightly different from (5) since T is replaced by $T + 1$. In contrast to our work, Ford and Fulkerson consider a discrete time setting where time horizon T means that flow can be sent at $T + 1$ discrete points in time $0, 1, 2, \ldots, T$. For more details on the relation between the two models we refer to [11].

into a maximal s-t-flow over time: It is a well-known result from network flow theory that any static flow x in $\mathcal{N}$ can be decomposed into a sum of flows x_P on simple paths $P \in \mathcal{P}$ and flow on cycles. Without loss of generality, flow on cycles is neglected (i.e., canceled) such that x can be written as a sum of path-flows: $x_e = \sum_{P \in \mathcal{P} : e \in P} x_P$ for all $e \in A$. The resulting temporally repeated flow f sends flow at rate x_P into each path $P \in \mathcal{P}$ during the time interval $[0, T - \tau(P))$, where $\tau(P) := \sum_{e \in P} \tau_e$. In other words, f is the sum of path-flows over time f_P with $f_P(\theta) = x_P$ for $\theta \in [0, T - \tau(P))$ and $f_P(\theta) = 0$ otherwise. Feasibility of f immediately follows from feasibility of x. Moreover, the flow value is

$$|f| \;=\; \sum_{P \in \mathcal{P}} (T - \tau(P))\, x_P \;=\; T\,|x| \;-\; \sum_e \tau_e\, x_e. \tag{6}$$

The second equality follows since $(x_P)_{P \in \mathcal{P}}$ is a path-decomposition of x.

For flows over time, a natural objective is to minimize the *makespan*, also called *time horizon*: the time T necessary to satisfy all demands. The *quickest s-t-flow problem* asks for an s-t-flow over time with given value d and minimum time horizon T. This problem can be generalized to the setting with bounded flow cost and multiple sources and sinks (*quickest transshipment problem*) and to the case of multiple commodities.

The most general problem that we consider here is the *quickest (multicommodity) transshipment problem (with bounded cost)* which is defined as follows.

QUICKEST MULTICOMMODITY TRANSSHIPMENT PROBLEM WITH BOUNDED COST
Given: A network (digraph) with capacities, costs, and transit times on the arcs; k commodities, each specified by a set of sources and sinks with supplies and demands; and a cost budget C.
Task: Find a multicommodity flow over time satisfying all supplies and demands with cost at most C and with minimal time horizon T.

Since this problem is NP-hard, we will mostly deal with finding approximate quickest flow. A natural variant of the stated problem is to bound the cost for every single commodity i by a budget C_i, i.e.,

$$\sum_{e \in A} c_{e,i} \int_0^T f_{e,i}(\theta)\; d\theta \;\le\; C_i$$

for all $i \in K$. All of our results also apply to problems with these additional cost constraints.

A note on time and size bounds. Our time bounds are sometimes expressed in terms of T^*, the optimal makespan. Since capacities and transit times are integers, we can assume that at every moment of time some flow is either progressing towards a sink or arriving at the sink. Thus, we obtain a gross upper bound on the optimal makespan: $T^* \le \sum_i d_i + \sum_e \tau_e$. As long as the dependency on T^* is polylogarithmic, the resulting bound is polynomial in size of the input.

3. A simple two-approximation algorithm. In this section we generalize the basic approach of Ford and Fulkerson [12, 13] to the case of multiple commodities (with multiple sources and sinks each) and costs. However, in contrast to the algorithm of Ford and Fulkerson which is based on a (static) min-cost flow computation, the method we propose employs length-bounded static flows.

3.1. Length-bounded static flows. While static flows are not defined with reference to transit times, we are interested in static flows that suggest reasonable routes with respect to transit times. To account for this, we consider decompositions of static flows into paths. We denote the set of all paths starting at some source of commodity i and leading to one of its sinks by $\mathcal{P}_i$. A static (multicommodity) flow x is called *T-length-bounded* if the flow of each commodity $i \in K$ can be decomposed into the sum of flows x^i_P on paths $P \in \mathcal{P}_i$ such that the length $\tau(P)$ of any path $P \in \mathcal{P}_i$ with $x^i_P > 0$ is at most T.

While the problem of computing a feasible static flow that satisfies the multicommodity demands can be solved efficiently, it is NP-hard to find such a flow which is in addition T-length-bounded, even for the special case of a single commodity. This follows by a straightforward reduction from the NP-complete PARTITION problem. On the other hand, the length-bounded flow problem can be approximated within arbitrary precision in polynomial time.

LEMMA 3.1. *If there exists a feasible T-length-bounded static flow x which satisfies the multicommodity demands, then for any $\varepsilon > 0$, a feasible $(1+\varepsilon)T$-length-bounded static flow x' of cost $c(x') \le c(x)$ satisfying all demands can be computed in time polynomial in the input size and $1/\varepsilon$.*

Proof. We first formulate the problem of finding a feasible T-length-bounded static flow as a linear program in path-variables. We assume without loss of generality that each commodity $i \in K$ has exactly one source s_i and one sink t_i with supply and demand $d_i := D_{s_i} = -D_{t_i}$; the general case with several sources and sinks can be handled by introducing one supersource and one supersink for each commodity. Let

$$\mathcal{P}^T_i \;:=\; \{P \in \mathcal{P}_i \mid \tau(P) \le T\} \subseteq \mathcal{P}_i$$

be the set of all s_i-t_i-paths whose lengths are bounded from above by T. The cost of path $P \in \mathcal{P}_i$ is defined as $c_i(P) := \sum_{e \in P} c_{e,i}$. The length-bounded min-cost flow problem can then be written as

$$\begin{aligned}
\min \quad & \sum_{i \in K} \sum_{P \in \mathcal{P}^T_i} c_i(P)\, x^i_P \\
\text{s.t.} \quad & \sum_{P \in \mathcal{P}^T_i} x^i_P \;\ge\; d_i && \text{for all } i \in K, \\
& \sum_{i \in K} \sum_{P \in \mathcal{P}^T_i : e \in P} x^i_P \;\le\; u_e && \text{for all } e \in A, \\
& x^i_P \;\ge\; 0 && \text{for all } i \in K,\ P \in \mathcal{P}^T_i.
\end{aligned}$$

Unfortunately, the number of paths in $\mathcal{P}^T_i$ and thus the number of variables in this linear program are in general exponential in the size of the underlying network $\mathcal{N}$. If we take the dual of the program we get

$$\begin{aligned}
\max \quad & \sum_{i \in K} d_i\, z_i \;-\; \sum_{e \in A} u_e\, y_e \\
\text{s.t.} \quad & \sum_{e \in P} (y_e + c_{e,i}) \;\ge\; z_i && \text{for all } i \in K,\ P \in \mathcal{P}^T_i, \\
& z_i, y_e \;\ge\; 0 && \text{for all } i \in K,\ e \in A.
\end{aligned}$$

The corresponding separation problem can be formulated as a length-bounded shortest path problem: Find a shortest s_i-t_i-path P with respect to the arc weights $y_e + c_{e,i}$

whose length $\tau(P)$ is at most T, i.e., $P \in \mathcal{P}_i^T$. While this problem is NP-hard [15], it can be solved approximately in the following sense: For any $\varepsilon > 0$, one can find in time polynomial in the size of the network $\mathcal{N}$ and $1/\varepsilon$ an s_i-t_i-path $P \in \mathcal{P}_i$ with $\tau(P) \leq (1+\varepsilon)T$ whose length with respect to the arc weights $y_e + c_{e,i}$ is bounded from above by the length of a shortest path in $\mathcal{P}_i^T$ [21, 26, 31]. Using the equivalence of optimization and separation [17], this means for our problem that we can find in polynomial time an optimal solution to a modified dual program which contains additional constraints corresponding to paths of length at most $(1+\varepsilon)T$. To be more precise, we find an optimal solution to a linear program that is more constrained than the above dual:

$$\begin{aligned} \max \quad & \sum_{i \in K} d_i \, z_i \;-\; \sum_{e \in A} u_e \, y_e & & \\ \text{s.t.} \quad & \sum_{e \in P} (y_e + c_{e,i}) \;\geq\; z_i & & \text{for all } i \in K,\ P \in \tilde{\mathcal{P}}_i, \\ & z_i, y_e \;\geq\; 0 & & \text{for all } i \in K,\ e \in A, \end{aligned}$$

where $\mathcal{P}_i^T \subseteq \tilde{\mathcal{P}}_i \subseteq \mathcal{P}_i^{(1+\varepsilon)T}$ for all $i \in K$. From this dual solution we get a primal solution that sends flow of commodity i only on paths in $\tilde{P}_i$. In particular, since $\tilde{\mathcal{P}}_i \subseteq \mathcal{P}_i^{(1+\varepsilon)T}$, this flow is $(1+\varepsilon)T$-length-bounded. □

Notice that the method described in the proof above relies on the ellipsoid method and is therefore of rather restricted relevance for solving length-bounded flow problems in practice. However, the FPTASs developed in [8, 16] for multicommodity flow problems can be generalized to the case of length-bounded flows: Those algorithms iteratively send flow on shortest paths with respect to some length function. In order to get a T-length-bounded solution, these shortest path computations must be replaced by a procedure that computes the $(1+\varepsilon)T$-length-bounded shortest path.[4]

3.2. The approximation algorithm. Any feasible flow over time f with time horizon T and cost at most C naturally induces a feasible static flow x on the underlying network $\mathcal{N}$ by averaging the flow on every arc over time, i.e.,

$$x_e^i \;:=\; \frac{1}{T} \int_0^T f_{e,i}(\theta) \; d\theta$$

for all $e \in A$ and $i \in K$. By construction, the static flow x is feasible, and it satisfies the following three properties, as explained below:

(i) it is T-length-bounded;
(ii) it satisfies a fraction of $1/T$ of the supplies and demands covered by the flow over time f;
(iii) $c(x) = c(f)/T$.

Due to the fixed time horizon T, flow f can travel only on paths of length at most T. Thus property (3.2) is fulfilled. Property (3.2) follows from (3). Finally, property (3.2) is a consequence of (1) and (4).

On the other hand, given an arbitrary feasible static flow x meeting requirements (3.2), (3.2), and (3.2), it can easily be turned into a feasible flow over time g

[4]The algorithms in [8, 16] return a solution of cost at most $1+\varepsilon$ times the minimum cost, which obeys capacities and serves at least $\frac{1}{1+\varepsilon}$ fraction of the demand. In the context of the approximation algorithm described in the next subsection, this approximate result is sufficient to still obtain the $2+\varepsilon$-approximation guarantee. This is because flow can be sent just a little longer on the chosen paths to fulfill all the demand.

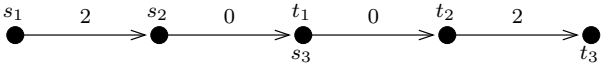


FIG. 2. *An instance of the quickest multicommodity flow problem containing three commodities $i = 1, 2, 3$, each with a single source s_i and a single sink t_i. Commodities* 1 *and* 3 *have demand value* 1*; commodity* 2 *has demand value* 2*. The numbers at the arcs indicate the transit times; all arcs have unit capacity. Note that an arc with* 0 *transit time and capacity* 1 *takes no additional time to cross but lets only one unit of flow through per unit time. A quickest flow with waiting at intermediate nodes allowed takes three time units and stores one unit of commodity* 2 *at the intermediate node $t_1 = s_3$ for two time units. However, if flow cannot be stored at intermediate nodes, an optimal solution takes time* 4*.*

with time horizon $2T$, meeting the same supplies and demands at the same cost as f: For every commodity $i \in K$, pump flow into every path P given by the length-bounded path decomposition of x at the corresponding flow rate x^i_P for T time units; then wait for at most T additional time units until all flow has arrived at its destination. In particular, no flow is stored at intermediate nodes in this solution. Therefore we can state the following structural result on the power of intermediate node storage.

LEMMA 3.2. *Allowing the storage of flow at intermediate nodes in $\mathcal{N}$ saves at most a factor of* 2 *in the optimal makespan. On the other hand, there are instances where the optimal makespan without storage at intermediate nodes is* 4/3 *times the optimal makespan with storage.*

Proof. The bound of 2 follows from the discussion above. In Figure 2 we give an instance with a gap of 4/3 between the optimal makespan without storing and the optimal makespan with storing at intermediate nodes. □

Notice that the gap of 4/3 is not an artifact of the small numbers in the instance depicted in Figure 2. It holds for more general demands and transit times as well: For instance, scale all transit times and capacities of arcs by a factor of q, and multiply all pairwise demands by a factor of q^2. For the new instance, there is still a gap of 4/3 between the optimal makespan without storing and the optimal makespan with storing at intermediate nodes.

In view of the discussion before Lemma 3.2, we can now state the core of our approximation algorithm; see Figure 3. If the given time horizon T is at least as large as the optimum makespan of the given instance, a static flow fulfilling requirements (3.2), (3.2), and (3.2) exists. In the first step of Algorithm LENGTHBOUNDED (Figure 3), we relax the length bound in property (3.2) by a factor $1 + \varepsilon$ so that the step can be performed in polynomial time; see section 3.1.

We state the main result of this section.

THEOREM 3.3. *For the quickest multicommodity transshipment problem with bounded cost, there exists a polynomial-time algorithm that, for any $\varepsilon > 0$, finds a solution of cost at most C (cost budget) with makespan at most $2 + \varepsilon$ times the optimal makespan. Moreover, the computed solution does not store flow at intermediate nodes.*

Proof. We embed Algorithm LENGTHBOUNDED into a binary search for the optimal makspan T^*. After $O(\log T^*/\varepsilon')$ steps, we get a guess of the optimal makespan with precision $1 + \varepsilon'/4$ for any $\varepsilon' > 0$. That is, we get T with

$$T^* \;\le\; T \;\le\; (1 + \varepsilon'/4)T^*.$$

If we call Algorithm LENGTHBOUNDED using T and $\varepsilon := \varepsilon'/4$, we get a flow over

ALGORITHM LENGTHBOUNDED

Input: An instance of the quickest multicommodity transshipment problem with cost bound C; tentative time horizon T; precision $\varepsilon > 0$.

Output: A flow over time satisfying all supplies and demands with makespan at most $(2+\varepsilon)T$ and cost bounded by C; or the information that T is strictly smaller than the optimal makespan.

1. Compute a static flow x such that
   - x is $(1+\varepsilon)T$-length bounded;
   - x satisfies a fraction $1/T$ of the supplies and demands;
   - $c(x) \le C/T$;

   or decide that no such flow exists. In the latter case stop and output "$T < T^*$".
2. Turn x into a flow over time satisfying all supplies and demands with makespan at most $(2+\varepsilon)T$ and cost bounded by C (see discussion in section 3.2).

FIG. 3. *The core of the* $(2+\varepsilon)$*-approximation algorithm.*

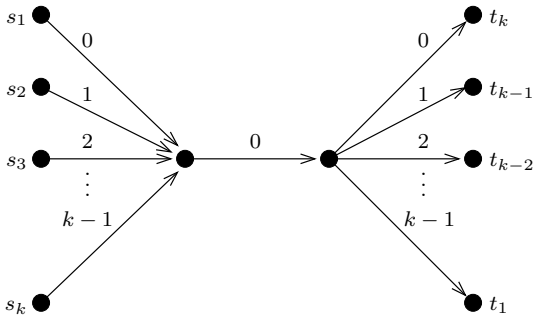


FIG. 4. *An instance with* k *commodities showing that the analysis in the proof of Theorem* 3.3 *is tight. All arcs have unit capacity and transit times as depicted above. The demand value of every commodity is* 1. *A quickest flow needs* $T^* = k$ *time units. However, any static flow can satisfy at most a fraction of* $1/k$ *of the demands. In particular, the makespan of the resulting flow over time is at least* $2k-1$.

time with makespan $(2+\varepsilon)T \le (2+\varepsilon')T^*$. In the last inequality we assume that ε' is chosen *small enough* (i.e., $\varepsilon' \le 4$). □

In Figure 4 we present an instance which shows that the analysis in the proof of Theorem 3.3 is tight. That is, the performance guarantee of the discussed approximation algorithm is not better than 2.

3.3. Avoiding the length-bounded flow computation. In contrast to Ford's and Fulkerson's temporally repeated flows, the flows over time resulting from T-length-bounded static flows described before Lemma 3.2 do not necessarily use flow-carrying paths as long as possible with respect to the time horizon $2T$. Instead, we stop sending flow into all paths at the same time T. In the following we argue that such a flow over time can easily be turned into a temporally repeated flow.

We simply extend the time interval $[0, T)$ during which flow is being sent into each path P such that the last flow on path P arrives at the sink exactly at time $2T$. Thus, the extended time interval for path P is $[0, 2T - \tau(P))$. On the other hand, we

compensate for the enlarged time interval by scaling the flow rate x_P^i on each path P to

$$\tilde{x}_P^i \;:=\; \frac{T}{2T-\tau(P)}\, x_P^i \;\le\; x_P^i.$$

Notice that the resulting temporally repeated flow obeys capacities since we have not increased the flow rate on any path. Moreover, by choice of $\tilde{x}_P^i$ the amount of flow that is sent on any path P has remained unchanged because $\tilde{x}_P^i(2T-\tau(P)) = x_P^i T$.

For the setting without costs, this observation also implies that the length-bounded flow computation in our algorithm can be replaced by a standard (and presumably faster) flow computation with costs, where transit times on arcs are interpreted as cost coefficients. To simplify the presentation of this result, we restrict to the case of only one source s_i and one sink t_i for every commodity $i \in K$. The scaled flow rates $\tilde{x}_P^i$ discussed above define a static multicommodity flow $\tilde{x}$. Let $|\tilde{x}^i|$ be the s_i-t_i-flow value of commodity i in $\tilde{x}$, and let d_i be the demand of commodity i. Since the temporally repeated flow with time horizon $2T$ and flow rates $\tilde{x}_P^i$ sends exactly d_i units of commodity i from s_i to t_i, it follows from (6) that

$$2T\,|\tilde{x}^i| \;-\; \sum_{e\in A} \tau_e\, \tilde{x}_e^i \;=\; d_i \qquad \text{for all } i \in K. \tag{7}$$

On the other hand, any feasible static multicommodity flow $\tilde{x}$ fulfilling (7) can easily be turned into a temporally repeated flow over time satisfying all demands within time $2T$: Compute any path decomposition of $\tilde{x}$ and send flow into every path at the corresponding flow rate as long as there is enough time left for the flow to arrive at its sink before time $2T$. This follows again from (6). We can now prove the following slightly improved approximation result which does not rely on a length-bounded static flow computation.

THEOREM 3.4. *There exists a 2-approximation algorithm for the quickest multicommodity transshipment problem. Moreover, the computed solution does not store flow at intermediate nodes.*

Proof. For the sake of simplicity, we again restrict to the case of one single source s_i and one sink t_i for every commodity $i \in K$. The algorithm first solves the following flow problem

$$\begin{aligned} &\min\ T \\ &\text{s.t.}\quad 2T\,|\tilde{x}^i| \;-\; \sum_{e\in A} \tau_e\, \tilde{x}_e^i \;=\; d_i \qquad \text{for every } i \in K, \\ &\qquad (\tilde{x}_e^i)_{e\in A,\, i\in K} \text{ is a feasible multicommodity flow.} \end{aligned}$$

Notice that this program is nonlinear since both T and the flow values $|x^i|$ are variables. On the other hand, it can be seen as a parametric multicommodity circulation problem by introducing an arc of infinite capacity and cost $2T$ from t_i to s_i for all $i \in K$. The problem can thus be solved in polynomial time.

It follows from (7) and the discussion above that the optimal solution value T is a lower bound on the time horizon of a quickest flow. Finally, the static flow $\tilde{x}$ can be turned into a temporally repeated flow over time with time horizon $2T$ by taking an arbitrary path-decomposition of $\tilde{x}$.[5] ∎

[5]If the path-decomposition contains paths of length longer than $2T$, these paths contribute negatively to the right-hand side of (7). Thus we can remove them along with flow on a set of shorter paths and obtain a smaller path-decomposition.

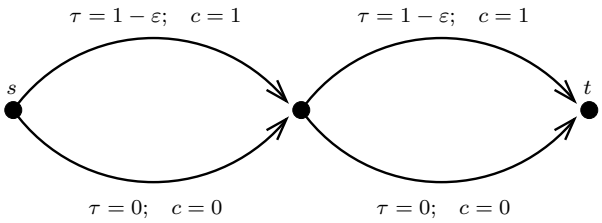


FIG. 5. *An instance showing that the cost of a temporally repeated flow depends on the particular path-decomposition of the underlying static s-t-flow. Consider the static flow of value* 2 *in the depicted network with flow value* 1 *on every arc. The time horizon is set to* 2*. There are two possible decompositions of this static flow into a sum of two path-flows. One of them leads to a temporally repeated flow of cost* $2 + 2\varepsilon$*; the other one has cost* 2ε*.*

Unfortunately, this result cannot be generalized to the quickest multicommodity transshipment problem with costs. The reason is that the cost of a temporally repeated flow is not uniquely determined by the underlying static flow but also depends on the chosen path decomposition. This has already been observed in [25]. We give an example in Figure 5. In fact, it can be shown by a reduction of the NP-complete problem PARTITION that finding a path-decomposition of a given static flow x yielding a cheapest temporally repeated flow is NP-hard.

4. Approximation schemes for quickest flows. In this section we present a framework for obtaining FPTASs for various quickest flow problems. In section 4.1 we introduce special time-expanded networks of polynomial size which are the backbone of this framework. In section 4.2 we present the basic idea of our approach and point out fundamental problems that need to be solved in order to make it work. Then, in section 4.3 we discuss the special case of acyclic graphs. Under the assumption that storage of flow at intermediate nodes is allowed, acyclic graphs are amenable to a simple analysis. On the other hand, we show in section 4.4 that optimal single commodity flows over time (with costs) do not require storage. Based on this insight, we give an FPTAS for the quickest transshipment problem with bounded cost which does not use storage at intermediate nodes in section 4.5. We describe a generalization of our approach to the multicommodity flow setting in section 4.6.

4.1. Condensed time-expanded networks. Traditionally, flows over time are solved in a time-expanded network. Given a network $\mathcal{N} = (V, A)$ with integral transit times on the arcs and an integral time horizon T, the *T-time-expanded network* of $\mathcal{N}$ denoted $\mathcal{N}^T$ is obtained by creating T copies of V, labeled V_0 through V_{T-1}, with the θth copy of node v denoted v_θ, $\theta = 0, \ldots, T-1$. The flow that passes through V_θ corresponds to flow over time in the interval $[\theta, \theta + 1)$. For every arc $e = (v, w)$ in A and $0 \leq \theta < T - \tau_e$, there is an arc e_θ from v_θ to $w_{\theta+\tau_e}$ with the same capacity and cost as arc e. For each terminal $v \in S_i$, $i \in K$, there is an additional infinite capacity *holdover arc* from v_θ to $v_{\theta+1}$ for all $0 \leq \theta < T - 1$, which models the possibility to hold flow at node v in the time interval $[\theta, \theta+1)$. We assume without loss of generality that a source (sink) has no incoming (outgoing) arc in $\mathcal{N}$. Thus, a terminal is never an intermediate node on a path flow.[6] We treat the first copy v_0 of a source $v \in S_i^+$ as the corresponding source in $\mathcal{N}^T$ and treat the last copy v_{T-1} of a sink $v \in S_i^-$ as the corresponding sink in $\mathcal{N}^T$. In the model with storage of flow at intermediate nodes, we introduce holdover arcs for all nodes $v \in V$. An illustration of a time-expanded network is given in Figures 6(a) and (b).

[6]Under this assumption, the flow storage level of commodity i at $v \in S_i$ never exceeds $D_{v,i}$.

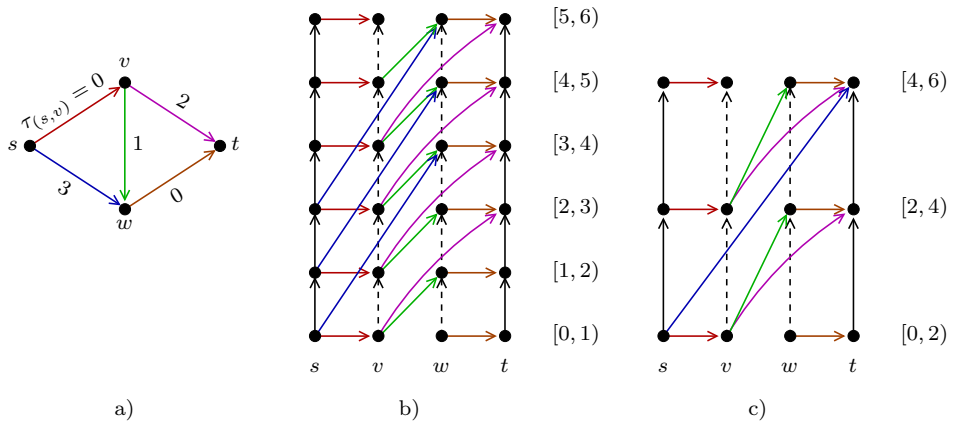


FIG. 6. (a) *A static network $\mathcal{N}$ with transit times on the arcs, one source s, and one sink t;* (b) *the corresponding time-expanded network $\mathcal{N}^T$ with time horizon $T = 6$;* (c) *the condensed time-expanded network $\mathcal{N}^T/\Delta$ with $\Delta = 2$. Notice that the transit time of arc (s, w) has been rounded up to 4 in $\mathcal{N}^T/\Delta$ since the original transit time 3 is not a multiple of Δ (see also section 4.2).*

Any static (multicommodity) flow in this time-expanded network corresponds to a (multicommodity) flow over time of equal cost: For any commodity, interpret the flow on arc e_θ as the flow rate entering arc $e = (v, w)$ in the time interval $[\theta, \theta + 1)$. Similarly, any flow over time completing by time T corresponds to a flow in $\mathcal{N}^T$ of the same value and cost obtained by setting the flow on e_θ to be the average flow rate into e over the interval $[\theta, \theta+1)$. More details can be found below in Lemma 4.1 (set $\Delta := 1$). Thus, we may solve any flow over time problem by solving the corresponding static flow problem in the time-expanded graph.

One problem with this approach is that the size of $\mathcal{N}^T$ depends linearly on T so that if T is not bounded by a polynomial in the input size, this is not a polynomial-time method of obtaining the required flow over time. However, if all arc lengths are a multiple of $\Delta > 0$ such that $\lceil T/\Delta \rceil$ is bounded by a polynomial in the input size, then instead of using the T-time-expanded network, we may rescale time and use a condensed time-expanded network that contains only $\lceil T/\Delta \rceil$ copies of V. Since in this setting every arc corresponds to a time interval of length Δ, capacities are multiplied by Δ. We denote this condensed time-expanded network by $\mathcal{N}^T/\Delta$ and the copies of V in this network by $V_{\rho\Delta}$ for $\rho = 0, \ldots, \lceil T/\Delta \rceil - 1$. Copy $V_{\rho\Delta}$ corresponds to flow through V in the interval $[\rho\Delta, (\rho + 1)\Delta)$. An illustration is given in Figure 6(c).

LEMMA 4.1. *Suppose that all arc lengths are multiples of Δ and T/Δ is an integer. Then, any (multicommodity) flow over time that completes by time T corresponds to a static (multicommodity) flow of equal cost in $\mathcal{N}^T/\Delta$, and any flow in $\mathcal{N}^T/\Delta$ corresponds to a flow over time of equal cost that completes by time T.*

Proof. Given an arbitrary (multicommodity) flow over time, a modified flow over time of equal value and cost can be obtained by averaging the flow value of every commodity on any arc in each time interval $[\rho\Delta, (\rho + 1)\Delta)$, $\rho = 0, \ldots, T/\Delta - 1$. This modified flow over time defines a static (multicommodity) flow in $\mathcal{N}^T/\Delta$ in a canonical way. Notice that the capacity constraints are obeyed since the total flow starting on arc e in interval $[\rho\Delta, (\rho + 1)\Delta)$ is bounded by Δu_e. The flow values on the holdover arcs are defined in such a way that flow conservation is obeyed in every node of $\mathcal{N}^T/\Delta$.

On the other hand, a static (multicommodity) flow on $\mathcal{N}^T/\Delta$ can easily be turned into a flow over time. The static flow on an arc with tail in $V_{\rho\Delta}$ is divided by Δ and

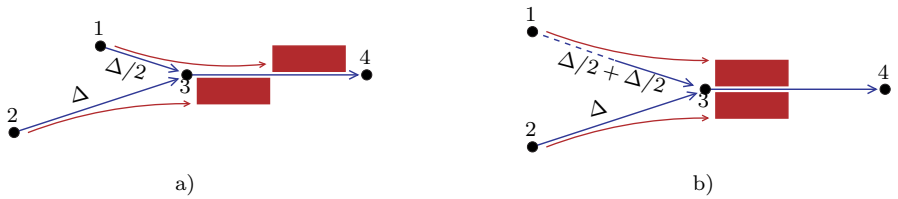


FIG. 7. (a) *A flow over time in the original network. The two packets of flow originating at nodes* 1 *and* 2 *are sent one after another into arc* $(3,4)$. (b) *The "same" flow over time in the network with rounded transit times causes congestion on arc* $(3,4)$ *since the two packets of flow arrive simultaneously on the arc.*

sent for Δ time units starting at time $\rho\Delta$. If the head of the arc is in $V_{\sigma\Delta}$ for $\sigma \geq \rho$, then the length of the arc is $(\sigma - \rho)\Delta$, and the last flow (sent before time $(\rho+1)\Delta$) arrives before time $(\sigma+1)\Delta$. Note that if costs are assigned to arcs of $\mathcal{N}^T/\Delta$ in the natural way, then the cost of the flow over time is the same as the cost of the corresponding flow in the time-expanded graph. □

If we drop the condition that T/Δ is integral, we get the following slightly weaker result.

COROLLARY 4.2. *Suppose that all arc lengths are multiples of* Δ. *Then, any (multicommodity) flow over time that completes by time* T *corresponds to a static (multicommodity) flow of equal value and cost in* $\mathcal{N}^T/\Delta$, *and any flow in* $\mathcal{N}^T/\Delta$ *corresponds to a flow over time of equal value that completes before time* $T+\Delta$.

4.2. Outline of an approximation scheme. The basic idea of our algorithm is to round up transit times to the nearest multiple of Δ for an appropriately chosen Δ, solve the static flow problem in the corresponding Δ-condensed time-expanded network, and then translate this flow back to the setting of the original transit times. In order to obtain provably good solutions in this way, one has to make sure that the following two conditions are fulfilled:

I. the makespan of an optimal solution to the instance with increased transit times (represented by the condensed time-expanded network) approximates the makespan of an optimal solution in the original setting;

II. the solution to the instance with increased transit times can be transformed into a flow over time with original arc lengths without too much loss in flow value.

Before discussing how to fulfill these conditions, we first give some simple examples to show that nontrivial problems have to be dealt with to address both I and II.

Consider first a (sub)network consisting of four nodes $\{1,2,3,4\}$ and three arcs $(1,3)$, $(2,3)$, and $(3,4)$ with unit capacity depicted in Figure 7(a). The transit times are $\tau_{(1,3)} = \Delta/2$, $\tau_{(2,3)} = \Delta$, and $\tau_{(3,4)} = \Delta$. A flow in the graph without rounded transit times can send $\Delta/2$ units of flow in interval $[0, \Delta/2)$ on each path $P_1 = 1 \to 3 \to 4$ and $P_2 = 2 \to 3 \to 4$. Path P_1 will use arc $(3,4)$ in interval $[\Delta/2, \Delta)$, and path P_2 will use arc $(3,4)$ in interval $[\Delta, 3\Delta/2)$. However, if we send flow simultaneously on paths P_1 and P_2 in the network with transit times rounded up to the nearest multiple of Δ, then this will cause a bottleneck on arc $(3,4)$; see Figure 7(b).

Now consider the unit capacity (sub)network depicted in Figure 8. If all transit times are rounded up to the nearest multiple of Δ, we may send Δ units of flow simultaneously on each path from s to t, and each path will use arc (v,t) in a distinct interval of time. If we try to interpret this flow in the network with original transit times, however, each path-flow will try to use arc (v,t) in the same time interval, causing a large bottleneck.

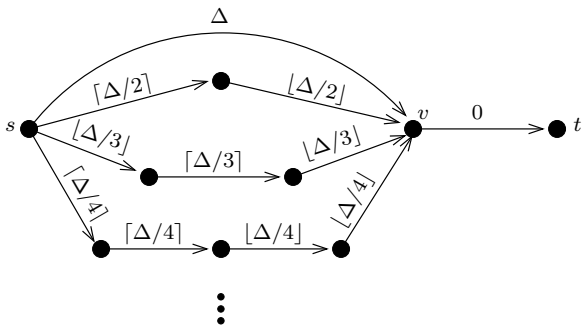


FIG. 8. *In this partially drawn unit capacity network, there are Δ paths from s to v. The i^{th} path contains i arcs, each with transit time roughly Δ/i.*

Condition II can be enforced by allowing storage of flow at nodes: If arc $e = (v, w)$ has length increased by $\Delta' \leq \Delta$, then this can be emulated in the original network by holding flow arriving at w for Δ' time units.[7] More generally, we can state the following observation.

Observation 4.3. Consider a network $\mathcal{N} = (V, A)$ and two transit time vectors $\tau, \tau' \in \mathbb{R}_+^A$ with $\tau_e \leq \tau'_e$ for all $e \in A$. Then, a flow over time in $\mathcal{N}$ with transit times τ' can be emulated in $\mathcal{N}$ with transit times τ by introducing waiting time $\tau'_e - \tau_e$ at the head of every arc e.

For the case of acyclic graphs, we give a simple argument in section 4.3 to show how to uphold condition I when storage is allowed. In section 4.5 we describe a more sophisticated approach that works for general graphs even when storage of flow at nodes is not allowed.

4.3. Acyclic graphs with storage. For acyclic graphs, the existence of a topological ordering of the nodes makes the problems illustrated above fairly easy to resolve. The algorithm is simple: For an appropriate guess of T, choose $\Delta := \frac{\varepsilon}{n}T$, and round transit times up to the nearest multiple of Δ. Form the Δ-condensed time-expanded network, and compute a solution f in this network. Output the solution f' obtained by modifying f by emulating the rounded transit times as described above.

It remains to show condition I: There exists a feasible flow of cost at most C (the given cost budget) satisfying the given (multicommodity) demands D by time $T^*(1+\varepsilon)$ in the network with transit times rounded up to the nearest multiple of Δ. Let f^* be a flow over time of cost at most C that satisfies demands D by time T^*. Moreover, let $\{v_0, v_1, \ldots, v_{n-1}\}$ be a topological ordering of V. Modify f^* to obtain $\hat{f}$ by setting $\hat{f}_e(\theta) = f^*_e(\theta - i\Delta)$ for $e = (v_i, v_j)$ and for all $\theta \in [0, \infty)$. With these modifications, flow traveling on any path that includes v_i is delayed from its original departure from i by exactly $i\Delta$ time units. Thus, flow arriving at node v_j in $\hat{f}$ arrives at most $j\Delta$ time units later than its arrival in f^*, and the time horizon of $\hat{f}$ is $T^* + \Delta(n-1) \leq T^*(1+\varepsilon)$. Since f^* is feasible, so is $\hat{f}$. Since flow travels on the same paths in f^* and $\hat{f}$, we have $c(f^*) = c(\hat{f})$, and $\hat{f}$ satisfies the same multicommodity demands as f^*.

Notice that $\hat{f}$ induces a flow in the network with the transit time of an arc (v_i, v_j) equal to $\tau_{ij} + \Delta(j - i) \geq \tau_{ij} + \Delta$. An alternate and equivalent view is that $\hat{f}$ induces a flow with storage in the network with transit time of (v_i, v_j) rounded up to $\tau_{ij} + \delta_{ij}$,

[7] If Δ' is large, then this requires a large amount of additional storage.

the nearest multiple of Δ. In this view, flow sent on (v_i, v_j) is then held at v_j for an additional $\Delta(j-i)-\delta_{ij} \geq 0$ units of time. This latter flow is a flow in the Δ-rounded network, implying the following theorem.

THEOREM 4.4. *In acyclic graphs with node storage, a* $(1+\varepsilon)$*-approximate solution to the quickest cost-bounded multicommodity flow problem can be obtained with a static flow computation in a network with* $O(n^2/\varepsilon)$ *nodes and* $O(nm/\varepsilon)$ *arcs.*

4.4. Minimum cost flows without storage. It follows from the work of Hoppe and Tardos [24, 22] that for the quickest transshipment problem there always exists an optimal solution which does not store flow at intermediate nodes. We generalize this result to the problem with costs and also to the more general case when the flow cost on an arc is a nondecreasing, convex function of the flow rate into the arc. As mentioned in section 4.1, when transit times are integers, the min-cost transshipment over time problem with or without storage at intermediate nodes can be solved by solving the corresponding static flow problem in the time-expanded network[8] $\mathcal{N}^T$.

THEOREM 4.5. *For nondecreasing, convex cost functions, the cost of a minimum cost transshipment over time that does not use intermediate node storage is no more than the cost of a minimum cost transshipment over time using intermediate node storage.*

The details of this proof are not essential for understanding the remainder of the paper.

Proof. Consider a minimum cost transshipment over time with intermediate node storage and a corresponding static min-cost flow x in the time-expanded network $\mathcal{N}^T$. Notice that the set X of all min-cost solutions x is the intersection of the polytope formed by all feasible solutions with a closed convex set given by the convex cost constraint. In particular, X is convex and compact.

For a node $z \in V$, let $x(\delta(z_\theta))$ be the net amount of flow leaving z in the time interval $[\theta, \theta+1)$:

$$x(\delta(z_\theta)) \;:=\; \sum_{e \in \delta^+(z)} x_{e_\theta} - \sum_{e \in \delta^-(z)} x_{e_{\theta-\tau_e}}.$$

Since X is compact, there exists an $x \in X$ minimizing the convex function $F(x) := \sum_{z \in V} \sum_{\theta=0}^{T-1} |x(\delta(z_\theta))|$. We show that x does not send flow along holdover arcs of nodes in $V \setminus S$.

By contradiction, let v_φ be the earliest copy of node $v \notin S$ to send flow along a holdover arc. We have that $x(\delta(v_\varphi)) = -x_{v_\varphi, v_{\varphi+1}} < 0$. Let $[\varphi+q, \varphi+q+1)$ for $q > 0$ and integral be the first time interval after $[\varphi, \varphi+1)$ in which v has more flow leaving it than entering it; i.e., $x(\delta(v_{\varphi+q})) > 0$. We show in the following that $F(x)$ can be decreased by augmenting flow along a cycle in the time-expanded network $\mathcal{N}^T$. This is a contradiction to the choice of x.

Consider a time-expanded network that is infinite in both directions, $\mathcal{N}^{(-\infty,+\infty)}$. Note that $\mathcal{N}^{(-\infty,+\infty)}$ looks the same at v_φ as it does at $v_{\varphi+q}$. However, x in this network looks different at each of these copies of v. We indicate this difference by

[8] Notice that the averaging argument used in the proof of Lemma 4.1 to turn an arbitrary flow over time into a static flow in the time-expanded network also works for the case of convex costs since averaging never increases cost.

coloring the arcs of $\mathcal{N}^{(-\infty,+\infty)}$ as follows. Color transit arc $(i_{\theta-\tau_{ij}}, j_\theta)$

$$\begin{aligned} \text{red} \quad \text{if} \quad & x_{(i_{\theta-\tau_{ij}},j_\theta)} < x_{(i_{\theta-\tau_{ij}-q},j_{\theta-q})} \\ \text{blue} \quad \text{if} \quad & x_{(i_{\theta-\tau_{ij}},j_\theta)} > x_{(i_{\theta-\tau_{ij}-q},j_{\theta-q})} \\ \text{no color} \quad \text{if} \quad & x_{(i_{\theta-\tau_{ij}},j_\theta)} = x_{(i_{\theta-\tau_{ij}-q},j_{\theta-q})}. \end{aligned}$$

All holdover arcs remain colorless. Note that there are no blue arcs leaving V_θ for $\theta \geq T-1$ and there are no red arcs entering V_θ for $\theta \leq q$.

Let P be a simple path consisting of backward red arcs and forward blue arcs from $v_{\varphi+q}$ to a node w_μ with the property that $x(\delta(w_\mu)) < x(\delta(w_{\mu-q}))$. We claim that such a P exists: Since $x(\delta(v_{\varphi+q})) - x(\delta(v_\varphi)) > 0$, node $v_{\varphi+q}$ has either a red arc entering it or a blue arc leaving it. Consider the set of all nodes which can be reached from $v_{\varphi+q}$ on a path consisting of backward red arcs and forward blue arcs. Since $x(\delta(v_{\varphi+q})) - x(\delta(v_\varphi)) > 0$, it follows from flow conservation that there must exist a node w_μ with $x(\delta(w_\mu)) - x(\delta(w_{\mu-q})) < 0$ in this set.

Note that $V(P) \subset \bigcup_{\theta=q}^{T-1} V_\theta$. We define the capacity $u(P)$ of P to be

$$u(P) := \min_{(i_\theta, j_{\theta+\tau_{ij}}) \in P} |x_{(i_\theta, j_{\theta+\tau_{ij}})} - x_{(i_{\theta-q}, j_{\theta-q+\tau_{ij}})}|.$$

We modify x to reduce $|x(\delta(v_\varphi))|$ and $|x(\delta(v_{\varphi+q}))|$. Let

$$\kappa := \min\{u(P),\ -x(\delta(v_\varphi)),\ x(\delta(v_{\varphi+q})),\ x(\delta(w_{\mu-q})) - x(\delta(w_\mu))\} > 0.$$

If an arc $(i_\theta, j_{\theta+\tau_{ij}}) \in P$ is red, then we modify x on $(i_\theta, j_{\theta+\tau_{ij}})$ and $(i_{\theta-q}, j_{\theta-q+\tau_{ij}})$ to

$$x_{(i_\theta, j_{\theta+\tau_{ij}})} := x_{(i_\theta, j_{\theta+\tau_{ij}})} + \kappa \quad \text{and} \quad x_{(i_{\theta-q}, j_{\theta-q+\tau_{ij}})} := x_{(i_{\theta-q}, j_{\theta-q+\tau_{ij}})} - \kappa.$$

If $(i_\theta, j_{\theta+\tau_{ij}}) \in P$ is blue, then

$$x_{(i_\theta, j_{\theta+\tau_{ij}})} := x_{(i_\theta, j_{\theta+\tau_{ij}})} - \kappa \quad \text{and} \quad x_{(i_{\theta-q}, j_{\theta-q+\tau_{ij}})} := x_{(i_{\theta-q}, j_{\theta-q+\tau_{ij}})} + \kappa.$$

Finally, we remove κ units of flow from the path of holdover arcs from v_φ to $v_{\varphi+q}$ and add κ to the path of holdover arcs from $w_{\mu-q}$ to w_μ. Notice that we have augmented flow on a cycle in $\mathcal{N}^T$ by κ. Since the domain of P is restricted to $V(P) \subset \bigcup_{\theta=q}^{T-1} V_\theta$, the flow x is still a feasible solution to our problem.

We next argue that the cost of x is not increased so that x is still in X. Since the flow augmentation transfers an equal amount of flow from one copy of an arc to a parallel copy, if flow costs are linear, this does not change the cost of our solution. Since the sum of flow on these two arcs does not change and we simply move flow so that the flow on each is closer to the average flow on each, if our flow costs are convex and nondecreasing, then the cost of our solution does not increase.

Finally, the augmentation by κ ensures that $|x(\delta(v_\varphi))|$ and $|x(\delta(v_{\varphi+q}))|$ are each reduced by κ and $|x(\delta(w_\mu))| + |x(\delta(w_{\mu-q}))|$ is not increased. (Either $|x(\delta(w_{\mu-q}))| > \kappa$ and $|x(\delta(w_\mu))| < -\kappa$ or, since $\kappa \leq x(\delta(w_{\mu-q})) - x(\delta(w_\mu))$, the quantities $|x(\delta(w_\mu))|$ and $|x(\delta(w_{\mu-q}))|$ exchange values.) Thus, $F(x) = \sum_{z \in V} \sum_{\theta=0}^{T-1} |x(\delta(z_\theta))|$ is decreased by at least $2\kappa > 0$. This concludes the proof. □

Theorem 4.5 implies that we can find a minimum cost flow over time in the time-expanded network *without* holdover arcs for intermediate nodes. We can even state the following stronger result.

COROLLARY 4.6. *For every instance of the minimum cost transshipment over time problem, when costs are nondecreasing convex functions of the flow rate, there exists an optimal solution without intermediate node storage such that any infinitesimal unit of flow visits every node at most once.*

Proof. We first consider the case that there is no cycle of zero cost in $\mathcal{N}$. If some path-flow in an optimal flow visits a node v more than once, it travels along a cycle in $\mathcal{N}$. Therefore the cost of the solution can be decreased by letting the flow wait at v. This is a contradiction to the optimality of the solution.

If there exists zero cost cycles in $\mathcal{N}$, we can increase the cost of every arc by a small amount such that an optimal solution to the modified problem always yields an optimal solution to the original problem. This eliminates cycles of zero cost and thus concludes the proof. □

4.5. General graphs without storage. Here we describe how to adapt the outline given in section 4.2 to yield an FPTAS for the quickest transshipment problem with bounded cost in general graphs. The computed solution does not use any intermediate node storage.

The approach has two main steps. First, we choose Δ small enough so that we can increase the time horizon by a sufficiently large amount relative to Δ to satisfy condition I. Second, we average the flow computed in the rounded network over sufficiently large intervals relative to Δ so that the resulting flow is almost feasible, satisfying condition II. This second step also increases the total time horizon of the flow, but again, by careful choice of Δ, by a sufficiently small amount.

The core of the FPTAS consists of an algorithm which gets as input an instance of the quickest transshipment problem with bounded cost together with a tentative time horizon T and precision $\varepsilon > 0$. The algorithm either finds a feasible solution (i.e., flow over time) with makespan at most $(1 + O(\varepsilon))T$ or decides that T is smaller than the optimal makespan. Throughout this section we denote the optimal makespan by T^*. A detailed description of the algorithm is given in Figure 9.

Before we discuss and analyze this algorithm in more detail, we first remark the following. A $(1 + O(\varepsilon))$-approximate flow over time can be computed by embedding Algorithm FPTAS-CORE into a binary search framework. We can begin with a standard binary search to find lower and upper bounds on the optimal makespan T^*

ALGORITHM FPTAS-CORE

Input: Network $\mathcal{N}$, capacities u, linear costs c, transit times τ, demand vector D, cost bound C, time horizon T, and precision $\varepsilon > 0$.

Output: Feasible flow over time f with time horizon $(1 + O(\varepsilon))T$ satisfying demands D at cost at most C; or the information that $T < T^*$.

1. set $\Delta := \varepsilon^2 T/n$ and $T' := \lceil (1+\varepsilon)^3 T/\Delta \rceil \Delta$;
2. compute static flow x in $\mathcal{N}^{T'}/\Delta$ satisfying demands $(1+\varepsilon)D$ at cost at most $(1+\varepsilon)C$; if no such flow exists, then stop and output "$T < T^*$";
3. transform x into a flow over time f in $\mathcal{N}$ with time horizon $(1+\varepsilon)T'$ satisfying demands D at cost at most C.

FIG. 9. *The core component of an FPTAS.*

that are within a constant multiple of each other. This requires $\log T^*$ calls of Algorithm FPTAS-Core.[9] Based on these upper and lower bounds, an estimate T with $T^* \le T \le (1+O(\varepsilon))\,T^*$ can be obtained by a geometric mean binary search[10] with $O(\log(1/\varepsilon))$ calls of Algorithm FPTAS-Core. In particular, the last call of Algorithm FPTAS-Core then returns a solution to the quickest flow problem with time horizon at most $(1+\varepsilon)\,T'$. By definition of T' and Δ, this makspan is bounded from above by $(1+\varepsilon)^4 T + (1+\varepsilon)\,\Delta$ and thus in $(1+O(\varepsilon))T^*$. The correctness of Algorithm FPTAS-Core follows from the next proposition.

Proposition 4.7. *Let $T \ge T^*$, $\Delta := \varepsilon^2 T/n$, and $T' := \lceil (1+\varepsilon)^3 T/\Delta \rceil\,\Delta$.*

(a) *There exists a static flow x in the Δ-condensed time-expanded network $\mathcal{N}^{T'}/\Delta$ satisfying demands $(1+\varepsilon)D$ at cost at most $(1+\varepsilon)C$.*

(b) *Given a flow x as in* (a)*, one can compute a flow over time f in $\mathcal{N}$ with time horizon at most $(1+\varepsilon)\,T'$ satisfying demands D at cost at most C.*

We start by proving the following lemma.

Lemma 4.8. *For any $\delta \ge 1$ and any $T \ge T^*$, there exists a flow over time $\tilde{f}$ with time horizon δT satisfying supplies and demands δD at cost at most δC.*

Proof. Consider an optimal solution f^* to the quickest flow problem. That is, f^* is a flow over time with time horizon $T^* \le T$ satisfying supplies and demands D at cost at most C. By rescaling time, we can assume without loss of generality that T and all transit times are integral. Let x^* be the static flow in the T-time-expanded network which corresponds to f^*. Consider a modified instance where all transit times of arcs are increased by a factor of δ. Then, the δ-condensed time-expanded network of the modified instance with time horizon δT is identical to the time-expanded network $\mathcal{N}^T$ but with arc capacities multiplied by δ. In particular, δx^* defines a feasible flow over time with time horizon δT and cost $\delta c(f^*) \le \delta C$ satisfying demands and supplies δD for the modified instance. Since transit times in the original instance are smaller, it can be seen as a relaxation of the modified instance. This yields the existence of $\tilde{f}$ and concludes the proof. ∎

In the following we denote the rounded transit time function by τ'; i.e., $\tau'_e := \lceil \tau_e/\Delta \rceil \Delta$ and $0 \le \tau'_e - \tau_e < \Delta$ for all $e \in A$.

Proof of Proposition 4.7(a). In order to prove the existence of a static flow x in $\mathcal{N}^{T'}/\Delta$ with the claimed properties, Lemma 4.1 implies that it suffices to show the following: In the network $\mathcal{N}$ with transit times τ' there exists a flow over time $\bar{f}$ with time horizon T' satisfying demands $(1+\varepsilon)D$ at cost at most $(1+\varepsilon)C$.

By Corollary 4.6 there exists a flow over time $\tilde{f}$ as in Lemma 4.8 with $\delta = (1+\varepsilon)^2$ that in addition sends flow only on simple paths and that never stores flow at intermediate nodes. This means that $\tilde{f}$ can be written as a sum of path-flows over time $\tilde{f}_P$, $P \in \mathcal{P}$: Consider an arbitrary arc $e = (v,w) \in A$. The total flow into arc e at time θ is

$$\tilde{f}_e(\theta) \;=\; \sum_{P \in \mathcal{P} \,:\, e \in P} \tilde{f}_P\big(\theta - \tau(P,e)\big), \tag{8}$$

where $\tau(P,e)$ denotes the length of the subpath of P which is obtained by removing arc e and all its successors.

[9] Alternatively, the constant factor approximation algorithm for the quickest transshipment problem with bounded cost presented in section 3 yields a lower bound L and an upper bound U on T^* with $U \in O(L)$.

[10] For details on this variant of binary search we refer to [21].

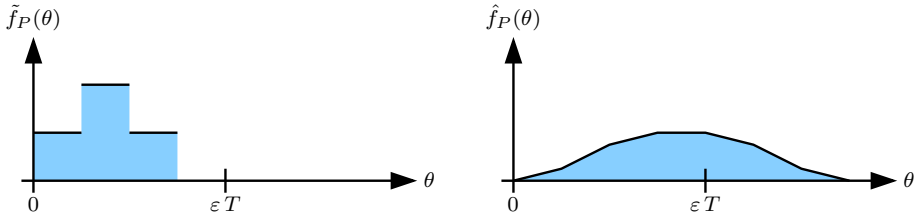


FIG. 10. *The "smoothed" path flow over time $\hat{f}_P$ in comparison to the original flow over time f'_P sent into path P.*

From $\tilde{f}$ we obtain a "smoothed" flow over time $(\hat{f}_P)_{P\in\mathcal{P}}$ which has a time horizon of $(1+\varepsilon)^2T + \varepsilon T$ by defining

$$\hat{f}_P(\theta) \;:=\; \frac{1}{\varepsilon T}\int_{\theta-\varepsilon T}^{\theta} \tilde{f}_P(\xi)\,d\xi \tag{9}$$

for $\theta \in [0, (1+\varepsilon)^2T + \varepsilon T)$ and $P \in \mathcal{P}$. An illustrative example is given in Figure 10. It is easy to check that $\hat{f}$ obeys capacity constraints and the total amount of flow sent on a path $P \in \mathcal{P}$ is the same in $\tilde{f}$ and $\hat{f}$. In particular, $c(\hat{f}) = c(\tilde{f}) \le (1+\varepsilon)^2 C$, and $\hat{f}$ satisfies demands $(1+\varepsilon)^2 D$.

Notice that $(\hat{f}_P)_{P\in\mathcal{P}}$ still describes a (not necessarily feasible) flow over time in $(\mathcal{N}, \tau')$. Since every path $P \in \mathcal{P}$ is simple, it contains at most $n-1$ arcs; therefore, $0 \le \tau'(P) - \tau(P) \le \varepsilon^2 T$, and

$$0 \;\le\; \tau'(P,e) - \tau(P,e) \;\le\; \varepsilon^2 T \tag{10}$$

for all $e \in P$. Thus, if we interpret $\hat{f}$ as a flow over time in $(\mathcal{N}, \tau')$, we get, for all $e \in A$ and $\theta \in [0, (1+\varepsilon)^2T + \varepsilon T + \varepsilon^2 T)$,

$$\begin{aligned}
\hat{f}_e(\theta) &\overset{(8)}{=} \sum_{P\in\mathcal{P}\,:\,e\in P} \hat{f}_P(\theta - \tau'(P,e)) \\
&\overset{(9)}{=} \frac{1}{\varepsilon T}\sum_{P\in\mathcal{P}\,:\,e\in P} \int_{\theta-\tau'(P,e)-\varepsilon T}^{\theta-\tau'(P,e)} \tilde{f}_P(\xi)\;d\xi \\
&\overset{(10)}{\le} \frac{1}{\varepsilon T}\sum_{P\in\mathcal{P}\,:\,e\in P} \int_{\theta-\tau(P,e)-\varepsilon^2 T-\varepsilon T}^{\theta-\tau(P,e)} \tilde{f}_P(\xi)\;d\xi \\
&= \frac{1}{\varepsilon T}\int_{\theta-\varepsilon^2 T-\varepsilon T}^{\theta} \sum_{P\in\mathcal{P}\,:\,e\in P} \tilde{f}_P\big(\xi - \tau(P,e)\big)\;d\xi \\
&\overset{(8)}{=} \frac{1}{\varepsilon T}\int_{\theta-\varepsilon^2 T-\varepsilon T}^{\theta} \tilde{f}_e(\xi)\;d\xi \\
&\le \frac{\varepsilon^2 T + \varepsilon T}{\varepsilon T}\,u_e \;=\; (1+\varepsilon)\,u_e.
\end{aligned} \tag{11}$$

(Above, a number over a relation indicates that the corresponding equation is used to obtain the right-hand side. In (11), we use (10) and the fact that $\tilde{f}_P(\xi) \ge 0$ for all ξ.) Thus, by dividing $\hat{f}$ by $1+\varepsilon$, we establish the existence of a feasible flow over time—namely $\bar{f} := \hat{f}/(1+\varepsilon)$—in $(\mathcal{N}, \tau')$. The time horizon of $\bar{f}$ is at

most $(1+\varepsilon)^2T+\varepsilon T+\varepsilon^2 T \le (1+\varepsilon)^3 T \le T'$, its cost is $c(\hat{f})/(1+\varepsilon) \le (1+\varepsilon)\,C$, and it satisfies demands $(1+\varepsilon)\,D$. □

We now turn to the second part of Proposition 4.7. The static flow x in $\mathcal{N}^{T'}/\Delta$ naturally induces a flow over time f' in $(\mathcal{N},\tau')$ with time horizon T'. If we choose to allow storage of flow at intermediate nodes, we may simplify the algorithm by finding a flow x satisfying demands D at cost C in step 2. Then the corresponding flow over time can be simulated in the network with transit times τ by holding flow at nodes.

If, however, storage of flow at intermediate nodes is not allowed, deriving the final flow over time f in $(\mathcal{N},\tau)$ from the static flow x computed in step 2 is a nontrivial task. The static flow x corresponds to a flow over time f' in $(\mathcal{N},\tau')$ with time horizon T' that satisfies demands $(1+\varepsilon)D$ at cost at most $(1+\varepsilon)C$. Since x lives in a (condensed) time-expanded network without holdover arcs at intermediate nodes, f' never stores flow at intermediate nodes in $(\mathcal{N},\tau')$. Moreover, using the same argument as in Corollary 4.6, we can assume that x is such that f' sends flow only on simple paths in the underlying network $\mathcal{N}$. This means that f' can be written as a sum of path-flows over time f'_P, $P \in \mathcal{P}$.

In the following lemma we show that a path-decomposition of the static flow x can be turned into a path-decomposition of the flow over time f' which features a simple structure. Notice that the number of arcs of the condensed time-expanded network $\mathcal{N}^{T'}/\Delta$ is in $O(mn/\varepsilon^2)$.

LEMMA 4.9. *A path-decomposition of x into flows on r paths in $\mathcal{N}^{T'}/\Delta$ can be turned into a path-decomposition $(f'_P)_{P\in\mathcal{P}}$ of f' on r paths such that the time interval $[0,T')$ can be partitioned into $T'/\Delta \in O(n/\varepsilon^2)$ subintervals where f'_P is constant for all $P \in \mathcal{P}$.*

Proof. Any path P used in a path-decomposition of x connects a source to a sink in $\mathcal{N}^{T'}/\Delta$. Each path P consists of a sequence of holdover arcs at the source, followed by a path of copies of arcs in $\mathcal{N}$, followed by a sequence of holdover arcs at the sink. The static path-flow in $\mathcal{N}^{T'}/\Delta$ of value x_P along P thus induces a path-flow over time $f'_{P'} : [0,T') \to \mathbb{R}^+$ along path P' in $\mathcal{N}$ such that the flow function $f'_{P'}$ is 0 except for an interval of length Δ where it is equal to x_P/Δ. Since there can be several paths in $\mathcal{N}^{T'}/\Delta$ that correspond to the same path P' in $\mathcal{N}$, the flow function on path P' in the final decomposition of f' is a piecewise constant function where the number of intervals with constant flow value is bounded by the number of time layers of $\mathcal{N}^{T'}/\Delta$ which is equal to T'/Δ. □

We are now ready to prove the second part of Proposition 4.7.

Proof of Proposition 4.7(b). Given x, we first derive the corresponding flow over time f' in $(\mathcal{N},\tau')$ with a path-decomposition $(f'_P)_{P\in\mathcal{P}}$ as in Lemma 4.9. Similar to the proof of Proposition 4.7(a), we consider a "smoothed" flow over time $\check{f}$ in $(\mathcal{N},\tau')$ defined by

$$\check{f}_P(\theta) \;:=\; \frac{1}{\varepsilon\,T'} \int_{\theta-\varepsilon\,T'}^{\theta} f'_P(\xi)\,d\xi \tag{12}$$

for $\theta \in [0,(1+\varepsilon)\,T')$ and $P \in \mathcal{P}$. The flow over time $(\check{f}_P)_{P\in\mathcal{P}}$ can be interpreted as a (not necessarily feasible) flow over time in $(\mathcal{N},\tau)$ with time horizon $(1+\varepsilon)T'$ satisfying demands $(1+\varepsilon)\,D$ at cost at most $(1+\varepsilon)\,C$. Moreover, by using essentially the same arguments as for $\hat{f}$ in the proof of Proposition 4.7(a), we get, for all $e \in A$

and $\theta \in [0, (1+\varepsilon)T')$,

$$
\begin{aligned}
\check{f}_e(\theta) &= \sum_{P\in\mathcal{P}\,:\,e\in P} \check{f}_P(\theta - \tau(P,e)) \\
&\overset{(12)}{=} \frac{1}{\varepsilon\,T'} \sum_{P\in\mathcal{P}\,:\,e\in P} \int_{\theta-\tau(P,e)-\varepsilon\,T'}^{\theta-\tau(P,e)} f'_P(\xi)\; d\xi \\
&\overset{(10)}{\leq} \frac{1}{\varepsilon\,T'} \sum_{P\in\mathcal{P}\,:\,e\in P} \int_{\theta-\tau'(P,e)-\varepsilon\,T'}^{\theta-\tau'(P,e)+\varepsilon^2 T'} f'_P(\xi)\; d\xi \qquad (13)\\
&= \frac{1}{\varepsilon\,T'} \int_{\theta-\varepsilon\,T'}^{\theta+\varepsilon^2 T'} \sum_{P\in\mathcal{P}\,:\,e\in P} f'_P\big(\xi - \tau'(P,e)\big)\; d\xi \\
&= \frac{1}{\varepsilon\,T'} \int_{\theta-\varepsilon\,T'}^{\theta+\varepsilon^2 T'} f'_e(\xi)\; d\xi \\
&\leq \frac{\varepsilon^2\,T' + \varepsilon\,T'}{\varepsilon\,T'}\,u_e \;=\; (1+\varepsilon)\,u_e.
\end{aligned}
$$

(Above, a number over a relation indicates that the corresponding equation is used to obtain the right-hand side.) Thus, by dividing $\check{f}$ by $1+\varepsilon$, we get the desired flow over time f in $(\mathcal{N}, \tau)$ with time horizon $(1+\varepsilon)\,T'$ satisfying demands D at cost at most C.

It remains to discuss the issue of how to actually compute f in step 3 of Algorithm FPTAS-Core. According to Lemma 4.9, a path-decomposition of x yields a path-decomposition of the corresponding flow over time f' such that f'_P is piecewise constant for all $P \in \mathcal{P}$ and has at most $O(n/\varepsilon^2)$ breakpoints. Since by definition

$$
f_P(\theta) \;=\; \frac{1}{1+\varepsilon}\frac{1}{\varepsilon\,T'} \int_{\theta-\varepsilon\,T'}^{\theta} f'_P(\xi)\,d\xi,
$$

by Lemma 4.9, the functions f_P, $P \in \mathcal{P}$, are piecewise linear (see Figure 10) and can be efficiently computed. This concludes the proof. □

It remains to discuss the running time of Algorithm FPTAS-Core. The condensed time-expanded network $\mathcal{N}^{T'}/\Delta$ (without holdover arcs) contains $O(n^2/\varepsilon^2)$ nodes and $O(mn/\varepsilon^2)$ arcs. Thus the static flow x' in step 2 can be computed in polynomial time. Since step 3 of Algorithm FPTAS-Core also takes polynomial time (see the proof of Proposition 4.7(b)), the overall running time of the algorithm is polynomial in the input size.

Theorem 4.10. *For an arbitrary $\varepsilon > 0$, a $(1+\varepsilon)$-approximate solution to the quickest transshipment problem with bounded cost can be obtained from $O(\log(1/\varepsilon))$ static min-cost flow computations in a condensed time-expanded network containing $O(n^2/\varepsilon^2)$ nodes and $O(mn/\varepsilon^2)$ arcs (without holdover arcs). In particular, this solution does not use intermediate node storage.*

For the case of the quickest transshipment problem without costs, the min-cost flow computations in the condensed time-expanded network can be replaced by max-flow computations.

4.6. Quickest multicommodity flows. The result of Theorem 4.10 can be generalized to the quickest multicommodity flow problem with bounded cost. Figure 2 shows that the optimal solution to the quickest multicommodity flow problem may require the use of storage of flow at intermediate nodes. On the other hand, if storing

at intermediate nodes is not allowed, then the optimal solution may contain nonsimple flow paths. The analysis in (11) and (13) relies on the fact that one can restrict to simple flow paths, since it uses (10). Indeed, it is shown in [19] that, unless P=NP, there is no FPTAS for the quickest multicommodity flow problem when intermediate node storage is prohibited and flow may only be sent on simple paths. If, however, intermediate node storage is allowed, then there exists an optimal solution that uses only simple flow paths: Instead of flow traveling around a cycle, it can simply wait at the start node of the cycle. In this case, the approach described in section 4.5 can be modified as follows.

A flow over time f which stores flow at intermediate nodes can no longer be decomposed into path-flows over time as described in (8). In order to handle the setting with storage of flow at intermediate nodes in a path-based model, we introduce the following notation: A *path with delays* P^δ is given by a path P in $\mathcal{N}$ consisting of nodes $(v_0, v_1, \dots, v_p)$ and a vector of nonnegative delays $\delta = (\delta_1, \dots, \delta_p)$. The value δ_i, $i = 1, \dots, p$, specifies the amount of time that flow is stored at node v_i before it continues its journey towards node v_{i+1} in a flow over time on P^δ. Thus, flow entering P^δ at time θ enters arc $e = (v_\ell, v_{\ell+1})$, $\ell = 0, \dots, p-1$, at time $\theta + \tau(P^\delta, e)$ with $\tau(P^\delta, e) := \sum_{j=1}^{\ell} (\tau_{(v_{j-1}, v_j)} + \delta_j)$.

Since in a given flow over time f with time horizon T every infinitesimal unit of flow describes a path with delays in $\mathcal{N}$, the flow over time f can be decomposed into (possibly infinitely many) flows over time f_{P^δ} on paths with delays P^δ. In this setting, (8) is replaced by

$$f_e(\theta) \;=\; \sum_{P^\delta \,:\, e \in P} f_{P^\delta}\big(\theta - \tau(P^\delta, e)\big) \tag{14}$$

for all $e \in A$. If f is given by a corresponding static flow x in a (condensed) time-expanded network with holdover arcs, then there exists a decomposition of f into flows over time on paths with delays in $\mathcal{N}$ whose number is bounded by the number of arcs in the (condensed) time-expanded network: Consider an arbitrary path-decomposition of x, and notice that any path in the (condensed) time-expanded network with holdover arcs yields a path with delays in $\mathcal{N}$.

Summarizing, a straightforward modification of the analysis in section 4.5 (i.e., replacing (8) by (14)) yields the following result.

THEOREM 4.11. *Consider an instance of the quickest multicommodity transshipment problem with bounded cost and intermediate node storage. For any $\varepsilon > 0$, a $(1+\varepsilon)$-approximate solution can be found by $O(\log(1/\varepsilon))$ static multicommodity flow computations with bounded cost in a condensed time-expanded network with $O(n^2/\varepsilon^2)$ nodes and $O(mn/\varepsilon^2)$ arcs (including holdover arcs).*

5. Earliest arrival flows. In this section, we address the multiple source, earliest arrival flow problem: Given a set of sources S with supplies $D_v > 0$ for $v \in S$ and a single sink t, send the supplies to the sink so that the amount of supplies arriving at the sink by time θ is the maximum possible for all $\theta \geq 0$. We show how the result from section 4 can be generalized to this problem.

We use the following notion of approximation for the problem of computing an earliest arrival flow. A flow over time is an α-approximate solution to this problem if, for all $d' \leq d := \sum_{v \in S} D_v$, the earliest point in time when d' units have arrived at the sink is within a factor of α of the earliest possible time. An algorithm which computes such a flow over time in polynomial time is called an α-approximation algorithm. In order to stress the property that the performance guarantee α is achieved *for all* $d' \leq d$, we

say that such an algorithm has *universal* performance guarantee α. For the earliest arrival problem, we find a solution with universal performance guarantee $(1+\varepsilon)$.

In section 5.2 we show that a unit-interval discretization may not be sufficiently fine to achieve this guarantee, and we describe how to determine a good initial discretization. This discretization may be too large, and it will be necessary to condense it. However, a uniformly crude discretization will not work, so in section 5.3, we introduce a general framework for nonuniform condensed time-expanded networks and prove some useful properties. Finally, in section 5.4 we derive a polynomial-time algorithm that yields a univeral performance guarantee for the earliest arrival flow problem by using a nonuniform, condensed time-expanded network with intervals of geometrically increasing size.

5.1. Previous work. In the discrete time model, a universally maximal s-t-flow over time can be computed in the time-expanded network by using lexicographically maximal flows introduced by Minieka [28]. A *lexicographically maximal flow* is defined in a static network with multiple sources and/or sinks. There is a strict ordering on the sources and sinks, e.g., $\{\nu_1, \nu_2, \ldots, \nu_k\}$, where ν_i is used here to denote either a source or a sink. A lexicographically maximal flow is a flow that simultaneously maximizes the flow leaving each ordered subset of sources and sinks $\{\nu_1, \nu_2, \ldots, \nu_i\}$, $i = 1, \ldots, k$. In the discrete time model, a universally maximal s-t-flow over time with time horizon T is a lexicographically maximal flow in the time-expanded graph with ordering of sources and sinks as

$$s_{T-1}, s_{T-2}, \ldots, s_1, s_0, t_{T-1}, t_{T-2}, \ldots, t_1, t_0.$$

However, due to the exponential size of the time-expanded network, this insight does not lead to an efficient algorithm for the problem. As mentioned above, the algorithms of Wilkinson [35] and Minieka [28] are based on the successive shortest path algorithm. These also are not efficient algorithms for computing universally maximal dynamic flows since the successive shortest path algorithm requires an exponential number of iterations in the worst case; see, e.g., Zadeh [36].

While the results of Wilkinson and Minieka were originally derived for the discrete time model, they also hold for the continuous time model. In this setting, the existence of universally maximal dynamic flows was first observed by Philpott [32]. Fleischer and Tardos [11] show how the algorithms for the discrete time model mentioned above can be carried over to the continuous time setting.

In the continuous time model, an equivalent problem is the *universally quickest flow problem* which asks for a flow over time of value d such that the earliest point in time when d' units have arrived at the sink is simultaneously minimized for all $d' \leq d$ and the earliest point in time when d' units have left the source is simultaneously maximized for all $d' \leq d$.

Hoppe and Tardos [23] compute a single-source single-sink universally maximal dynamic flow where the amount of flow is approximately optimal at any moment of time. Our Lemma 4.8 implies that this algorithm also achieves a universal guarantee.

An earliest arrival flow also exists for the case of multiple sources and a single sink [34]. In the discrete time model, such a flow over time can again be found by a lexicographically maximal flow computation in the time-expanded network. On the other hand, Fleischer [9] presents an instance with two sources and two sinks for which an earliest arrival flow does not exist.

Nonuniform time-expanded networks have been used previously to obtain exact algorithms for the quickest transshipment problem in the setting of zero transit

times [9]. Partitioning time into intervals of geometrically increasing size has been used previously in conjunction with dynamic programming to derive approximation algorithms for problems in the area of machine scheduling [20, 4, 5, 1].

5.2. A sufficiently fine discretization of time. Unfortunately, a lexicographically maximal flow in a time-expanded network does not necessarily yield a universal performance guarantee for the quickest flow problem in the continuous time model. Although we can interpret a static flow in a time-expanded network as a continuous flow over time (see proof of Lemma 4.1), in doing so, all the solutions we get are such that the rate of flow arriving at the sink is constant (i.e., averaged) within each discrete time interval. While this effect is negligible for late intervals in time, as the following example shows, it could be significant in the first time intervals.

Example 1. The network is a single arc from source s to sink t with capacity 2 and transit time 0. There is a unit of supply at s and a unit of demand at t. A universally quickest flow sends flow from s to t at rate 2 during the interval $[0, \frac{1}{2})$ so that flow arrives at t at rate 2 in the interval $[0, \frac{1}{2})$. Averaging the flow over unit intervals yields a flow that arrives at t at rate 1 in the interval $[0, 1)$. Thus this flow has universal performance guarantee of at most 2.

The problem illustrated by this example can be resolved as follows. For an arbitrary $\varepsilon > 0$ with $1/\varepsilon$ integral, we discretize time into intervals of size ε. Then, averaging flow within each such interval delays flow that arrives at the sink after time 1 by at most a factor of $1 + \varepsilon$. It thus remains to take care of what happens until time 1.

Notice that flow arriving at the sink before time 1 can use only arcs with transit time 0. In particular, an earliest arrival flow until time 1 can be computed by restricting attention to the subnetwork consisting of these arcs. Hajek and Ogier [18] describe an algorithm that finds an earliest arrival flow in a network with a single sink and zero transit times using $O(n)$ maximum flow computations. Fleischer [9] gives an improved algorithm that solves the problem in the same asymptotic time as one maximum flow computation. Moreover, the analysis of this algorithm shows that the function describing the optimal rate of inflow into the sink is piecewise constant and has at most k breakpoints $\theta_1, \ldots, \theta_k$, where k is the number of sources. These breakpoints are independent of the discretization after time 1 and can be computed without this information in polynomial time. They can then be used to determine the appropriate discretization in $[0, 1)$. Thus, a sufficiently fine discretization of time guarantees that an earliest arrival flow until time 1 can be computed in the corresponding time-expanded network. In the example given above, a discretization of time into intervals of size $1/2$ is sufficient. Together with the observation stated above, this yields the following result.

LEMMA 5.1. *Consider an instance of the earliest arrival problem with multiple sources with supply vector D and a single sink, and let $\theta_1, \ldots, \theta_k$ be defined as above. Then, a $(1 + \varepsilon)$-approximate earliest arrival flow can be obtained from a lexicographically maximal flow computation in the time-expanded network $\mathcal{N}^T/\Delta$, where Δ is chosen such that $\theta_1, \ldots, \theta_k$ and ε are multiples of Δ.*

By discussion in [9] (specifically, Theorem 3.6 and Theorem 4.1), we can bound Δ from below by $(m \sum_{e \in A} u_e)^{-k}$. Thus, the size of the resulting time-expanded network $\mathcal{N}^T/\Delta$ is pseudopolynomial in the input size of the problem. Since time can be rescaled by a factor of $1/\Delta$, we can and will assume without loss of generality that $\Delta = 1$ and thus $\mathcal{N}^T/\Delta = \mathcal{N}^T$. In the next subsection we show how $\mathcal{N}^T$ can be turned into a network of polynomial size while only losing another factor of $1 + \varepsilon$ in the performance guarantee of the resulting polynomial-time algorithm.

5.3. Nonuniform condensed time-expanded networks. As we mentioned above, the size of the time-expanded network $\mathcal{N}^T$ is only pseudopolynomial in the input size. In contrast to the situation for the quickest flow problem discussed in section 4, using a uniformly rough discretization can lead to a much worse performance guarantee for the earliest arrival flow problem. Instead we will use a nonuniform discretization. In this section we describe a general framework for nonuniform time-expanded networks and establish, where possible, the appropriate generalizations of properties of uniform time-expanded networks.

Overloading the notation introduced in section 4.1, for a sorted list $L = \langle \theta_0, \dots, \theta_r \rangle$ with

$$0 \;=\; \theta_0 \;<\; \theta_1 \;<\; \cdots \;<\; \theta_{r-1} \;<\; \theta_r \;=\; T,$$

the *L-time-expanded network* of $\mathcal{N} = (V, A)$ denoted by $\mathcal{N}^L$ is obtained by creating r copies of V, labeled V_0 through V_{r-1}, with the qth copy of node v denoted v_q for $q = 0, \dots, r-1$. For every arc $e = (v, w) \in A$ and for every $q \geq 0$ with $\theta_q + \tau_e \leq \theta_{r-1}$, there is an arc e_q from v_q to $w_{q'_e}$ with

$$q'_e \;:=\; \min\{q'' \mid \theta_q + \tau_e \leq \theta_{q''}\}. \tag{15}$$

When e is clear from context, we use q' instead of q'_e. The capacity of arc e_q is set to $u_e\,(\theta_{q+1} - \theta_q)$. In addition, there is a holdover arc from v_q to v_{q+1} with infinite capacity for all $v \in V$ and $0 \leq q < r-1$. Notice that this definition generalizes the definition of the T-time-expanded network $\mathcal{N}^T$; in particular, $\mathcal{N}^T = \mathcal{N}^L$ for $L = \langle 0, 1, \dots, T \rangle$.

Whenever we consider a static flow in $\mathcal{N}^L$, we implicitly assume that t_{r-1} is its only sink; all flow arriving at t_q for $q < r-1$ is sent to this sink on holdover arcs.

LEMMA 5.2. *Let $L = \langle 0 = \theta_0, \dots, \theta_r \rangle$ with $\theta_q - \theta_{q-1} \leq \theta_{q+1} - \theta_q$ for all $q = 1$ to $r-1$. Then any static flow in $\mathcal{N}^L$ corresponds to a flow over time in $\mathcal{N}$ such that the amount of flow reaching t in $\mathcal{N}$ by time θ_{q+1}, $q = 0, \dots, r-1$, is the same as the amount of flow reaching node t_q in $\mathcal{N}^L$.*

Proof. Let x be a static flow in $\mathcal{N}^L$. We interpret x as a "generalized" flow over time f' where, in contrast to "standard" flows over time, transit times can now vary over time. Interpret the flow x_{e_q} on arc $e_q = (v_q, w_{q'})$ as flow f'_e sent at the constant rate $x_{e_q}/(\theta_{q+1} - \theta_q)$ into arc $e = (v, w)$ during the time interval $[\theta_q, \theta_{q+1})$ and arriving at node w during the time interval $[\theta_{q'}, \theta_{q'+1})$ at the constant rate $x_{e_q}/(\theta_{q'+1} - \theta_{q'})$. In particular, in the time interval $[\theta_q, \theta_{q+1})$, the transit time on arc e varies between $\theta_{q'} - \theta_q$ (flow entering the arc at time θ_q) and $\theta_{q'+1} - \theta_{q+1}$ (flow entering the arc immediately before time θ_{q+1}) in f'. Thus, since the sizes of the time intervals $[\theta_q, \theta_{q+1})$, $q = 0, \dots, r-1$, are nondecreasing and by choice of q', the transit time on arc e in f' is never smaller than τ_e.

By design, f' obeys flow rate capacity constraints and flow conservation. Moreover, for $q = 0, \dots, r-1$, the amount of flow reaching t in f' by time θ_{q+1} is the same as the amount of flow reaching node t_q in x. Finally, since transit times in f' are always lower bounded by the actual transit times τ_e of arcs $e \in A$ (by (15)), it can easily be interpreted as a regular flow over time f in $\mathcal{N}$ by introducing appropriate waiting times for every infinitesimal flow unit at the head of each arc. ∎

LEMMA 5.3. *Let $L = \langle 0 = \theta_0, \dots, \theta_r \rangle$ with $\theta_q - \theta_{q-1} \leq \theta_{q+1} - \theta_q$ for all $q = 1$ to $r-1$. Let f be any flow over time that completes by time θ_r in the network $\mathcal{N}$ modified so that all transit times are increased by $\Delta := \theta_r - \theta_{r-1}$. Then, f induces a feasible static flow in $\mathcal{N}^L$ of the same value.*

Proof. For all $e \in A$ and $0 \le q \le r-1$, define x_{e_q} to be the total f-flow into e in interval $[\theta_q, \theta_{q+1})$. Since f is feasible, it obeys the capacity constraints; then by construction so does x. By design, the value of x equals the value of f.

To establish flow conservation, consider the path $P = \langle w^0, w^1, \ldots, w^h = t\rangle$ traveled by an infinitesimal unit of flow in f. This infinitesimal unit of flow arrives at w_i at time φ_i and leaves w_i at some time $\xi_i \ge \varphi_i$. We argue that the corresponding unit of flow in x arrives at $w^i_{p_i}$ for some p_i satisfying $\theta_{p_i} \le \varphi_i$. Thus it is available to be sent along P in $\mathcal{N}^L$ on arc $(w^i_{q_i}, w^{i+1}_{q'_i})$, where q_i is the maximum index ℓ satisfying $\theta_\ell \le \xi_i$. This will show that x satisfies flow conservation.

Set $q := q_{i-1}$. By design of x, the infinitesimal unit of flow leaves w^{i-1}_q in $\mathcal{N}^L$ and thus arrives at $w^i_{q'}$, where q' is defined according to (15). By definition of q', we have that

$$\begin{aligned}\theta_{q'} - \theta_q &= \theta_{q'-1} - \theta_q + (\theta_{q'} - \theta_{q'-1}) \\ &< \tau_{(w^{i-1},w^i)} + (\theta_r - \theta_{r-1}) = \tau_{(w^{i-1},w^i)} + \Delta.\end{aligned}$$

This yields $\theta_{q'} < \theta_q + \tau_{(w^{i-1},w^i)} + \Delta \le \xi_{i-1} + \tau_{(w^{i-1},w^i)} + \Delta = \varphi_i \le \xi_i$. (The second inequality follows by definition of $q := q_{i-1} \le \xi_{i-1}$ at the end of the previous paragraph; the subsequent equality follows by definition of φ_i.) This implies that $\theta_{q'} \le \theta_{q_i}$, and thus the flow obeys conservation constraints. ∎

5.4. An approximation scheme. To obtain a $(1+\varepsilon)$-universal guarantee for the earliest arrival flow problem, we use a geometrically increasing time discretization. Let $p := \lceil 2n/\varepsilon^2 \rceil$, and define the list

$$\begin{aligned}L \;:=\; \langle &0, 1, 2, \ldots, 2p, \\ &2(p+1), 2(p+2), \ldots, 4p, \\ &4(p+1), 4(p+2), \ldots, 8p, \\ &\ldots \\ &2^{\ell-1}(p+1), 2^{\ell-1}(p+2), \ldots, 2^\ell p\rangle.\end{aligned}$$

Here $\ell \in \mathbb{N}_0$ is chosen such that the resulting time-expanded network $\mathcal{N}^L$ is large enough to allow for a $(1+O(\varepsilon))$-approximate time horizon. To be more precise, we choose the smallest ℓ such that $2T^* \le 2^\ell p$, where T^* is the time horizon of an optimal flow over time. Notice that the length of the list L is in $O(p \log T^*)$ and hence polynomially bounded in the input size and $1/\varepsilon$. It thus remains to be shown that a lexicographically maximal flow in the corresponding *geometrically condensed time-expanded network* $\mathcal{N}^L$ yields a flow over time with universal performance guarantee $1+O(\varepsilon)$.

LEMMA 5.4. *A lexicographically maximal flow in $\mathcal{N}^L$ induces a flow over time in $\mathcal{N}$ with universal performance guarantee $1+O(\varepsilon)$.*

Proof. Let $T \le T^*$ be the minimal time required to send $d' \le d$ units of flow to the sink. We consider the time-expanded network $\mathcal{N}^{L'}$ defined by the sublist $L' := \langle \theta_0, \ldots, \theta_{r'}\rangle$ of L with $r' := \min\{q \mid \theta_q \ge (1+\varepsilon)^2 T\}$. In other words, $\mathcal{N}^{L'}$ is obtained from $\mathcal{N}^L$ by removing all time layers V_q with $q \ge r'$ and all incident arcs. We show below that there is a flow in $\mathcal{N}^{L'}$ of value d'. Then, since in a lexicographically maximum flow in $\mathcal{N}^L$ at least d' units of flow reach $t_{r'-1}$, the corresponding flow over time in $\mathcal{N}$ sends at least d' to t by time $\theta_{r'}$ by Lemma 5.2. Hence this flow yields a universal performance guarantee of $1+O(\varepsilon)$.

Let $\Delta' = \theta_{r'} - \theta_{r'-1}$. By choice of p and since we can assume that $\theta_{r'} \leq 2T$, we have that

$$\Delta' \;=\; \theta_{r'} - \theta_{r'-1} \;\leq\; \theta_{r'}/p \;\leq\; \varepsilon^2 T/n. \tag{16}$$

Since there is flow of value d' in $\mathcal{N}$ by time T, Lemma 4.8 implies that there is flow of value $d'(1+\varepsilon)$ in $\mathcal{N}$ by time $T(1+\varepsilon)$. Then, the same argument as in the proof of Proposition 4.7(a) (together with (16)) implies that there is flow of value d' that completes by time $(1+\varepsilon)T + \varepsilon\, T + \varepsilon^2\, T = (1+\varepsilon)^2 T$ in the network $\mathcal{N}$ with all transit times increased by Δ'. By Lemma 5.3, this implies that there is flow of value d' in $\mathcal{N}^{L'}$. □

We have now established the main result of this section.

THEOREM 5.5. *There exists an FPTAS for the problem of computing an earliest arrival flow in a network with multiple sources and a single sink.*

Acknowledgments. We thank Dan Stratila for his helpful comments on an earlier version of this paper. We also thank the two anonymous referees for their numerous valuable comments that helped to improve the presentation of the paper.

REFERENCES

[1] F. AFRATI, E. BAMPIS, C. CHEKURI, D. KARGER, C. KENYON, S. KHANNA, I. MILIS, M. QUEYRANNE, M. SKUTELLA, C. STEIN, AND M. SVIRIDENKO, *Approximation schemes for minimizing average weighted completion time with release dates*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, New York City, NY, 1999, pp. 32–43.

[2] J. E. ARONSON, *A survey of dynamic network flows*, Ann. Oper. Res., 20 (1989), pp. 1–66.

[3] R. E. BURKARD, K. DLASKA, AND B. KLINZ, *The quickest flow problem*, ZOR Methods Models Oper. Res., 37 (1993), pp. 31–58.

[4] S. CHAKRABARTI, C. PHILLIPS, A. S. SCHULZ, D. B. SHMOYS, C. STEIN, AND J. WEIN, *Improved scheduling algorithms for minsum criteria*, in Automata, Languages and Programming, Lecture Notes in Comput. Sci. 1099, F. Meyer auf der Heide and B. Monien, eds., Springer, Berlin, 1996, pp. 646–657.

[5] F. A. CHUDAK AND D. B. SHMOYS, *Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds*, J. Algorithms, 30 (1999), pp. 323–343.

[6] L. FLEISCHER AND M. SKUTELLA, *The quickest multicommodity flow problem*, in Integer Programming and Combinatorial Optimization, Lecture Notes in Comput. Sci. 2337, W. J. Cook and A. S. Schulz, eds., Springer, Berlin, 2002, pp. 36–53.

[7] L. FLEISCHER AND M. SKUTELLA, *Minimum cost flows over time without intermediate storage*, in Proceedings of the 14th Annual ACM–SIAM Symposium on Discrete Algorithms, Baltimore, MD, 2003, pp. 66–75.

[8] L. K. FLEISCHER, *Approximating fractional multicommodity flows independent of the number of commodities*, SIAM J. Discrete Math., 13 (2000), pp. 505–520.

[9] L. K. FLEISCHER, *Faster algorithms for the quickest transshipment problem*, SIAM J. Optim., 12 (2001), pp. 18–35.

[10] L. K. FLEISCHER, *Universally maximum flow with piece-wise constant capacity functions*, Networks, 38 (2001), pp. 1–11.

[11] L. K. FLEISCHER AND É. TARDOS, *Efficient continuous-time dynamic network flow algorithms*, Oper. Res. Lett., 23 (1998), pp. 71–80.

[12] L. R. FORD AND D. R. FULKERSON, *Constructing maximal dynamic flows from static flows*, Oper. Res., 6 (1958), pp. 419–433.

[13] L. R. FORD AND D. R. FULKERSON, *Flows in Networks*, Princeton University Press, Princeton, NJ, 1962.

[14] D. GALE, *Transient flows in networks*, Michigan Math. J., 6 (1959), pp. 59–63.

[15] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[16] N. Garg and J. Könemann, *Faster and simpler algorithms for multicommodity flow and other fractional packing problems*, in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, Palo Alto, CA, 1998, pp. 300–309.
[17] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Algorithms Combin., Springer, Berlin, 1988.
[18] B. Hajek and R. G. Ogier, *Optimal dynamic routing in communication networks with continuous traffic*, Networks, 14 (1984), pp. 457–487.
[19] A. Hall, S. Hippler, and M. Skutella, *Multicommodity flows over time: Efficient algorithms and complexity*, Theoret. Comput. Sci., to appear.
[20] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein, *Scheduling to minimize average completion time: Off-line and on-line approximation algorithms*, Math. Oper. Res., 22 (1997), pp. 513–544.
[21] R. Hassin, *Approximation schemes for the restricted shortest path problem*, Math. Oper. Res., 17 (1992), pp. 36–42.
[22] B. Hoppe, *Efficient Dynamic Network Flow Algorithms*, Ph.D. thesis, Cornell University, Ithaca, NY, 1995.
[23] B. Hoppe and É. Tardos, *Polynomial time algorithms for some evacuation problems*, in Proceedings of the 5th Annual ACM–SIAM Symposium on Discrete Algorithms, Arlington, VA, 1994, pp. 433–441.
[24] B. Hoppe and É. Tardos, *The quickest transshipment problem*, Math. Oper. Res., 25 (2000), pp. 36–62.
[25] B. Klinz and G. J. Woeginger, *Minimum-cost dynamic flows: The series-parallel case*, Networks, 43 (2004), pp. 153–162.
[26] D. H. Lorenz and D. Raz, *A simple efficient approximation scheme for the restricted shortest path problem*, Oper. Res. Lett., 28 (2001), pp. 213–219.
[27] N. Megiddo, *Combinatorial optimization with rational objective functions*, Math. Oper. Res., 4 (1979), pp. 414–424.
[28] E. Minieka, *Maximal, lexicographic, and dynamic network flows*, Oper. Res., 21 (1973), pp. 517–527.
[29] R. G. Ogier, *Minimum-delay routing in continuous-time dynamic networks with piecewise-constant capacities*, Networks, 18 (1988), pp. 303–318.
[30] J. B. Orlin, *Minimum convex cost dynamic network flows*, Math. Oper. Res., 9 (1984), pp. 190–207.
[31] C. A. Phillips, *The network inhibition problem*, in Proceedings of the 25th Annual ACM Symposium on the Theory of Computing, San Diego, CA, 1993, pp. 776–785.
[32] A. B. Philpott, *Continuous-time flows in networks*, Math. Oper. Res., 15 (1990), pp. 640–661.
[33] W. B. Powell, P. Jaillet, and A. Odoni, *Stochastic and dynamic networks and routing*, in Network Routing, Handbooks Oper. Res. Management Sci. 8, M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, eds., North–Holland, Amsterdam, 1995, Chap. 3, pp. 141–295.
[34] D. Richardson and É. Tardos, *private communication*, 2002.
[35] W. L. Wilkinson, *An algorithm for universal maximal dynamic flows in a network*, Oper. Res., 19 (1971), pp. 1602–1612.
[36] N. Zadeh, *A bad network problem for the simplex method and other minimum cost flow algorithms*, Math. Program., 5 (1973), pp. 255–266.

SIAM J. COMPUT.
Vol. 36, No. 6, pp. 1631–1647

A CONSTANT-FACTOR APPROXIMATION ALGORITHM FOR OPTIMAL 1.5D TERRAIN GUARDING*

BOAZ BEN-MOSHE†, MATTHEW J. KATZ‡, AND JOSEPH S. B. MITCHELL§

Abstract. We present the first constant-factor approximation algorithm for a nontrivial instance of the optimal guarding (coverage) problem in polygons. In particular, we give an $O(1)$-approximation algorithm for placing the fewest point guards on a 1.5D terrain, so that every point of the terrain is seen by at least one guard. While polylogarithmic-factor approximations follow from set cover results, our new results exploit the geometric structure of terrains to obtain a substantially improved approximation algorithm.

Key words. geometric optimization, guarding, approximation algorithms

AMS subject classifications. 68W25, 68W40, 05C69

DOI. 10.1137/S0097539704446384

1. Introduction.

For a geometric domain $\mathcal{D}$, the *optimal guarding (coverage) problem* is to determine the smallest number, k^*, of *guards* (e.g., points) that can be placed in $\mathcal{D}$ so that each point of $\mathcal{D}$ is *seen* by at least one guard. The optimal guarding problem is an instance of a set cover problem that is induced by a geometric setting. The many related problems, both combinatorial and algorithmic, fall into the general category known as *art gallery problems*, which have been studied extensively; see, e.g., [12, 14, 16, 17].

We give the first constant-factor approximation algorithm for a nontrivial instance of the optimal guarding problem. All prior approximation bounds were polylogarithmic in n and/or k^*.

The instance we study here is the 1.5D *terrain guarding problem*, in which the input domain is the two-dimensional region above an x-monotone polygonal chain having n vertices. We restrict guards to being placed at points on the terrain; with no restriction on guard placement, a single guard (at a high enough altitude) suffices to see the terrain.

Related work. The classical combinatorial result, the "art gallery theorem," states that $\lfloor n/3 \rfloor$ guards are sufficient, and sometimes necessary, to guard an n-vertex simple polygon [14]. Combinatorial results on the number of guards needed for various forms of guarding on terrains are given in [2].

The optimal guarding problem is known to be NP-hard, even if $\mathcal{D}$ is a simple polygon [15]. Thus, efforts have concentrated on the approximability of optimal

*Received by the editors October 22, 2004; accepted for publication (in revised form) September 2, 2006; published electronically February 26, 2007. An extended abstract of this paper appeared in [1]. The second and third authors are partially supported by grant 2000160 from the U.S.-Israel Binational Science Foundation. The first and second authors are partially supported by the MAGNET program of the Israel Ministry of Industry and Trade (LSRT consortium). The third author also acknowledges support from Honda Fundamental Research Lab, NASA Ames Research (NAG2-1620), the National Science Foundation (CCR-0098172, ACI-0328930), and Metron Aviation.

http://www.siam.org/journals/sicomp/36-6/44638.html

†Department of Computer Science and Mathematics, College of Judea & Samaria, Ariel 44837, Israel (benmo@yosh.ac.il).

‡Department of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel (matya@cs.bgu.ac.il).

§Department of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, NY 11794-3600 (jsbm@ams.sunysb.edu).

guarding problems. Ghosh [10] gave an $O(\log n)$-approximation for optimal coverage of a polygon by vertex guards, based on standard set cover results. Recent interest [6, 11] has focused on methods of efficiently applying the Brönnimann–Goodrich technique [3], which exploits finiteness of VC-dimension. Efrat and Har-Peled [6] obtain an $O(\log k^*)$-approximation algorithm for polygon guarding with vertex guards, using time $O(n(k^*)^2 \log^4 n)$, where k^* is the optimal number of vertex guards. Their technique can be applied to nonvertex guards lying at points of a dense grid, adding to the running time a factor polylogarithmic in the grid density. (No approximation algorithm is known if the guards are completely unrestricted and all of the polygon is to be guarded.) Their results apply also to polygons with holes and to 2.5D terrains, still with polylogarithmic approximation factors. Most recently, Cheong, Efrat, and Har-Peled [5] have shown how to place k guards in order to optimize (approximately) the total area seen by the guards.

Eidenbenz [7], and Eidenbenz, Stamm, and Widmayer [9] have shown that it is APX-hard to solve the optimal guarding problem in simple polygons; thus, there exists an $\epsilon > 0$ such that it is NP-hard to obtain a $(1+\epsilon)$-approximation algorithm. They further show that guarding polygons with holes, as well as guarding 2.5D terrains, is as hard as set cover; these problems do not have polynomial-time algorithms with approximation ratio less than $c \ln n$, for some $c > 0$, unless $P = NP$ (and do not have approximation ratio $\frac{1-\epsilon}{12} \ln n$, for any $\epsilon > 0$, unless $NP \subset TIME(n^{O(\log \log n)})$). These hardness of approximation results are complemented by $O(\log n)$-approximation algorithms based on greedy set cover [8].

For 1.5D terrains, Chen, Estivill-Castro, and Urrutia [4] claim that a modification of the hardness proof of [15] shows that the problem is NP-hard (details are omitted and are still to be verified). In the very restricted setting where all guards are to be placed at a constant altitude (above "sea level," not above the terrain surface), the optimal guarding problem is readily solved in polynomial time (it reduces to a one-dimensional problem on intervals); in fact, it can be solved in linear time [7, 13].

Motivation. The terrain guarding problem arises in optimal placement of antennas for communication networks. We are motivated to study the problem in two dimensions, in an effort to understand better the much more difficult three-dimensional (2.5D) terrain guarding problem (which Eidenbenz shows has no better than a log-approximation). Further, the two-dimensional problem shows up as a subproblem in heuristics for solving the three-dimensional problem, and it arises directly in applications of coverage along a highway, and security lamp and camera placement along walls and streets. In some applications, one needs to cover only a portion of the input domain; our results can be extended to yield an algorithm for covering a subset of the terrain surface optimally.

Our results. Our main result is an efficient $O(1)$-approximation algorithm for the terrain guarding problem in two dimensions. Our results rely on developing several geometric properties of terrains, in order that our algorithm can exploit the special structure.

The main difficulty in proving our main result is in showing that one can obtain a constant-factor approximation to the dominating set problem in graphs that can be realized as the visibility graph of the vertices of a terrain polygon (i.e., to the problem of placing vertex guards on a terrain in order to see all other vertices). It is then relatively straightforward to extend this result to apply to placing arbitrary guards on the terrain in order to guard all of the terrain.

It is worth noting that some natural and simple approaches to the problem do

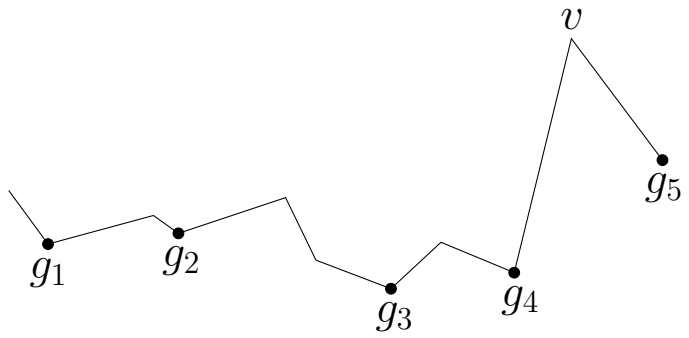


FIG. 1. *None of the guards $g_1, \ldots, g_5$ is redundant, but a single guard at v would suffice.*

not yield $O(1)$-approximations: One can give examples showing a subset of vertices guarding T, without redundant vertices, that is arbitrarily bad with respect to optimal (see Figure 1); similarly, simple sweep approaches that add guards "as needed" can be arbitrarily bad.

We begin with some definitions and basic structural results.

2. Preliminaries. Let T be a terrain of complexity n in the plane; more precisely, let T be an x-monotone polygonal curve specified by n vertices, $v_1, \ldots, v_i = (x_i, y_i), \ldots, v_n$, indexed in x-increasing order. We refer to the *terrain polygon*, P_T, which is the closed region of the plane bounded from below by T and from the sides by the upwards vertical rays emanating from v_1 and v_n, respectively.

Let $p = (p_x, p_y)$ and $q = (q_x, q_y)$ be two points on the polygonal curve T. We say that p *sees* q (and q *sees* p) if and only if the line segment $\overline{pq}$ is contained in P_T. We write $p < q$ if p is to the left of q (i.e., $p_x < q_x$). For any two vertices $u, v \in T$ with $u < v$, the *subterrain* $[u, v]$ of T is simply the portion of T between u and v (including u and v). For a point $p \in T$, let $L(p)$ (resp., $R(p)$) denote the leftmost (resp., rightmost) point on T that sees p. It is easy to see that $L(p)$ and $R(p)$ are necessarily vertices of T.

A *guard* is a point of P_T that we consider to be able to view ("guard" or "cover") other points of P_T. A *terrain guard* is any point on T; a *vertex guard* is a vertex of T.

The *must-guard set*, M, is a subset of P_T that we require to be seen by a set of guards. The *dominating set problem on terrains* (DSPT) is to compute a minimum-cardinality set of vertex guards for $M = \{v_1, \ldots, v_n\}$, the vertex set of T. We spend most of our effort in obtaining an approximation algorithm for the DSPT; then, we describe how this result extends to the case of terrain guards and to the case of $M = T$. (Note that a set of guards that sees $M = T$ necessarily sees all of P_T; however, it is not the case that guarding the boundary of a general simple polygon P implies guarding the interior of P.)

We say that a subterrain $[u, v]$ *can be guarded from the right (resp., left)* if there exists a set of vertex guards to the right of v (resp., to the left of u) that covers $M \cap [u, v]$.

CLAIM 2.1 (order claim). *Let $a < b < c < d$ be four points on T. If a sees c and b sees d, then a sees d.*

Proof. Since a sees c, b lies below the line segment $\overline{ac}$; similarly, c lies below $\overline{bd}$. Thus, $\overline{ac}$ and $\overline{bd}$ cross each other at a point ξ (above T), and T lies below the chain (a, ξ, d), implying that a sees d. ∎

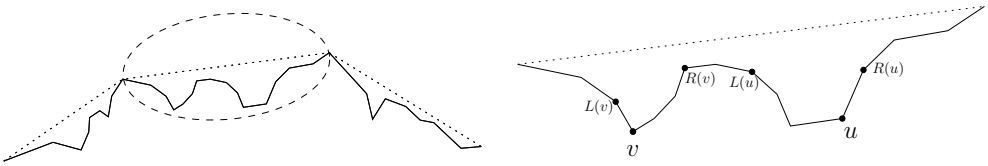


FIG. 2. *The overall structure of the algorithm.*

3. The main algorithm. In this section we outline a constant-factor approximation algorithm for the DSPT: Find a minimum-size subset G of vertices of T such that each vertex of T is visible from one of the vertices in G. (The missing details are presented in section 4.)

We begin by computing the (upper) convex hull of T, denoted $CH(T)$. Notice that if m is the number of vertices in $CH(T)$, then at least $\lfloor \frac{m}{2} \rfloor$ guards are needed in order to guard T. In the first stage, we thus place a guard at each vertex of $CH(T)$. Notice that if T' is a subterrain defined by two consecutive vertices of $CH(T)$, then no point in its interior is visible from outside T'. We therefore may consider each of the $m-1$ subterrains that are defined by the vertices of $CH(T)$ separately (see Figure 2).

Let $T'' = [u, v]$ be a subterrain of T'. We say that T'' *requires a local guard* if there exists a vertex w in the interior of T'' that is not seen from $(T' - T'') \cup \{u, v\}$, or, in other words, $u < L(w) < R(w) < v$.

Let T' be one of the subterrains obtained from the first stage. In the second stage we partition T' into subterrains such that each of them does not require a local guard. We do this as follows. For each internal vertex v of T', we compute the points (vertices) $L(v)$ and $R(v)$. Let $T(v)$ denote the subterrain $[L(v), R(v)]$. Viewing the subterrains $T(v)$ as open horizontal intervals and considering only *minimal* intervals (i.e., intervals that do not contain other intervals), we compute a maximal set $\mathcal{S}$ of disjoint such intervals. Let $\mathcal{S}'$ be the complementary set of subterrains; thus, any two consecutive subterrains in $\mathcal{S}$ define at most one subterrain in $\mathcal{S}'$.

CLAIM 3.1. *Let T'' be a subterrain in $\mathcal{S} \cup \mathcal{S}'$. Then T'' does not require a local guard.*

Proof. The proof follows immediately from the construction. □

We place guards at the end vertices of the subterrains in $\mathcal{S}$. In addition, for each $T'' \in \mathcal{S} \cup \mathcal{S}'$, we place guards at the at most four vertices $R(a_l), R(a_r), L(b_l), L(b_r)$, where a_l (resp., a_r) is the leftmost (resp., rightmost) internal vertex of T'' that is seen from the right of T'', and b_l (resp., b_r) is the leftmost (resp., rightmost) internal vertex of T'' that is seen from the left of T''. Since $|\mathcal{S}|/2$ is clearly a lower bound on the number of guards needed to guard T', we increase the number of vertices by only a constant factor.

We now "solve" each of the subterrains in $\mathcal{S} \cup \mathcal{S}'$ *separately*. Each such subterrain is considered to be a *base case*, in that it does not require a local guard, and is solved using the base-case algorithms detailed in section 4. The *independence property* (based on Claims 3.3–3.4) shown below justifies this approach, i.e., that the subterrains in $\mathcal{S} \cup \mathcal{S}'$ may be solved separately without hurting the approximation bound.

The overall structure of the algorithm is thus as follows:

- Given a terrain T, compute its (upper) convex hull, place guards at the vertices of the convex hull, and solve each subterrain T' separately.
- Given a subterrain T', partition it into subterrains $\mathcal{S} \cup \mathcal{S}'$, as described above.

For each subterrain $T'' \in \mathcal{S} \cup \mathcal{S}'$, place guards at the end vertices of T'' and at the vertices $R(a_l), R(a_r), L(b_l), L(b_r)$, and solve the remaining unguarded fragments of T'' separately, using the base-case algorithms detailed in section 4.

The rest of this section deals with the independence property and its proof.

LEMMA 3.2 (the independence property). *Let T be a terrain, and let $T_1, \ldots, T_k$ be k disjoint subterrains of T. (Two subterrains may have a common end vertex.) Assume that the size l of an optimal solution for $T_1 \cup \cdots \cup T_k$ (by placing guards on T) is greater than or equal to k/c_1, for some constant c_1. Also assume that for each subterrain T_i we can compute a c_2-solution for (the yet unguarded fragments of) T_i, for some constant c_2. Then we can compute a c-solution for $T_1 \cup \cdots \cup T_k$, for some constant c. (A c_0-solution is a solution whose size is at most c_0 times the size of an optimal solution; thus a c-solution is of size at most cl.)*

We construct a c-solution for $T_1 \cup \cdots \cup T_k$. We begin by placing at most $2k$ guards at the end vertices of the subterrains $T_1, \ldots, T_k$. We need the following two claims.

CLAIM 3.3. *Let $r_1 < r_2 < \cdots < r_m$ be the internal vertices of T_i that can be seen from the right of T_i. Then*

1. *$R(r_1) \geq R(r_2) \geq \cdots \geq R(r_m)$.*
2. *If r_i, for $1 < i < m$, cannot be seen both from $R(r_m)$ and from $R(r_1)$, then it can be seen only (when viewing from the right of T_i) from vertices in the interior of the subterrain $[R(r_m), R(r_i)] \subseteq [R(r_m), R(r_1)]$.*

Proof. The first part follows immediately from Claim 2.1. If $R(r_2) > R(r_1)$, then by Claim 2.1, r_1 must see $R(r_2)$, which is impossible since r_1 does not see beyond $R(r_1)$. Similarly, we argue that $R(r_2) \geq R(r_3)$, etc.

To prove the second part, assume that r_i (for some $1 < i < m$) cannot be seen from $R(r_m)$ or from $R(r_1)$. Then r_i cannot be seen from a vertex to the left of $R(r_m)$ (and to the right of T_i), since if r_i is seen from such a vertex, then, by Claim 2.1, it is also seen from $R(r_m)$. Similarly, r_i cannot be seen from a vertex to the right of $R(r_1)$, since if it is seen from such a vertex, then, by Claim 2.1, r_1 is also seen from this vertex, which is impossible (by the definition of $R(r_1)$). ∎

Set $R(T_i) = [R(r_m), R(r_1)]$, and let $L(T_i)$ be the symmetric subterrain that is defined by considering the internal vertices of T_i that can be seen from the left of T_i .

CLAIM 3.4. *Let T_i, T_j be two of the subterrains above, such that T_j lies to the right of T_i. Let $R_j(T_i) \subseteq R(T_i)$ be the subterrain defined by considering only the internal vertices in T_i that can be seen from the right of T_j. Then $R(T_j)$ lies to the left of $R_j(T_i)$, where the right end vertex of $R(T_j)$ and the left end vertex of $R_j(T_i)$ may coincide.*

Proof. Assume there is an internal vertex in T_i that can be seen from the right of T_j. (Otherwise, $R_j(T_i)$ is empty.) Let u be the leftmost vertex in T_j that can be seen from the right of T_j. Then $R(u)$ defines the right end vertex of $R(T_j)$. By Claim 2.1, if v is any vertex in T_i that can be seen from the right of T_j, then $R(v) \geq R(u)$. ∎

A symmetric claim can be formulated using the subterrains $L(T_i)$ and $L_i(T_j)$ instead of $R(T_j)$ and $R_j(T_i)$, respectively.

We are now ready to continue the construction of a c-solution for $T_1 \cup \cdots \cup T_k$. For each T_i, we place guards at the end vertices of the subterrains $R(T_i)$ and $L(T_i)$. We have thus placed at most $4k$ guards in this step (and at most $6k$ guards in the first two steps). Next, for each T_i we compute a c_2-solution $\mathcal{V}_i$ for (the yet unguarded fragments of) T_i. We claim that the at most $6k$ guards of the first two steps together with the sets $\mathcal{V}_i$ form a c-solution for $T_1 \cup \cdots \cup T_k$.

Let $\mathcal{V}_{opt}$ be an optimal solution for $T_1 \cup \cdots \cup T_k$. Recall that we are assuming that $|\mathcal{V}_{opt}| \geq k/c_1$, for some constant c_1. Let $\mathcal{U}$ denote the set of at most $4k$ vertices that are the end vertices of the subterrains $R(T_i)$ and $L(T_i)$. Let $v \in \mathcal{V}_{opt} - \mathcal{U}$. We observe that there is at most one subterrain T_i to the left of v, which has an internal vertex that is not seen by a vertex of $\mathcal{U}$ but is seen from v. If there are two such subterrains T_i and T_j, where T_j is to the right of T_i, then, by Claim 3.3, v is an internal vertex of both $R(T_i)$ and of $R(T_j)$. More precisely, v is in $R_j(T_i)$ (but not its right end vertex), and v is an internal vertex of $R(T_j)$. But this is impossible by Claim 3.4. Thus v can help guarding at most one subterrain to its left, at most one subterrain to its right, and possibly the subterrain in which it lies. Our construction replaces v with at most $3c_2$ guards.

4. Base-case algorithms. Let T be a terrain with n vertices, and let $\mathcal{G}$ be the subset of vertices of T where guards have already been placed. Let A be a subterrain of T that does not require a local guard, and let A' be the subset of vertices of A that cannot be seen by $\mathcal{G}$. We wish to compute a set of guards $\mathcal{V}(A')$ located at vertices of T, some of which may be located within A, that together see all vertices in A', and such that the size of $\mathcal{V}(A')$ is within some constant factor of the size of an optimal such set of guards.

We distinguish between three base cases:

Case 0. We require that $\mathcal{V}(A')$ consist only of vertices that lie to the *left* of subterrain A.

Case 1. We require that $\mathcal{V}(A')$ consist only of vertices that lie to the *left* of or *within* subterrain A.

Case 2. We make no requirements of the set $\mathcal{V}(A')$ of vertices of T that guard subterrain A.

Each subterrain $T'' \in \mathcal{S} \cup \mathcal{S}''$ (see section 3) is passed to the Case 2 algorithm, which in turn may pass it on to the Case 1 algorithm; see Figure 3.

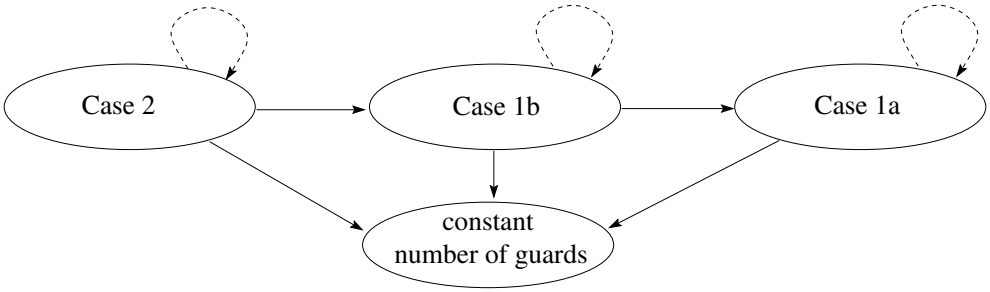


FIG. 3. *The program flow of the base-case algorithms: A dashed line denotes a recursive call, while a solid line denotes a call to another (simpler) algorithm.*

DEFINITION 4.1. *Let T be a terrain, and let $\mathcal{G}$ be the subset of vertices of T at which guards have already been placed. Let T' be the subset of vertices of T that cannot be seen by $\mathcal{G}$. We say that two subterrains $T_1, T_2 \subset T$ are* guard-independent with respect to $\mathcal{G}$ *if the set of vertices of T that are seen by $T_1 \cap T'$ and the set of vertices seen by $T_2 \cap T'$ are disjoint. In other words, T_1 and T_2 are* guard-independent *subterrains if any guard g that can see a vertex in $T_1 \cap T'$ cannot see a vertex in $T_2 \cap T'$ and vice versa.*

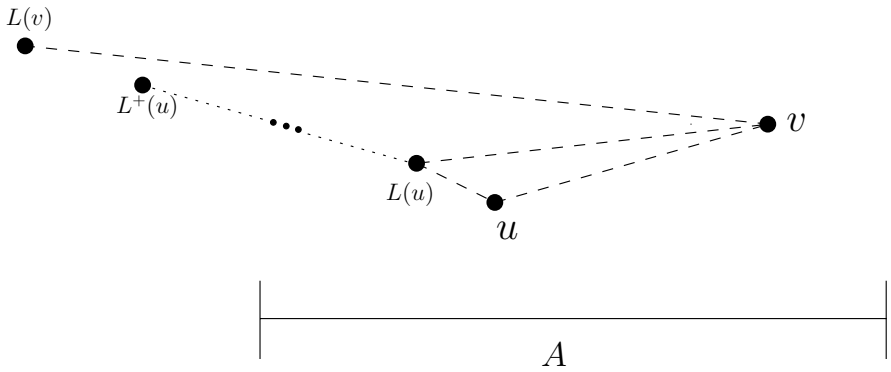


FIG. 4. *If u can see v, then $L(u)$ can also see v. If $L(u) \in A$, we apply Claim 4.3 to $L(u)$ to conclude that $L(L(u))$ dominates $L(u)$ and therefore also sees v, etc., until we reach a vertex that is not in A that dominates u.*

4.1. Case 0. Let A be a subterrain of T such that it is possible to guard the set of (so far unguarded) vertices A' using only guards at vertices of T to the left of A. Our goal is to determine a minimum-cardinality such set of *left guards* for A'. In this specially constrained case, we are able to determine an *optimal* set, $\mathcal{V}(A')$, of guards, using the following algorithm.

While A' contains an unguarded vertex, **do**

Place a guard at $L(a)$, where a is the leftmost vertex in A' that is not yet guarded.

LEMMA 4.2. *The algorithm above computes an optimal subset of left guards for A'.*

Proof. When the algorithm locates a guard at vertex $L(a)$, no vertex to the left of $L(a)$ can see a (by the definition of $L(a)$), and any vertex v to the right of $L(a)$ (and still to the left of A) that sees a is "dominated" by $L(a)$ (by Claim 2.1), in that a guard at $L(a)$ will see any vertex of A' that v sees. We continue by induction. ∎

4.2. Case 1. Let $A = [a, b]$ be a subterrain that does not require a local guard, let A' be the subset of vertices of A that remain to be guarded, and assume that A' can be guarded from the left of A, that is, by placing guards only at vertices of T to the left of A.

In this case (Case 1), our goal is to compute a set of guards $\mathcal{V}(A')$ for A' such that each guard in $\mathcal{V}(A')$ is either to the left of A or within A (but not to the right of A). We present a constant-factor approximation algorithm for computing a minimum-cardinality such set of guards.

We will need the following claim, which tells us that it does not make sense to place a guard within A if its sole purpose is to view rightwards.

CLAIM 4.3. *Let $A = [a, b]$ be a subterrain as above. Let $u \neq b$ be any vertex in A. Then $L(u)$ dominates u, in the sense that any vertex $v \in A'$ to the right of u that is seen by u is also seen by $L(u)$.*

Proof. Let $v \in A'$ be a vertex to the right of u that is seen by u. Recall that by our assumption, $L(v)$ lies to the left of A. We may also assume that $L(u) \neq L(v)$, since otherwise $L(u)$ clearly sees v. Now, on the one hand, both u and $L(u)$ must lie below the line $l(L(v), v)$, and, on the other hand, $L(u)$ must lie above the line $l(L(v), u)$. Thus $L(u)$ can see v (see Figure 4). ∎

COROLLARY 4.4. *There is always a vertex to the left of A that dominates u (with*

respect to the vertices of A' to the right of u). We shall denote by $L^+(u)$ any such vertex.

Proof. If $L(u) \in A$, then we apply Claim 4.3 to $L(u)$ to conclude that $L(L(u))$ also sees v, etc., until we reach a vertex that is not in A. This vertex clearly dominates u. □

COROLLARY 4.5. *$L^+(u)$ dominates any vertex w that lies between $L^+(u)$ and u (with respect to the vertices of A' to the right of u).*

Proof. The proof follows from the fact that w must lie below the chain $u, L(u)$, $L(L(u)), \ldots, L^+(u)$. □

We consider two subcases, according to whether or not the endpoints of A see each other.

4.2.1. Case 1a. The endpoints of A see each other. Let $A = [a, b]$ be a subterrain such that a and b see each other. For a vertex $q \in A$, $q \neq b$, that is visible from b, we denote by $A'_l(q)$ the vertices of A' that lie to the left of q and are not visible from b or $L^+(q)$. Similarly, let $A'_r(q)$ denote the vertices of A' that lie to the right of q and are not visible from b or $L^+(q)$. If both $A'_l(q)$ and $A'_r(q)$ are nonempty, we say that q implies a *nontrivial division.*

CLAIM 4.6. *If there exists a vertex $q \in A$, $q \neq b$, such that, b sees q, and both $A'_l(q)$ and $A'_r(q)$ are nonempty (i.e., q implies a nontrivial division), then $A'_l(q)$ and $A'_r(q)$ are guard-independent.*

Proof. The vertices that can see one or more vertices of $A'_l(q)$ can lie either to the right of $L^+(q)$ and to the left of A, or in the subterrain $[a, q]$. This follows from the fact that b can see q, and that any vertex to the left of $L^+(q)$ cannot see into $A'_l(q)$. Similarly, the vertices that can see one or more vertices in $A'_r(q)$ can lie either to the left of $L^+(q)$ or in the subterrain (q, b). Notice that q cannot see a vertex of $A'_r(q)$, since $L^+(q)$ cannot see such a vertex and, according to Claim 4.3, $L^+(q)$ dominates q with respect to the subterrain to the right of q. □

We say that A is in the *single-pocket case* if each vertex $q \in A$ to the left of b that is visible from b implies a trivial division (either $A'_l(q)$ is empty or $A'_r(q)$ is empty).

CLAIM 4.7. *Consider a subterrain A that is in the single-pocket case. Then there exists an (open) subterrain $A^* = (c, d) \subset A$, $d \neq b$, such that b cannot see any vertex in A^*, and all the vertices of A' that are not visible from b, $L^+(c)$, and $L^+(d)$ are in A^*.*

Proof. We may assume that there is no vertex q (to the left of b and visible from b) for which *both* $A'_l(q)$ and $A'_r(q)$ are empty; otherwise, we could place guards at b and $L^+(q)$ to see all of A'. Let c be the rightmost vertex (to the left of b and visible from b) that implies a (trivial) division in which $A'_l(c)$ is empty. Let d be the first vertex to the right of c that is visible from b. Then d implies a (trivial) division in which $A'_r(d)$ is empty. Set $A^* = (c, d)$. By definition, b cannot see any vertex in A^*. Also, any vertex in A' that is not visible from b, $L^+(c)$, or $L^+(d)$ must lie in A^*, since, by definition, b and $L^+(c)$ cover $[a, c) \cap A'$ as well as c, and b and $L^+(d)$ cover $(d, b] \cap A'$ as well as d. □

The single-pocket case. Consider the single-pocket case, where $A^* = (c, d)$ denotes the "pocket." Notice that (i) none of the vertices in $(d, b]$ can see into the pocket A^* (by Claim 2.1), and (ii) the vertices c and d see each other (since b sees both c and d but does not see any vertex in A^*). Also we know that b, $L^+(c)$, and $L^+(d)$ together cover $A' - A^*$.

For a subset of vertices S, we denote by $V(S, p)$ the subset of vertices of S that are visible from a vertex p, and by $V(S, P)$ the subset of vertices of S that are visible

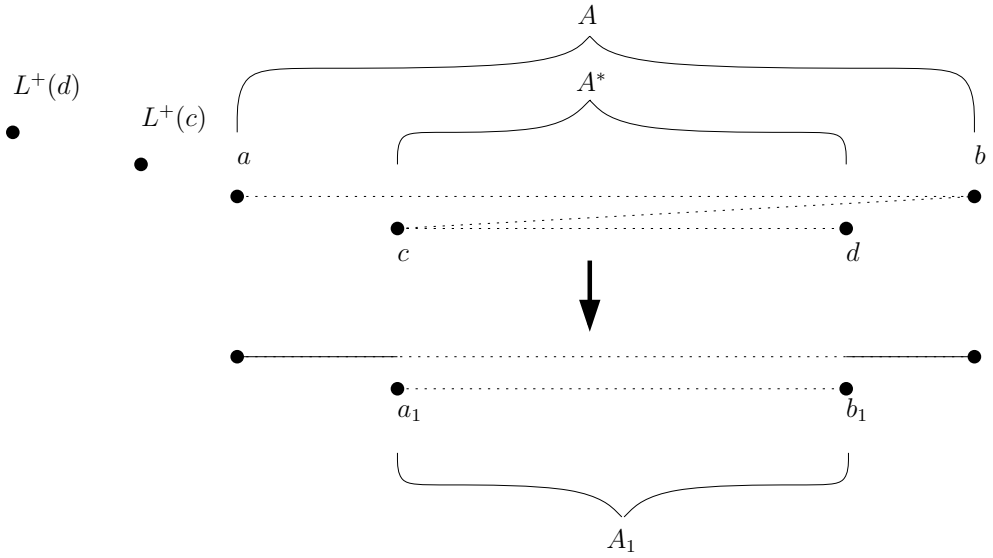


FIG. 5. *We can reduce the problem A to the Case* 1a *problem* $A_1 = [a_1, b_1] = [c, d]$, *since* $A' - A^* \subseteq V(A', L^+(d)) \cup V(A', d)$.

from at least one of the vertices of P.

LEMMA 4.8. *If* $V(A', L^+(d)) \cup V(A', d) \not\supseteq A' - A^*$, *then* (i) *one must place at least one guard outside of* A^* *in order to guard* $A' - A^*$, *and* (ii) *there exists a constant-size set of guards* U *that guards* $A' - A^*$, *and any other set of guards* U' *that guards* $A' - A^*$ *includes a guard* g' *such that* $V(A' \cap A^*, U) \supseteq V(A' \cap A^*, g')$.

Proof. To prove the first part, we observe that $L^+(d)$ dominates any vertex in A^* with respect to the subterrain to the right of d (applying Corollary 4.5), and d dominates any vertex in A^* with respect to the subterrain to the left of c (applying Claim 2.1 to d and c). Now let $v \in A' - A^*$ be a vertex that is not seen from $L^+(d)$ or from d (such a vertex exists by our assumption). Clearly any guard that sees v cannot lie in A^*.

To prove the second part, set $U = \{L^+(d), L^+(c), b\}$. We already know that U guards $A' - A^*$. We now show that if g' is a guard that sees v, then $V(A' \cap A^*, U) \supseteq V(A' \cap A^*, g')$. If v is to the right of A^*, then in order to guard v one must locate a guard g' either to the left of $L^+(d)$ or to the right of d. In both cases $V(A' \cap A^*, g') = \emptyset$. Otherwise, if v is to the left of A^*, then in order to guard v one must locate a guard g' either in $[L^+(c), c]$ or to the right of d. In the former case it is possible that g' sees into A^*, but it is dominated by $L^+(c)$. □

The lemma above implies that if $V(A', L^+(d)) \cup V(A', d) \not\supseteq A' - A^*$, then we may place three guards at $L^+(d)$, $L^+(c)$, and b and charge these guards to the guard g' (in the proof above).

We now consider the case in which $V(A', L^+(d)) \cup V(A', d) \supseteq A' - A^*$ (see Figure 5). In this case we say that A is *reducible*, and reduce $A_0 = A$ to the Case 1a problem $A_1 = [a_1, b_1] = [c, d]$. Now, if A_1 is also reducible, that is, if A_1 is a single-pocket case with pocket $A_1^* = (c_1, d_1)$, and $V(A', L^+(d_1)) \cup V(A', d_1) \supseteq A' - A_1^*$, where A' is the *original* A', then we reduce A_1 to the Case 1a problem $A_2 = [a_2, b_2] = [c_1, d_1]$. This process continues until we reach a problem $A_i = [a_i, b_i]$, $i > 0$, that is not reducible.

Notice that if we now place two guards at b_i and at $L^+(b_i)$, then all remaining unguarded vertices in A' will necessarily lie in the interior of A_i. This is because $b_i = d_{i-1}$ and A_{i-1} was reducible. Thus we place these two guards and proceed as follows, according to the state that we entered. There are three possible such states:

- A_i is not a single-pocket case; i.e., there is a vertex $q \in A_i$ that implies a nontrivial division.
- A_i is a single-pocket case, but $V(A', L^+(d_i)) \cup V(A', d_i) \not\supseteq A' - A_i^*$.
- A_i has a constant-size solution.

In the first state, we divide A_i into two guard-independent subterrains (see Claim 4.6) as follows. Let q_l be the leftmost vertex in A_i that is seen from b_i and implies a nontrivial division. Let a_i' be the rightmost vertex to the left of q_l that is seen by b_i. Notice that q_l and a_i' see each other, since b_i sees both of them and does not see any vertex between them. Also notice that b_i and $L^+(a_i')$ together cover the subterrain $[a_i, a_i']$, since a_i' implies a (left) trivial division. (Notice that it is impossible that a_i' implies a right trivial division, since, if it did, then so would q_l.)

Thus, in addition to the two guards that were already placed (i.e, at b_i and at $L^+(b_i)$), we also place guards at $L^+(q_l)$ and at $L^+(a_i')$. We now solve the two guard-independent subterrains $A_l = [a_i', q_l]$ and $A_r = (q_l, b_i]$ by applying to each of them the Case 1a algorithm. We charge the four guards that were placed to the increase by one of the lower bound due to the presence of a vertex that implies a nontrivial division.

In the second state we also place guards at $L^+(d_i)$ and $L^+(c_i)$ (in addition to the guards at b_i and $L^+(b_i)$) according to Lemma 4.8 above, and solve the subproblem $[c_i, d_i]$ by applying the Case 1a algorithm. We charge the four guards that were placed to the guard g' (see above). In the third state we simply solve the original problem with a constant number of guards.

All the above implies the following algorithm for Case 1a.

ALGORITHM 4.9. (Case 1a)

1. *If there exist two guards that together see all vertices in A'—done.*
2. *Let Q be the set of all vertices $q \in A$ that imply a nontrivial division.*
3. *If $Q \neq \emptyset$, let q_l be the leftmost vertex in Q. Locate three guards at b, $L^+(q_l)$, and $L^+(a')$, where a' is the rightmost vertex to the left of q_l that is visible from b. Solve each of the (guard-independent) subterrains $A_l = [a', q_l]$ and $A_r = (q_l, b]$ recursively using the Case* 1a *algorithm.*
4. *If $Q = \emptyset$ (the single-pocket case)*
   (a) *Compute the pocket $A^* = (c_0, d_0)$.*
   (b) *If $A = A_0$ is not reducible, place a guard at b. Otherwise, reduce until a subterrain $A_i = [a_i, b_i]$ that is not reducible is reached, and place guards at b_i and $L^+(b_i)$.*
   (c) *If A_i, $i \geq 0$, is a single-pocket case, then place guards at $L^+(c_i)$ and $L^+(d_i)$ and solve $[c_i, d_i]$ recursively using the Case* 1a *algorithm.*
   (d) *Else, solve A_i recursively using the Case* 1a *algorithm.*

The following lemma summarizes the result for Case 1a.

LEMMA 4.10. *Algorithm* 4.9 *computes a set of guards $\mathcal{V}(A')$ for A', whose size is bounded by some constant times the size of a minimum-cardinality such set of guards.*

We now turn to the general case, where A's endpoints do not see each other.

4.2.2. Case 1b: The endpoints of A do not see each other. We begin by computing the (upper) convex hull of A. Each of the edges of this convex hull corresponds to a subterrain whose endpoints see each other. Some of these subterrains

may already be fully guarded (by $\mathcal{G}$); we consider only those that are not yet fully guarded. Let $A_1 = [a, b]$ be the leftmost subterrain that is not yet fully guarded, let A_1' be the subset of vertices in A_1 that are not yet guarded, and let u be the leftmost vertex in A_1'. If there exists a single vertex to the left of A that sees all vertices in A_1', then $L(u)$ is necessarily such a vertex. (Note that $L(u)$ is necessarily to the left of A since $u \in A_1'$.) Moreover, in this case, any vertex g (either to the left of A or in A) that sees u is dominated by the pair of vertices $L(u)$ and $L(b)$. Thus, in this case we can place guards at $L(u)$ and at $L(b)$ and charge this to the guard of the optimal solution that sees u.

Assume now that A_1' cannot be fully guarded by a single vertex to the left of A. In this case, we place a guard at $L(b)$ and distinguish between two cases. If after placing a guard at $L(b)$ there is no vertex to the right of b that remains to be guarded, then apply Algorithm 4.9 to subterrain A_1 and charge the guard at $L(b)$ to the one-time event of leaving Case 1b. If, however, there still remains an unguarded vertex to the right of b, then the two subterrains A_1 and $A_2 = A - A_1$ are guard-independent. Indeed, any guard that can help guarding A_1' must lie in $(L(b), b]$, and any guard that can help guarding A_2' must lie either in A_2 or to the left of $L(b)$. Notice that b cannot help in guarding A_2', since $L(b)$ dominates b with respect to visibility to the right of b.

ALGORITHM 4.11. (Case 1b)

1. *If all of A is guarded—done.*
2. *Let $A_1 = [a, b]$ be the leftmost subterrain that is not yet fully guarded, let A_1' be the subset of remaining unguarded vertices in A_1, and let u be the leftmost vertex in A_1'.*
3. *If there exists a single vertex to the left of A_1 that sees all vertices in A_1', then locate guards at $L(u)$ and at $L(b)$, go to Step #1.*
4. *Else, locate a guard at $L(b)$ and solve the two guard-independent subproblems A_1, using the Case* 1a *algorithm, and $A - A_1$, using the Case* 1b *algorithm.*

The following lemma summarizes the result for Case 1b.

LEMMA 4.12. *Algorithm* 4.11 *computes a set of guards $\mathcal{V}(A')$ for A', whose size is bounded by some constant times the size of a minimum-cardinality such set of guards.*

4.3. Case 2. Given a terrain T, let $A = [a, b]$ be a subterrain that does not require a local guard, let A' be the subset of all vertices in A that are not yet guarded. In this case (Case 2), our goal is to compute a set of guards $\mathcal{V}(A')$ for A', where the guards in $\mathcal{V}(A')$ may be located anywhere in T. We present a constant-factor approximation algorithm for computing a minimum-cardinality such set of guards.

We may assume that the endpoints of A are in A', because otherwise we can simply replace A with the subterrain $[a', b']$, where a' (resp., b') is the leftmost (resp., rightmost) vertex in A', and any (Case 2) solution to A is also a (Case 2) solution to $[a', b']$ and vice versa. In particular, we may assume that a and b are not vertices of the convex hull of the initial terrain (since A is contained in a subterrain defined by two consecutive vertices v_1, v_2 of the convex hull of the initial terrain, and we already placed guards at these vertices).

We will need the following observation.

OBSERVATION 4.13. *For any vertex $v \in A$, the vertices $L(v)$ and $R(v)$ can see each other.*

(If there were a vertex v for which $L(v)$ and $R(v)$ could not see each other, then the upper angle formed by $L(v)$, v, and $R(v)$ would be greater than π, and v would be a vertex of the convex hull of the original terrain (in between v_1 and v_2), contradicting

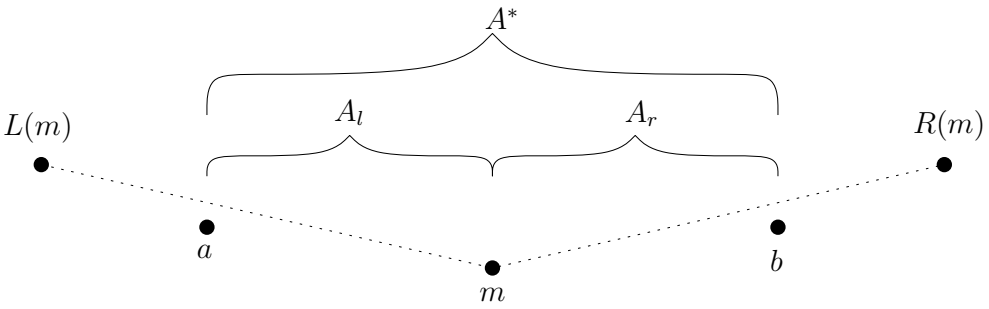


FIG. 6. *The two subterrains defined by a shared vertex that implies a nontrivial division are guard-independent.*

our assumptions above.)

A vertex $m \in A$ is called a *shared vertex* if m can be seen both from the left of A and from the right of A. For a shared vertex m, let $A'_l(m)$ (resp., $A'_r(m)$) denote the subset of vertices of A' that lie to the left (resp., to the right) of m and are not visible from $L(m)$, m, or $R(m)$. As in Case 1a, we say that m implies a *nontrivial division* if both $A'_l(m)$ and $A'_r(m)$ are not empty.

CLAIM 4.14. *Let m be a shared vertex that implies a nontrivial division. Then the two subterrains $A_l = [a, m)$ and $A_r = (m, b]$ are guard-independent.*

Proof. Any guard that can see a vertex in $A'_l(m)$ must lie either between $L(m)$ and m or to the right of $R(m)$, while any guard that can see a vertex in $A'_r(m)$ must lie either to the left of $L(m)$ or between m and $R(m)$. See Figure 6. ∎

CLAIM 4.15. *There is at least one shared vertex in $A = [a, b]$.*

Proof. a is surely seen from the left of A (e.g., by the vertex immediately to its left), and b is surely seen from the right of A. Let c be the rightmost vertex in A that is seen from the left of A. (If $c = b$, then let c be the previous vertex in A that is seen from the left of A.) Let d be the vertex immediately to the right of c. Then d is necessarily seen from the right of A. We observe that either $L(c)$ or $R(d)$ must lie above the line through c and d. (Otherwise, c and d are vertices of the convex hull of the original terrain—impossible; see assumptions and observation above.) Thus, either c or d is a shared vertex; e.g., if $L(c)$ lies above the line through c and d, it must also see d, so d is a shared vertex in this case. ∎

Let M be the set of all shared vertices in A. According to the claim above, $M \neq \emptyset$. We next show that any subterrain of A without shared vertices (i.e., vertices of M) can be nicely divided into two subterrains.

CLAIM 4.16. *Let $A^* = [l, r] \subset A$ be a subterrain such that $A^* \cap M = \emptyset$. Let u be a vertex in A^*. If u can be seen from the left of A, then any vertex in A^* to the right of u can also be seen from the left of A (and therefore cannot be seen from the right of A).*

Proof. Since A does not require a local guard, any vertex in A^* can be seen either from the left of A or from the right of A, but since $A^* \cap M = \emptyset$, it can be seen from only one of these sides. Assume that there is a vertex to the right of u that is seen from the right of A. Let v be the leftmost such vertex, and let u' be the vertex immediately to the left of v. Then u' (which is possibly u) is seen from the left of A. Observe that either $L(u')$ or $R(v)$ must lie above the line through u' and v (since otherwise u' and v would be vertices of the convex hull of the original terrain), and therefore at least one of the two vertices u', v is a shared vertex—contradicting our

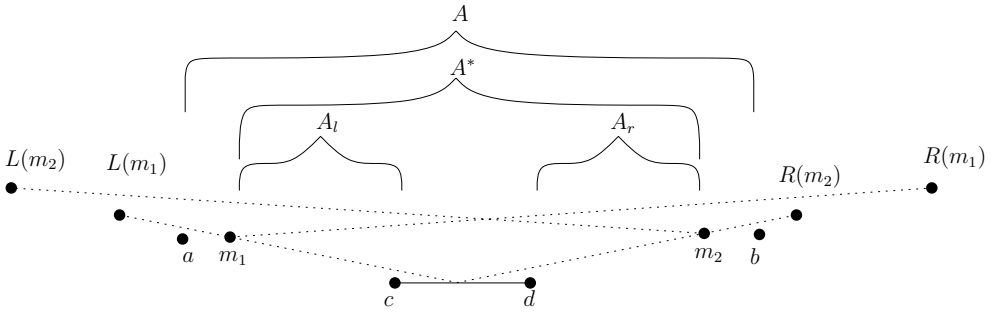


FIG. 7. *By locating guards at $L(m_1)$, m_1, m_2, and $R(m_2)$, we obtain two guard-independent subproblems, A_l and A_r, both of Case* 1.

assumption. We conclude that there is no vertex to the right of u that is seen from the right of A.

Now let d be the leftmost vertex in A^* that can be seen from the left of A. If there is no such d, then every vertex in A^* is seen from the right of A but not from the left of A, and if $d = l$, then any vertex in A^* is seen from the left of A but not from the right of A. Otherwise, let c be the vertex immediately to the left of d. Then every vertex in the subterrain $A_l = [l, c]$ is seen from the right of A but not from the left of A, and every vertex in the subterrain $A_r = [d, r]$ is seen from the left of A but not from the right of A. □

CLAIM 4.17. *Assume that for each $m \in M$, m implies a trivial division. Then, by placing a constant number of guards, one can reduce A either to a single instance of Case* 1*, or to two guard-independent instances of Case* 1*.*

Proof. Let $m \in M$ be a shared vertex (such a vertex exists as shown in Claim 4.15). We know that m implies a trivial division. Assume, e.g., that (after placing guards at $L(m)$, m, and $R(m)$) A_l is fully guarded but A_r is not. Let m_1 be the rightmost shared vertex for which A_l is fully guarded. We now distinguish between two possible situations: (i) there exists another shared vertex to the right of m_1, and (ii) m_1 is the rightmost shared vertex. In both cases the subterrain $[a, m_1]$ is fully guarded by $L(m_1)$, m_1, and $R(m_1)$. Consider the former more general situation. Let m_2 be the shared vertex immediately to the right of m_1. Now m_2 implies a trivial division such that A_r is fully guarded (after placing guards at $L(m_2)$, m_2, and $R(m_2)$). Set $A^* = (m_1, m_2)$. A^* does not have a shared vertex, and $[m_2, b]$ is fully guarded by $L(m_2)$, m_2, and $R(m_2)$. We now use Claim 4.16 to divide (m_1, m_2) into two subterrains $A_l = (m_1, c]$, $A_r = [d, m_2)$ (one of them might be empty) such that any vertex in A_l is seen from the right of A but not from the left of A, and any vertex in A_r is seen from the left of A but not from the right of A. Observe that $L(m_1)$ and m_1 dominate the visibility of any vertex in $[a, m_1)$ with respect to A_l, so none of the vertices to the left of A_l can help in guarding the remaining unguarded vertices in A_l. Similarly, $R(m_2)$ and m_2 dominate the visibility of any vertex in $(m_2, b]$ with respect to A_r. Also if we place guards at c and $R(c)$ and at d and $L(d)$, then the two subterrains A_l and A_r are guard-independent, and both can be treated as Case 1 problems (see Figure 7).

Consider the second situation (i.e., there is no shared vertex in $A_r = (m_1, b]$). We apply Claim 4.15 to observe that no vertex in A_r can be seen from left of A. (b can be seen from right of A, and therefore any vertex in A_r can be seen from the right of A, and if there were a vertex that could also be seen from the left of A, then we would

have a shared vertex.) Thus, locating guards at $L(m_1)$, m_1, and $R(m_1)$ guarantees that the only vertices that can help guarding A_r are within A_r or to the right of A_r, which is a Case 1 problem. (Once again none of the vertices in $[a, m_1)$ can help guard A_r, since $L(m_1)$ and m_1 dominate these vertices with respect to A_r.) ∎

ALGORITHM 4.18. (Case 2)

1. *If there exist two (or any constant number of) guards that together see all vertices in A'—done.*
2. *Compute the set M of all shared vertices.*
3. *If there exists $m \in M$ that implies a nontrivial division, then locate guards at $L(m)$, m, and $R(m)$ and solve (recursively) each of the two subproblems A_l and A_r.*
4. *Else, for each $m \in M$ one of the sides is fully guarded by $L(m)$, m, and $R(m)$, and the other is not. Use Claim* 4.17 *in order to reduce A (by placing a constant number of guards) either to a single instance of Case* 1 *or to two guard-independent instances of Case* 1*. (Notice that if A is reduced to a single instance of Case* 1*, we charge the guards that were placed in doing so to the one-time event of leaving Case* 2*.)*

The following lemma summarizes the result for Case 2.

LEMMA 4.19. *Algorithm* 4.18 *computes a set of guards $\mathcal{V}(A')$ for A', whose size is bounded by some constant times the size of a minimum-cardinality such set of guards.*

5. Algorithm analysis. Throughout our description of the approximation algorithm for the DSPT, whenever guards were placed we gave an appropriate charging argument to justify why we could afford to place them. Consequently, we have shown that the size of the guarding set that is computed by the algorithm is bounded by a constant times the size of a minimum-size such set: We have obtained an $O(1)$-approximation, as desired. We have not attempted to minimize the constant factor; we leave this task to future work.

Concerning the running time of the algorithm, it is clear that it is polynomial. The following lemma shows that the running time is $O(n^2)$.

LEMMA 5.1. *The running time of the constant-factor approximation algorithm for the DSPT is $O(n^2)$.*

Proof. As a preliminary stage, we compute the *visibility graph*, $\mathrm{V}G_T(V)$, of the terrain vertices. This can be done easily in $O(n^2)$ time (or in output-sensitive time, but this does not yield an improved overall time bound for our algorithm). For each vertex $v \in V$, we compute in linear time the subset of vertices in V that lie to the left (alternatively, to the right) of v and are visible from v; the vertices in these subsets are found one-by-one in decreasing (alternatively, increasing) x-order. In particular the vertices $L(v)$ and $R(v)$ are the leftmost in the left list and rightmost in the right list, respectively. It remains to show that the subsequent stages of the algorithm can be carried out within the quadratic time bound.

In the first stage, we simply compute the (upper) convex hull of T in $O(n)$ time. In the second stage, we partition each "concave" subterrain (corresponding to an edge of the convex hull) into subterrains that do not require a *local guard* (see section 3). This can be done in $O(m \log m)$ time per "concave" subterrain of m vertices, and therefore in overall $O(n \log n)$ time.

In the final stage, each of the subterrains obtained in the second stage is solved separately, using the base-case algorithms, in $O(m^2)$ time, where m is the size of the subterrain (see Lemma 5.2 below). Since the subterrains are disjoint, the overall running time of this stage is $O(n^2)$. ∎

LEMMA 5.2. *Let A, $A \subset T$, be a subterrain that does not require a* local guard. *Then A can be solved using the base-case algorithms (with minor modifications) in $O(m^2)$ time, where m is the size of A.*

Proof. For the lemma to be true, we need to modify the first step in each of the base-case algorithms, so that, unless all vertices in A' are already guarded, one must continue by recursive calls. This change increases the depth of the recursion by only a constant number of levels.

Case 2. The condition in the modified step 1 can be checked in constant time. In step 2 all *shared vertices* can be computed in linear time, since one needs to consider only the *ranges* of the vertices in A (which were computed in the preliminary stage of the main algorithm). Checking in step 3 whether a *nontrivial division* is possible is also done in linear time, since for each shared vertex one needs to consider only a constant number of vertices. Moreover, the number of such divisions is clearly $O(m)$. In step 4 we reduce the problem to two guard-independent instances of Case 1; the reduction is done in constant time.

Case 1b. Step 2 is clearly linear in m. In steps 3 and 4 we reduce the problem to a smaller one; therefore the number of these reductions is $O(m)$.

Case 1a. Computing all vertices that imply a nontrivial division (in step 2) is done in linear time (since, for each vertex $q \in A$, one needs to consider only a constant number of vertices). In step 3 we reduce the problem to two guard-independent and disjoint problems, so the number of these reductions is $O(m)$; the reduction itself takes only constant time. In step 4 we reduce the problem to a smaller one. Again, the number of these reductions is clearly $O(m)$, and the reduction itself takes only constant time. ∎

The following theorem summarizes this section.

THEOREM 5.3. *The algorithm of sections* 3 *and* 4 *computes a constant-factor approximation for the DSPT in $O(n^2)$ time.*

6. The terrain guarding algorithm. In this section we generalize the constant-factor approximation algorithm for the DSPT to the general 1.5D terrain guarding problem, where guards can be placed anywhere on the terrain, and all points of the terrain (not only its vertices) must be guarded. For this we present a reduction from the general 1.5D terrain guarding problem to the DSPT. Figure 8 (right) shows that a solution to the DSPT is not necessarily a solution to the general 1.5D terrain guarding problem.

Let V be the set of vertices of T, and set $n = |V|$.

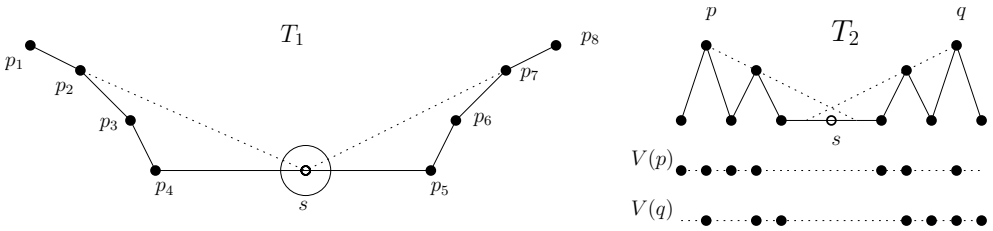


FIG. 8. *Left: A single guard at s sees all points of the terrain T_1, but if one may locate guards only at vertices, then two guards are needed. Right: All vertices of the terrain T_2 can be guarded with two guards (at p and at q), but in order to guard all points of T_2 three guards are needed.*

OBSERVATION 6.1. *Any solution to the general* 1.5*D terrain guarding problem can be transformed into another solution whose size is at most twice the size of the original solution, such that all guards in the new solution are vertex guards, i.e., located at vertices of the terrain.*

Proof. Replace each guard g in the original solution that is not a vertex of the terrain with the two endpoints v_i, v_{i+1} of the edge T on which g lies. Clearly, any point of T that is seen by g is also seen by at least one of these two vertices. □

LEMMA 6.2. *There exists a set U of points on T such that every subset of vertices $V' \subseteq V$ that guards U also guards T. Moreover, $|U| = O(n^2)$ and U can be computed in $O(n^2)$ time.*

Proof. Let p be a point on T, and let $\mathrm{vis}(p)$ denote the set of all points on T that are visible from p. $\mathrm{vis}(p)$ is the union of a linear number of maximal subterrains. For such a subterrain A, it is easy to see that if $p \in A$, then both endpoints of A must be vertices of T, and if $p \notin A$, then the farther of the two endpoints of A must be a vertex of T. We refer to the endpoints of the maximal subterrains in $\mathrm{vis}(p)$ as the *visibility events* induced by p.

Let U' be the set of visibility events induced by the vertices in V. Let U'' be the set that is obtained by picking an arbitrary point of T between each pair of consecutive points in U'. Set $U = V \cup U' \cup U''$. It is clear that $|U| = O(n^2)$ and that U can be computed in $O(n^2)$ time.

The set U' induces a partition of T into $O(n^2)$ intervals. Observe that if p is a point in the interior of an interval s of this partition and p' is the point of U'' that lies in the interior of s, then $\mathrm{vis}(p) \cap V = \mathrm{vis}(p') \cap V$. This follows immediately from the definition of U', since if there were a vertex $v \in V$ such that p saw v and p' did not see v (or vice versa), then there would be a visibility event somewhere in the interior of s.

Now let V' be a subset of V that guards U, and let p be any point on T, $p \notin U$. We need to show that p is guarded by V'. Let s be the interval of the partition of T induced by U' such that p lies in its interior, and let p' be the point of U'' that lies in the interior of s. Then by our assumption p' is guarded by V', and, therefore, by the observation above, so is p. □

We are now ready to present the algorithm for the general 1.5D terrain guarding problem.

ALGORITHM 6.3. General 1.5D terrain guarding.

1. *Given a terrain T with vertex set V, compute the set U.*
2. *Solve the DSPT with U as the vertex set (using the DSPT algorithm as a "black box"). Let $\mathcal{G}'$, $\mathcal{G}' \subseteq U$, denote the solution obtained.*
3. *Replace each point $g \in \mathcal{G}' - V$ with the two vertices of V adjacent to it. Let $\mathcal{G}$ be the resulting set.*
4. *Return $\mathcal{G}$.*

From Observation 6.1 and Lemma 6.2 it is clear that $\mathcal{G}$ is indeed a solution to the general 1.5D terrain guarding problem. Moreover, $\mathcal{G}$ can be computed in $O(n^4)$ time (since $|U| = O(n^2)$ and the running time of the DSPT algorithm is $O(|U|^2)$). It remains to prove that the size of $\mathcal{G}$ is bounded by some constant times the size of an optimal solution.

Let $opt(A, B)$ be an optimal solution to the problem of guarding A by placing guards at points of B, where A (alternatively, B) is either a subterrain of T or a discrete set of points of T. In particular, $opt(T, T)$ is an optimal solution to the general 1.5D terrain guarding problem. Notice that if $A' \subseteq A$ and $B' \subseteq B$, then

$|opt(A,B)| \geq |opt(A',B)|$ and $|opt(A,B')| \geq |opt(A,B)|$.

LEMMA 6.4. *Let c' be the (constant) approximation factor of the DSPT algorithm. Then $|\mathcal{G}| \leq 4c'|opt(T,T)|$.*

Proof. Using the observation above and Observation 6.1, $|opt(T,T)| \geq |opt(U,T)| \geq |opt(U,V)|/2 \geq |opt(U,U)|/2 \geq \mathcal{G}'/(2c') \geq \mathcal{G}/(4c')$. □

The following theorem summarizes the main result of this section.

THEOREM 6.5. *Algorithm* 6.3 *computes a constant-factor approximation for the general* 1.5*D terrain guarding problem in $O(n^4)$ time.*

7. Conclusion. There are two notable open problems: (1) Are the DSPT and 1.5D terrain guarding problems NP-hard? (the hardness claim in [4] has gaps in the proof); and (2) is there an approximation algorithm for guarding a simple polygon? So far, our attempts to generalize our methods to simple polygons have failed; our method strongly exploits the special structure of 1.5D terrains.

REFERENCES

[1] B. BEN-MOSHE, M. J. KATZ, AND J. S. B. MITCHELL, *A constant-factor approximation algorithm for optimal terrain guarding*, in Proceedings of the 16th Annual ACM-SIAM Symposium Discrete Algorithms, Vancouver, BC, 2005, SIAM, Philadelphia, 2005, pp. 515–524.

[2] P. BOSE, T. SHERMER, G. TOUSSAINT, AND B. ZHU, *Guarding polyhedral terrains*, Comput. Geom. Theory Appl., 7 (1997), pp. 173–185.

[3] H. BRÖNNIMANN AND M. T. GOODRICH, *Almost optimal set covers in finite VC-dimension*, Discrete Comput. Geom., 14 (1995), pp. 263–279.

[4] D. Z. CHEN, V. ESTIVILL-CASTRO, AND J. URRUTIA, *Optimal guarding of polygons and monotone chains*, in Proceedings of the 7th Annual Canadian Conference on Computational Geometry, Quebec City, QB, 1995, pp. 133–138.

[5] O. CHEONG, A. EFRAT, AND S. HAR-PELED, *On finding a guard that sees most and a shop that sells most*, in Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA, 2004, SIAM, Philadelphia, 2004, pp. 1091–1100.

[6] A. EFRAT AND S. HAR-PELED, *Locating guards in art galleries*, in Proceedings of the 2nd Annual IFIP International Conference on Theoretical Computer Science, Montréal, QB, 2002, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002, pp. 181–192.

[7] S. J. EIDENBENZ, *(In-)Approximability of Visibility Problems on Polygons and Terrains*, Ph.D. thesis, Department of Computer Science, ETH Zürich, Switzerland, 2000.

[8] S. J. EIDENBENZ, *Approximation algorithms for terrain guarding*, Inform. Process. Lett., 82 (2002), pp. 99–105.

[9] S. J. EIDENBENZ, C. STAMM, AND P. WIDMAYER, *Inapproximability results for guarding polygons and terrains*, Algorithmica, 31 (2001), pp. 79–113.

[10] S. K. GHOSH, *Approximation algorithms for art gallery problems*, in Proceedings of the Canadian Information Processing Society Congress, 1987, pp. 429–434.

[11] H. GONZÁLEZ-BANOS AND J.-C. LATOMBE, *A randomized art-gallery algorithm for sensor placement*, in Proceedings of the 17th Annual ACM Symposium on Computational Geometry, Medford, MA, 2001, pp. 232–240.

[12] J. M. KEIL, *Polygon decomposition*, in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, eds., Elsevier Science, North–Holland, Amsterdam, 2000, pp. 491–518.

[13] B. J. NILSSON, *Guarding Art Galleries—Methods for Mobile Guards*, Ph.D. thesis, Department of Computer Science, Lund University, Lund, Sweden, 1995.

[14] J. O'ROURKE, *Art Gallery Theorems and Algorithms*, Internat. Ser. Monogr. Comput. Sci., Oxford University Press, New York, 1987.

[15] J. O'ROURKE AND K. J. SUPOWIT, *Some NP-hard polygon decomposition problems*, IEEE Trans. Inform. Theory, 30 (1983), pp. 181–190.

[16] T. C. SHERMER, *Recent results in art galleries*, Proc. IEEE, 80 (1992), pp. 1384–1399.

[17] J. URRUTIA, *Art gallery and illumination problems*, in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, eds., North–Holland, Amsterdam, 2000, pp. 973–1027.

SIAM J. COMPUT.
Vol. 36, No. 6, pp. 1648–1671

FINDING PATHS AND CYCLES OF SUPERPOLYLOGARITHMIC LENGTH*

HAROLD N. GABOW†

Abstract. Let ℓ be the number of edges in a longest cycle containing a given vertex v in an undirected graph. We show how to find a cycle through v of length $\exp(\Omega(\sqrt{\log \ell / \log\log \ell}))$ in polynomial time. This implies the same bound for the longest cycle, longest vw-path, and longest path. The previous best bound for longest path is length $\Omega((\log \ell)^2/\log\log \ell)$ due to Björklund and Husfeldt. Our approach, which builds on Björklund and Husfeldt's, uses cycles to enlarge cycles. This self-reducibility allows the approximation method to be iterated.

Key words. approximation algorithms, graph algorithms, long paths, long cycles

AMS subject classifications. 68R10, 68W25, 05C38, 05C85

DOI. 10.1137/S0097539704445366

1. Introduction.

Interest in approximating the longest path of a graph was rekindled by Karger, Motwani, and Ramkumar [13], who were motivated by the large gap between known performance guarantees and known hardness results. We make some progress in reducing the gap by presenting the best known polynomial time approximation algorithm. In particular we show how to find paths of greater than polylogarithmic length in polynomial time.

Previous work. Monien [15] investigated fixed parameter algorithms for long paths and cycles. In particular he showed how to find a vw-path of length (exactly) k for all pairs of vertices v, w for which such a path exists, in time $O(k!nm)$. (Throughout this paper, n and m denote the number of vertices and edges, respectively, of the given graph.) Monien also proved a fact about undirected cycle length that is useful in finding cycles of length $\geq k$; a variant of this fact is stated below.

In other early work, Fellows and Langston showed how to find undirected cycles of length $\geq k$ using Robertson–Seymour theory [9]. Bodlaender [2] used dynamic programming to find long undirected paths and cycles, improving Monien's bounds.

Alon, Yuster, and Zwick [1] introduced the technique of color coding to find paths, cycles, and other subgraphs of guaranteed size. For example, in a directed or undirected graph, color coding finds a path whose length is either the greatest possible or $\geq \log n$ in polynomial time. The same holds for vw-paths. For cycles, for any $k \leq \log n$, color coding finds a cycle of length exactly k in polynomial time if one exists.

Björklund and Husfeldt [3] find an undirected path of length $\Omega((\log \ell/\log\log \ell)^2)$ in polynomial time, for ℓ the longest path length. Gabow and Nie [11] observe that the length guarantee is actually $\Omega(\log^2 \ell/\log\log \ell)$. This is the best known bound to date for undirected paths.

Better results are known for undirected graphs of low degree. Feder, Motwani, and Subi [8] find a cycle of length $\geq \ell^{\log_9 2} > \ell^{0.315}$ in graphs of maximum degree

*Received by the editors July 15, 2004; accepted for publication (in revised form) August 25, 2006; published electronically March 2, 2007. A preliminary version of this paper appeared in STOC'04 [10].

http://www.siam.org/journals/sicomp/36-6/44536.html

†Department of Computer Science, University of Colorado at Boulder, Boulder, CO 80309-0430 (hal@cs.colorado.edu).

three; here ℓ is the longest cycle length, and the algorithm runs in polynomial time. A similar result holds for longest path. The length guarantee improves on 3-connected cubic graphs.

Gabow and Nie [11] investigate long directed cycles. They also prove a variant of Monien's undirected cycle structure theorem: If a connected undirected graph has a cycle of $\geq k$ edges, either every depth-first search spanning tree has a fundamental cycle of $\geq k$ edges or some cycle has between k and $2k$ edges. This result, or Monien's, allows color coding to be applied to the problem of finding a cycle of $\geq k$ edges. (For example, it allows color coding to find a cycle of length $\geq \log n$ in polynomial time, if one exists.)

The best known hardness results are due to Karger, Motwani, and Ramkumar [13]. They showed that getting a constant factor approximation to the longest undirected path is NP-hard. Furthermore for any $\epsilon > 0$, approximating to within a factor $2^{O(\log^{1-\epsilon} n)}$ is quasi-NP-hard. Stronger hardness results for directed graphs are given in [4].

Our work also draws on previous work on a different problem on cycles: finding a cycle through three given vertices. Robertson and Seymour showed that more generally the fixed vertex subgraph homeomorphism problem can be solved in polynomial time [16]. However, huge constants are involved in this approach. Instead we use the algorithm of LaPaugh and Rivest [14], which finds a cycle through three given vertices in linear time with no large hidden constants. (See also [18].)

Our contribution. A *v-cycle* is a cycle containing the vertex v. We take the basic problem to be approximating the longest v-cycle, where v is a given vertex of degree two. The problems of approximating a longest cycle, a longest vw-path, and a longest path are all easily reduced to our problem in polynomial time.

Let ℓ be the length of a longest v-cycle in an undirected graph. We show that a v-cycle of length $\exp(\Omega(\sqrt{\log \ell / \log\log \ell}))$ can be found in polynomial time. This implies the same bound for the longest cycle, longest vw-path, and longest path. This improves the previous best bound of Björklund and Husfeldt for longest path given above. It also improves the previous best bound for longest vw-paths and cycles, which was only $\log n$ using color coding.

We note that our results give further evidence of the difference in complexity between the directed and undirected versions of the long path and cycle problems. The best known length guarantees for long directed paths and cycles are only logarithmic: a directed path (cycle) of length $\log n$ ($\log n / \log\log n$) remains the best that can be found in polynomial time even when a Hamiltonian path (Hamiltonian cycle) exists [1, 11]. Further, [4] provides hardness results that support the relative difficulty of the directed problem.

Our approach builds on Björklund and Husfeldt's idea of using cycles to enlarge paths. We use cycles to enlarge cycles, giving a self-reducibility property that allows the construction to be iterated. We note that the hardness results of Karger, Motwani, and Ramkumar [13] are based on a self-improvability property of the longest path problem involving graph products; it is unclear whether this has any relation to our self-reducibility property.

Our results hinge on properties of biconnected components, cutpoints, and separating pairs. The self-reducibility gives rise to a family of recursive algorithms. We can recur only a limited number of times because of the need to keep the graphs large.

The paper is organized as follows. Section 2 presents the facts on cutpoints and separation pairs that underlie our algorithm. Section 3 presents the algorithm. The

next two sections give the analysis: section 4 proves the length guarantee, and section 5 provides details of the implementation and uses them to prove the polynomial time bound. Section 6 gives some concluding remarks. We close this section with our terminology.

Terminology. A fraction a/bc is always an abbreviation of $a/(bc)$, e.g., $a/2k$. All logarithms are base two unless noted otherwise. When used as a number, e is the base of natural logarithms. $\exp(x)$ denotes e^x.

Our graph terminology is consistent with [19] whenever possible. All graphs in this paper are undirected and simple. $G[X]$ denotes the subgraph induced by vertex set X. For X and Y disjoint vertex sets, $E[X,Y]$ consists of all edges joining X and Y. Furthermore, by convention, writing $xy \in E[X,Y]$ means $x \in X$ and $y \in Y$.

All paths and cycles in this paper are simple. In contrast a walk can have repeated vertices. (So a path is a simple walk.) Let x and y be vertices. An *xy-path* is a path from x to y. For $x \neq y$, $d(x,y)$ denotes the length of a shortest xy-path. If the graph of interest is unclear in some notation, we include it as a subscript, e.g., $d_G(x,y)$.

We represent a path as a list of vertices, e.g., x, y, z. We also allow paths in the list; e.g., if xy is an edge and Z is a path starting at y, then x, Z denotes a path that is one edge longer than Z. Sometimes we write x, y, Z for the same path to remind the reader of the first edge xy. This will not cause any confusion.

If P is a path containing x and y with x preceding y, then $P[x,y]$ denotes the subpath of P from x to y. We occasionally write $P[y,x]$ to refer to the subpath from y to x in the reverse path of P; however, to prevent confusion we always indicate when this extended notation is being used. If C is a cycle containing x, y, and v, then $C_v[x,y]$ ($C_{\overline{v}}[x,y]$) denotes the subpath of C from x to y that contains (avoids) v, respectively. We extend the subpath notation to allow open-sided intervals; e.g., $P(v,w]$ is the path $P[v,w] - v$.

If P is a path, we use P to denote a set of vertices or edges, as is convenient; if the exact meaning is not clear from context, we state it explicitly. $|P|$ always denotes the number of edges in P.

2. Approach. Throughout this paper G is a given connected undirected graph. Our main algorithm finds a long v-cycle, where v is a given vertex of degree two. This algorithm can be used to find a long xy-path, by adding a new vertex v with edges vx, vy. Letting x and y vary, we can approximate the longest path in the graph. Similarly we can approximate the longest cycle.

Extending v-cycles. The overall approach is due to Björklund and Husfeldt [3]. They use a cycle to find a long path. Since our algorithm seeks a long cycle rather than a long path, the approach must be modified. This section presents the combinatoric ideas on which our algorithm is based.

The setting for our algorithm is illustrated in Figure 1: C is a v-cycle. X is a connected component of $G - V(C)$. We sometimes write $G[X]$ to emphasize that we are dealing with the graph induced by vertices X. P is an a_0a_1-path through X, more precisely for distinct vertices $a_0, a_1 \in C$, $P = a_0, P[x_0,x_1], a_1$ with $P[x_0,x_1] \subseteq X$.

Here is how the figure relates to the algorithm of the next section. Let C^* be a longest v-cycle. C will be a v-cycle already found by the algorithm. The algorithm is attempting to enlarge C to a longer v-cycle. P will be either a path that our algorithm has found recursively (in Lemma 2.1(i) below) or a subpath of C^* (in Lemma 2.1(ii)).

We use two main techniques to enlarge C. The first is similar to Björklund and Husfeldt's idea to use cycles to extend paths.

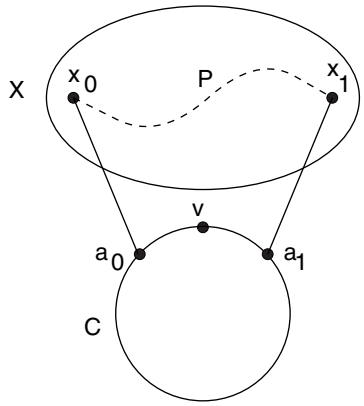


FIG. 1. *A v-cycle C and an a_0a_1-path P passing through a connected component X of $G - V(C)$.*

LEMMA 2.1. *Consider a v-cycle C and a connected component X of $G - V(C)$. Let P be an a_0a_1-path with $P(a_0, a_1) \subseteq X$.*

(i) *$P, C_v[a_1, a_0]$ is a v-cycle of length $> |P| + d_C(a_0, v)$.*

(ii) *For any vertex $c \in C$ adjacent to X, there is a path $Q \subseteq X$ and a vertex $c' \in C - c$ such that c, Q, c' is a cc'-path of length $\geq |P|/2 + 1$.*

Proof. (i) Note $|C_v[a_1, a_0]| > d_C(a_0, v)$ since a_1 is distinct from the degree two vertex v.

(ii) Let P start with the edge a_0x_0 and end with edge x_1a_1. So $P(a_0, a_1) = P[x_0, x_1] \subseteq X$. Choose $x \in X$ a neighbor of c as follows. If $c = a_i$ for $i \in \{0, 1\}$, then $x = x_i$. Otherwise x is arbitrary.

Take any minimal path R from x to P in X. Let R end at vertex $r \in P$. Choose index $j \in \{0, 1\}$ so that $|P[r, x_j]| \geq |P[r, x_{1-j}]|$. (One of these two subpaths of P actually involves the reverse of P.) The desired path c, Q, c' is $c, R, P[r, x_j], a_j$. This walk is simple since the choice of x guarantees $c \neq a_j$. The path's length is $\geq 1 + (|P| - 2)/2 + 1 = |P|/2 + 1$. □

When at least one distance $d_C(a_i, v)$ is large, part (i) allows us to make good progress in enlarging C. When both distances $d_C(a_i, v)$ are small another method is needed. The idea, illustrated in Figure 2, is to take two recursively found paths P (e.g., the two paths P in Figure 2) and find a v-cycle that contains both paths. We use the following strategy to guarantee that the desired v-cycle exists: for two components X of $G - V(C)$ we find a separation pair r_0, r_1 of G (e.g., the two separation pairs r_0, r_1 in Figure 2) with both vertices r_0, r_1 contained in C^*. Each path P will be a (recursively found) r_0r_1-path in X. No matter what r_0r_1-paths P the algorithm chooses, they will be contained in a v-cycle (since portions of C^* provide such a v-cycle). Of course the algorithm does not know C^*, so some guessing will be involved in finding the right separation pairs. Furthermore the separation pairs need not even exist. But once some other cases have been treated the existence of the separators will be guaranteed.

The next several lemmas give tools that allow us either to enlarge C or to find these separation pairs r_0, r_1. Our discussion uses connected components, biconnected components, and a form of triconnected components. For clarity we use terminology that explicitly differentiates all these types of components.

Bicomponents. A *bicomponent* is the set of edges of a biconnected component. The following characterization is often used as the definition of a biconnected component (see e.g., [17]).

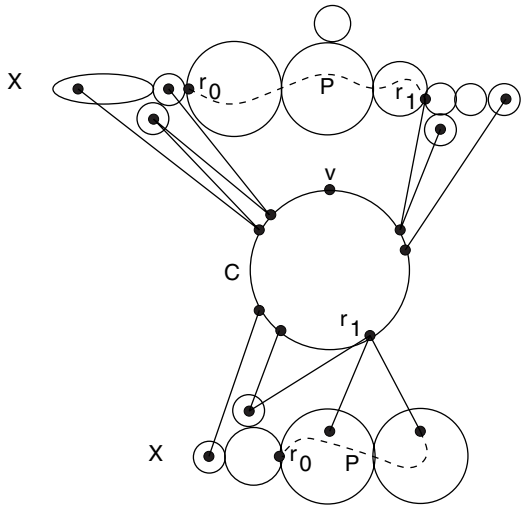


FIG. 2. *A v-cycle C with two connected components X of $G - V(C)$. Each component X has a separation pair r_0, r_1 and a corresponding r_0r_1-path P.*

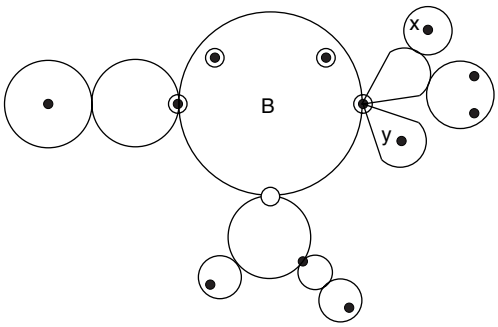


FIG. 3. *A bicomponent B and twelve vertices (drawn solid) that determine five distinct projections $\pi_B(v)$ (circled or drawn hollow).*

FACT 1. *Two edges belong to the same bicomponent if and only if some cycle contains both of them.*

We are interested in when a bicomponent B separates two vertices. By definition, the edge set B separates vertices x and y if every xy-path includes an edge of B. Note that it is possible for two vertices to be separated by a vertex of $V(B)$ but not separated by B, e.g., x and y in Figure 3.

The following notation, illustrated in Figure 3, elucidates the concept. Let B be the union of one or more bicomponents forming a connected subgraph. (In most of our applications B will be a single bicomponent, as in Figure 3.) For any vertex v, the *projection of v onto B*, denoted $\pi_B(v)$, is the vertex of $V(B)$ that is the end of every minimal path from v to $V(B)$ ("minimal" means that no vertex but the last one belongs to $V(B)$). For example, a vertex $v \in V(B)$ has $\pi_B(v) = v$.

Assuming the graph is connected, any projection $\pi_B(v)$ is unique. In proof, suppose not. So there are two minimal paths from v to $V(B)$, say P_1 and P_2, with P_i ending in vertex $b_i \in V(B)$ and $b_1 \neq b_2$. Let Q be a b_1b_2-path in B. $P_1 \cup Q \cup P_2$

contains a cycle C that includes edges belonging to B as well as edges not belonging to B. So the same holds for some bicomponent B_0 contained in B, i.e., C includes edges belonging to B_0 as well as edges not belonging to B_0. But this contradicts Fact 1.

Projection gives this alternate characterization of separation: v and w are separated by B if and only if their projections are distinct, $\pi_B(v) \neq \pi_B(w)$. We use the characterization only when B is a bicomponent, but the following proof of the characterization holds for B a connected union of bicomponents too: If $\pi_B(v) = \pi_B(w)$, then by definition there is a vw-path avoiding B; i.e., B does not separate v and w. To prove the opposite direction a vw-path avoiding B gives a $\pi_B(v)\pi_B(w)$-walk avoiding B. In general any xy-walk avoiding B and having $x, y \in V(B)$ has $x = y$ (by Fact 1). So $\pi_B(v) = \pi_B(w)$.

We now give some basic properties of separation by a bicomponent. Say that vertex s *weakly separates* sets $A, B \subseteq V$ if every path from A to B contains s. It is possible that s belongs to A or B.

LEMMA 2.2. *Let G be connected, and let B be a bicomponent.*

(i) *Any two distinct vertices in $V(B)$ are separated by B.*

(ii) *Any two distinct vertices x, y are separated by B if and only if every xy-path contains an edge of B if and only if some xy-path contains an edge of B.*

(iii) *For any set of vertices W, either B separates some two vertices of W, or some vertex of $V(B)$ weakly separates W and $V(B)$.*

Proof. (i) and (ii) follow easily from the characterization of separation in terms of projections. For (iii) note that if B does not separate any two vertices of W, then every $w \in W$ has the same projection $\pi_B(w)$ (by (ii)). By definition, this vertex $\pi_B(w)$ weakly separates W and $V(B)$. □

For any real value $b > 2$, a bicomponent is *b-round* if it contains a cycle of $\geq b$ edges. $D(x, y)$ denotes the length of a longest xy-path.

LEMMA 2.3. *Consider a connected graph and two distinct vertices x, y.*

(i) *If x and y are separated by a b-round bicomponent, then $D(x, y) \geq b/2$.*

(ii) *If $b = D(x, y)/d(x, y) > 2$, then x and y are separated by a b-round bicomponent.*

Remark. (ii) is consistent with x and y being vertices of the same biconnected component.

Proof. (i) Let B be a bicomponent containing a cycle A of length $\geq b$. First suppose that x and y are distinct vertices of $V(B)$. Biconnectedness implies there are distinct vertices $r, s \in A$ with an xr-path and a ys-path that are disjoint and also both disjoint from A until their last vertex. (This fact is well known [19, Exercise 4.2.9]. It is also easy to see by adding edge xy to the graph if it is not already present and applying Fact 1 to xy and an edge of A.) Piece these paths together to form an xy-path of length $\geq b/2$. (Specifically, follow the xr-path, then a path $A[r, s]$ of length $\geq b/2$, and then the ys-path reversed.)

Now suppose that x and y are arbitrary vertices separated by B. Take an xy-path. It contains an $x\pi_B(x)$-subpath and a $y\pi_B(y)$-subpath. We have $\pi_B(x) \neq \pi_B(y)$, so the subpaths are disjoint. Form the desired xy-path from the two subpaths plus the $\pi_B(x)\pi_B(y)$-path of length $\geq b/2$ constructed above (this path is contained in B, so our overall path is simple).

(ii) We'll prove a slightly more general statement: $D(x, y) > d(x, y)$ implies that x and y are separated by a $(D(x, y)/d(x, y) + 1)$-round bicomponent.

Let L (S) be a longest (shortest) xy-path. Let the vertices of $L \cap S$ be $x = s_0, s_1, \ldots, s_r = y$, ordered as they occur in S. (This need not be their order in L.) For each consecutive pair s_i, s_{i+1}, paths $L[s_i, s_{i+1}]$ and $S[s_i, s_{i+1}]$ are internally

vertex-disjoint. ($L[s_i, s_{i+1}]$ may actually be a subpath of the reverse of L.) The various paths $L[s_i, s_{i+1}]$ can share vertices and edges, but certainly some $L[s_j, s_{j+1}]$ has length $\geq |L|/r \geq |L|/|S|$. Thus $L[s_j, s_{j+1}], S[s_{j+1}, s_j]$ is a cycle A. (A is simple, and its length is $\geq 1 + \lceil |L|/|S| \rceil \geq 3$.) A is contained in some bicomponent B. A makes B $(|L|/|S| + 1)$-round. Since S contains an edge of B, B separates x and y. □

Finding separation pairs. Recall that two edges are *independent* if all four endpoints are distinct. A set of edges forms a *star* if some vertex (its *center*) is incident to each of the edges. For a star that is a single edge we choose the center as follows: the stars in this paper are always subsets of edge sets $E[X, Y]$, where X and Y partition the vertex set; we always choose the center to be in the set named X. For an edge $e \in E[X, Y]$, let $X(e)$ denote the endpoint of e in X. For $F \subseteq E[X, Y]$, $X(F)$ denotes $\{X(e) : e \in F\}$.

LEMMA 2.4. *Let the vertex set of graph G be partitioned into sets X and Y, with $G[X]$ connected. Let B be a bicomponent of $G[X]$, and let $F = E[X, Y]$ with F nonempty.*

Suppose that no two independent edges $e_1, e_2 \in F$ have their ends $X(e_1), X(e_2)$ separated by B, in graph $G[X]$. Then either F is a star or no two vertices of $X(F)$ are separated by B.

Proof. Suppose that F is not a star. Our goal is to prove that every vertex $x \in X(F)$ has the same projection $\pi_B(x)$. (All projections in this argument are calculated in graph $G[X]$.)

The supposition on F implies that there are independent edges $xy, x'y' \in E[X, Y]$. (A quick way to see this is by König's theorem: in a bipartite graph a maximum matching and a minimum vertex cover have the same size.) The lemma's hypothesis implies $\pi_B(x) = \pi_B(x')$. Consider any edge $zw \in E[X, Y]$ with $z \neq x, x'$. This edge is independent with either xy or $x'y'$ (or both). Hence the lemma's hypothesis implies $\pi_B(z) = \pi_B(x')$. □

The next two lemmas are used by the algorithm to find the separation pairs r_0, r_1 described earlier (recall Figure 2). Each vertex r_i is found separately. The first of the lemmas shows how to find one vertex r_i. The construction is based on a function $\Pi(F, x)$ that we now define.

We start by extending the projection operation, as follows. Consider a connected graph G and a set S of at least two vertices. Let B denote the union of all bicomponents that separate some two vertices of S. We'll use this alternate characterization of B: let T be a minimal tree of G that spans S. Minimality means that every leaf belongs to S. Then B is the union of all bicomponents that contain an edge of T. This characterization of B follows from Lemma 2.2(ii).

For any vertex v define the projection $\pi_S(v)$ to be $\pi_B(v)$, i.e., the vertex of $V(B)$ that is the end of every minimal path from v to $V(B)$. The construction gives these two properties:

(a) For any vertex v, $\pi_S(v)$ weakly separates S from v.

(b) For any vertices $v \in V$ and $s \in S$, the bicomponents that separate two vertices of S collectively contain an $s\pi_S(v)$-path.

Now consider a graph G whose vertices are partitioned into sets X and Y, with $G[X]$ connected. For any nonempty set of edges $F \subseteq E[X, Y]$ and any vertex $x \in X$, define the *near separator* $\Pi(F, x)$ as follows:

(i) if F is a star, then $\Pi(F, x)$ is its center;

(ii) if F is not a star, then $\Pi(F, x) = \pi_{X(F)}(x)$ (where the latter is calculated in $G[X]$).

In (i) if F is just a single edge, then recall that our convention chooses $\Pi(F,x)$ to be the vertex $X(F)$. In (ii) note that $|X(F)| \geq 2$. Hence $\pi_{X(F)}(x)$ is defined.

Continuing with this G, X, and Y, for any real value $b > 2$ call a set of edges $F \subseteq E[X,Y]$ *b-close* if no two independent edges $e_1, e_2 \in F$ have their ends $X(e_1), X(e_2)$ separated by a b-round bicomponent in $G[X]$. Observe that if F is b-close and B is a b-round bicomponent of $G[X]$, then Lemma 2.4 applies. It shows that either F is a star or no two vertices of $X(F)$ are separated by B.

LEMMA 2.5. *Let the vertex set of graph G be partitioned into sets X and Y, with $G[X]$ connected. Let $F = E[X,Y]$ be b-close for some $b > 2$.*

Let P be an x_0x_1-path in $G[X]$, with $x_0 \in X(F)$. Suppose $|P| \geq d_X(x_0,x_1)b > 0$.

Then either F is a star or the near separator $r = \Pi(F,x_1)$ has these properties: $r \in P$, r weakly separates $X(F)$ from $V(P(r,x_1])$ in graph $G[X]$, and $|P[x_0,r]| < d_X(x_0,x_1)b$.

Remark. The lemma is illustrated four times in Figure 2. First discard all edges incident to C except the four in the upper left. These four edges constitute set F. Extend P of the figure to the left so it begins at one of the three vertices of $X(F)$. Now r_0 is vertex r of the lemma. Similarly for r_0 in the lower left. In the upper right, r_1 is r of the lemma, this time a weak separator. The lower right illustrates the case of F a star.

Proof. Suppose that F is not a star. Property (a) above shows that r weakly separates $X(F)$ from x_1. Hence $r \in P$. Furthermore (a) shows that r weakly separates $X(F)$ from $V(P(r,x_1])$.

It remains only to deduce the inequality of the lemma. Clearly it suffices to show these two inequalities:

$$|P[x_0,r]| < d_X(x_0,r)b, \qquad d_X(x_0,r) \leq d_X(x_0,x_1).$$

The second inequality is true because r weakly separates x_0 and x_1. For the first inequality, property (b) above shows there is an x_0r-path within the bicomponents that separate two vertices of $X(F)$. None of these bicomponents is b-round. (As mentioned in the definition of b-closeness, this follows from Lemma 2.4.) So no b-round bicomponent separates x_0 and r. Now Lemma 2.3(ii) implies $|P[x_0,r]| < d_X(x_0,r)b$. □

Recall that two vertices a, b in a biconnected graph G form a *separation pair* if $G - \{a,b\}$ is disconnected. In that case an *a,b-tricomponent* T is a maximal set of edges, any two of which are joined by a path that does not contain a or b internally [12]. Alternatively, T is a connected component of $G - \{a,b\}$ plus the edges from the component to a or b. (Note that T needn't be 3-connected.) The next lemma shows how we combine two near separators r_0, r_1 of the previous lemma into a separation pair.

LEMMA 2.6. *Let G be a biconnected graph whose vertex set is partitioned into sets X and Y, with $G[X]$ connected. Let $E[X,Y] = F_0 \cup F_1$ with $Y(E[X,Y]) \neq Y$. For some $b > 2$ let both sets F_i be b-close.*

Let x_i, $i = 0,1$, be distinct vertices with $x_i \in X(F_i)$. Let P be an x_0x_1-path in $G[X]$. Suppose $|P| \geq 2d_X(x_0,x_1)b + 2$. For $i = 0,1$ let r_i be the near separator $\Pi(F_i, x_{1-i})$. Then

(i) *r_0, r_1 is a separation pair of G.*

(ii) *For $i = 0,1$ if F_i is a star centered in $r_i \in Y$, then extend P by the edge x_i, r_i. Let $\overline{P}$ be the resulting path. $\overline{P}$ contains r_0 and r_1, and $\overline{P}[r_0,r_1]$ is contained in an r_0r_1-tricomponent that is contained in $G[X \cup \{r_0,r_1\}]$.*

Remark. $\overline{P}$ is equal to either P (if both $r_i \in X$), or P plus an edge at one end (if exactly one $r_i \in Y$), or P plus an edge at both ends (if both $r_i \in Y$).

Proof. Note that Lemma 2.5 applies to both sets F_i. Note also that in (ii), $\overline{P}$ is a path, even if both vertices r_0, r_1 belong to Y. This follows from the biconnectivity of G. It is clear that in all cases $\overline{P}$ contains both r_0 and r_1.

The argument treats P and $\overline{P}$ as oriented paths. In particular, the notation $\overline{P}[a,b]$ is well formed only if a precedes b in P. Take a vertex $x \in \overline{P}(r_0, r_1)$. (More precisely, x must occur after r_0 and before r_1.) We claim that such an x exists. In particular this claim implies that r_0 precedes r_1 in $\overline{P}$.

To prove the claim first observe that $r_i \in X$ implies

$$|P[x_i, r_i]| < d_X(x_0, x_1)b.$$

If F_i is not a star, this inequality follows from Lemma 2.5. If F_i is a star, then r_i is its center, so $r_i \in X$ implies $r_i = x_i$ and $|P[x_i, r_i]| = 0 < d_X(x_0, x_1)b$.

Hence if both $r_i \in X$, the displayed inequality plus the hypothesized lower bound on $|P|$ imply $|P[r_0, r_1]| \geq 2$; i.e., r_0 precedes r_1 in P and $V(P(r_0, r_1)) \neq \emptyset$, as desired (recall that $\overline{P} = P$ in this case). On the other hand, if some $r_i \in Y$, then we can simply take x to be the vertex x_i. It is easy to check that $x_i \neq r_{1-i}$, again using the hypothesized bound on $|P|$. The claim now follows in all cases.

We prove (i) by showing that r_0, r_1 separates vertex x and $Y - \{r_0, r_1\}$. Let Q be a path from $Y - \{r_0, r_1\}$ to x; we must show that Q contains r_0 or r_1. To get from Y to X, Q contains an edge of $E[Y, X]$. Let $yz \in E[Y, X]$ be the last such edge in Q. For definiteness assume $yz \in F_0$; we will show $r_0 \in V(Q)$. If F_0 is a star, then clearly Q contains its center, which is r_0. Suppose that F_0 is not a star. Lemma 2.5 shows that r_0 weakly separates $z \in X(F_0)$ and $x \in V(P(r_0, x_1])$ in graph $G[X]$. The choice of yz ensures that $Q[z, x]$ is contained in X. Hence $r_0 \in V(Q[z, x])$.

It is easy to see that we have also proved (ii). In particular the last part of (ii) follows since $x \in \overline{P}(r_0, r_1)$. □

Here's how the algorithm uses the separation pairs r_0, r_1 (recall Figure 2). Say that a path P *traverses* a subgraph H if P contains a subpath of ≥ 3 edges, where the first and last subpath edge do not belong to H but all other subpath edges do belong to H. Let a, b be a separation pair. Clearly any path P traversing an a, b-tricomponent T contains both a and b. Furthermore P traverses T only once; i.e., $E(P) \cap T = E(P[a, b])$.

Let C^* be a v-cycle. Let r_0, r_1 be a separation pair of G contained in C^*, and abbreviate $C^*_v[r_0, r_1]$ to C^*_1. Let T_1 be the r_0, r_1-tricomponent of G that contains C^*_1. Thus C^* traverses T_1. Let Q_1 be an arbitrary $r_0 r_1$-path contained in T_1. Next define C^*_2, T_2, and Q_2 similarly from a second separation pair r'_0, r'_1 that is also contained in C^*. Assume that C^*_1 and C^*_2 are edge-disjoint; e.g., T_1 and T_2 are edge-disjoint.

The *gluing principle* states that regardless of the choice of Q_i, a v-cycle containing paths Q_1 and Q_2 exists. In proof, the edge set $A = (C^* - \cup_{i=1}^2 C^*_i) \cup \cup_{i=1}^2 Q_i$ is such a cycle. This hinges on the fact that A is guaranteed to be simple, because C^* traverses each tricomponent T_1, T_2 only once.

The algorithm will identify the above separation pairs r_0, r_1 and r'_0, r'_1 and the corresponding tricomponents T_1 and T_2. The algorithm will find its own $r_0 r_1$-path Q_1 and $r'_0 r'_1$-path Q_2 (recursively). The gluing principle guarantees a v-cycle through Q_1 and Q_2. We shall see that such a cycle can be found efficiently.

The existence of the separating pairs r_0, r_1 will be guaranteed by applying Lemma 2.6. The lemma's $x_0 x_1$-path P will be a subpath of C^*, and $\overline{P}[r_0, r_1]$ will be the above

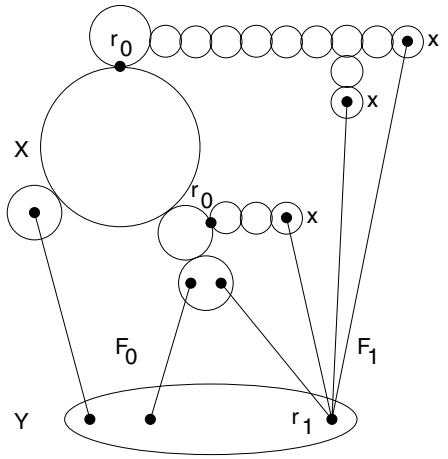


FIG. 4. *Finding a tricomponent cover on X. In this figure F_0 is not a star, F_1 is a star centered at $r_1 \in Y$, and there are two possible vertices $r_0 = \Pi(F_0, x)$.*

path C_1^* (or C_2^*). Since the algorithm doesn't know P, x_0, or x_1, it doesn't know the separation pair r_0, r_1 (since r_i is defined in terms of x_i). Similarly the algorithm doesn't know the corresponding tricomponent containing $\overline{P}[r_0, r_1]$. To compensate, the algorithm finds a set of tricomponents that includes the desired tricomponent containing $\overline{P}[r_0, r_1]$. The set is captured in the following definition.

Consider sets G, X, Y, F_0, and F_1 of Lemma 2.6. As in the lemma, the sets F_i are b-close. A *tricomponent cover* on X is a collection $\mathcal{T}$ of triplets of the form (r_0, r_1, T) satisfying the following conditions:

(i) r_0, r_1 is a separation pair of G with T an r_0r_1-tricomponent, where $r_0, r_1 \in X \cup Y(E[X, Y])$ and $V(T) \subseteq X \cup \{r_0, r_1\}$.

(ii) The edge sets T of $\mathcal{T}$ are pairwise disjoint.

(iii) Let P be any x_0x_1-path satisfying the conditions of Lemma 2.6. Then the lemma gives the separation pair r_0, r_1. $\mathcal{T}$ must include a triplet (r_0, r_1, T), where, as in the lemma, $r_i = \Pi(F_i, x_{1-i})$ for $i = 0, 1$ and T is the r_0r_1-tricomponent containing $\overline{P}[r_0, r_1]$.

$\mathcal{T}$ may include other triplets besides those required by (iii).

Characterizing a tricomponent cover depends on how many sets F_0, F_1 are stars. Lemma 2.7 below is the heart of the characterization. Part (ii) of the lemma, which handles the case of exactly one star, is illustrated in Figure 4.

LEMMA 2.7. *Suppose that an application of Lemma* 2.6 *determines the separation pair r_0, r_1 from G, X, Y, F_0, F_1, and P. Suppose that F_0 is not a star.*

(i) *Suppose that F_1 is also not a star. Then every vertex $x \in X(F_1)$ determines the same near separator; i.e., $\Pi(F_0, x) = r_0$.*

(ii) *Suppose that F_1 is a star and its center r_1 belongs to Y. For two vertices $x_j \in X(F_1)$, $j = 1, 2$, let $\Pi(F_0, x_j), r_1$ be a separation pair with T_j a corresponding tricomponent such that $x_j \in V(T_j) - \Pi(F_0, x_j)$. Then T_1 and T_2 are either identical or edge-disjoint.*

Proof. Recall that the notion $\Pi(F, x)$ is defined in terms of the notion $\pi_S(v)$, whose definition in turn is based on the set B of bicomponents that separate two vertices of S.

(i) For $i = 0, 1$ let B_i be the set of bicomponents used to define $\Pi(F_i, \cdot)$. Observe that $r_0 \notin V(B_1)$. To see this recall that, by definition, $r_1 = \Pi(F_1, x_0)$ is the first

vertex of the path $P[x_0, r_1]$ that belongs to $V(B_1)$. The proof of Lemma 2.6 shows that r_0 precedes r_1 in P. Hence $r_0 \notin V(B_1)$.

Any $x \in X(F_1)$ is joined to x_1 by a path Q contained in $V(B_1)$ (by construction). Q avoids $V(B_0)$. To see this assume the contrary. Then Q contains a path from x_1 to $V(B_0)$. Hence Q contains r_0, since r_0 is defined as $\Pi(F_0, x_1)$. But we have just seen that this is not the case.

Combining Q and $P[r_0, x_1]$ gives a path from x to r_0 whose first vertex in $V(B_0)$ is r_0. So, by definition, $r_0 = \Pi(F_0, x)$.

(ii) The lemma is obvious if $\Pi(F_0, x_1) = \Pi(F_0, x_2)$. So assume $\Pi(F_0, x_1) \neq \Pi(F_0, x_2)$. To treat this case it suffices to show that in general, for any separation pair $\Pi(F_0, x), r_1$ with tricomponent T having $x \in V(T) - \Pi(F_0, x)$, every vertex $v \in V(T) - r_1$ has the same near separator $\Pi(F_0, v) = \Pi(F_0, x)$.

Let B_0 be the set of bicomponents used to define $\Pi(F_0, \cdot)$. Let Q be a path from v to x that avoids $\Pi(F_0, x)$ and r_1. (Q exists by the definition of tricomponent.) Let Q' be a path from x to $\Pi(F_0, x)$ that contains no vertex of $V(B_0)$ besides $\Pi(F_0, x)$. (Q' exists by the definition of $\Pi(F_0, x)$.) Combining Q and Q' gives a path from v to $\Pi(F_0, x)$ containing no vertex of $V(B_0)$ besides $\Pi(F_0, x)$. This shows $\Pi(F_0, v) = \Pi(F_0, x)$. □

Using the lemma, a tricomponent cover $\mathcal{T}$ on X can be found as follows. Let P be an x_0x_1-path satisfying the conditions of Lemma 2.6, and let (r_0, r_1, T_0) be the corresponding triplet that must be included in $\mathcal{T}$. Consider the following three cases that, taking into account the symmetry between F_0 and F_1, cover all possibilities.

Case 1. Neither set F_i is a star. We construct $\mathcal{T}$ containing just one triplet, the desired one: Lemma 2.7(i) shows how to identify the two separating vertices r_0, r_1. T_0 is the (unique) r_0r_1-tricomponent that is contained in X (Lemma 2.6(ii)).

Case 2. F_1 is a star centered in X, or both F_0 and F_1 are stars. Both near separators are known: the near separator of a star is its center. If F_0 is not a star and F_1 is a star centered in $r_1 \in X$, then Lemma 2.6 shows $r_0 = \Pi(F_0, r_1)$.

In $\mathcal{T}$ the tricomponents T that correspond to the separation pair r_0, r_1 range over all the r_0r_1-tricomponents whose vertices are in $X \cup \{r_0, r_1\}$. There may be any number of such tricomponents. However, these tricomponents are clearly edge-disjoint, and one of them is the desired tricomponent T_0 containing $\overline{P}[r_0, r_1]$.

Case 3. F_1 is a star centered in Y and F_0 is not a star. (This case is illustrated in Figure 4.) Construct $\mathcal{T}$ as follows. r_1 is the center of star F_1. For each vertex $x \in X(F_1)$, set $r_0 = \Pi(F_0, x)$. If $r_0 \neq x$ and r_0, r_1 is a separation pair, then let T be the corresponding tricomponent containing x and add (r_0, r_1, T) to $\mathcal{T}$.

Lemma 2.7(ii) shows that all tricomponents so constructed are pairwise edge-disjoint.

To show that $\mathcal{T}$ contains the required triplet for P, recall that the proof of Lemma 2.6 shows $x_1 \neq r_0$ (where x_1 is the last vertex of P). In other words, $x_1 \neq \Pi(F_0, x_1)$. Thus $\mathcal{T}$ contains $(\Pi(F_0, x_1), r_1, T_0)$, as required.

The algorithm of the next section constructs tricomponent covers on a number of vertex sets X. These sets X are pairwise vertex-disjoint. (Specifically each X is a connected component of $G - V[C]$, where C is the algorithm's current cycle.) It is clear that property (ii) of the definition of tricomponent cover extends to this context: any two edge sets T from the same or different tricomponent covers $\mathcal{T}$ are disjoint.

We end this section with an observation that is not needed for the logical development but may prevent misconceptions. It is possible that two edge-disjoint subpaths of C^* both satisfy the hypothesis of Lemma 2.6 and both traverse the same set X (where, as just mentioned, X is a connected component of $G - V[C]$). This can oc-

cur in Case 3 above. For instance, in Figure 4, C^* enters X on the left edge of F_0, goes through the upper r_0 vertex and the upper row of bicomponents to an edge xr_1, then goes through the lower row of bicomponents to the lower r_0 vertex, and finally proceeds to the right edge of F_0.

3. Algorithm. This section presents an algorithm to approximate the longest cycle C^* through a given vertex v of degree two in a given graph G. Sections 4–5 give the analysis (as well as some lower level details of the algorithm in section 5).

Let ℓ be the length of C^*. For every integer $p \geq 1$ we give an algorithm $\mathcal{A}_p$ that finds a v-cycle of length $\Omega((\log \ell / \log\log \ell)^p)$. $\mathcal{A}_p$ uses $\mathcal{A}_{p-1}$ as a subroutine, and the hidden constant decreases with p. We choose an appropriate value p^* of p to get the desired overall result on superpolylogarithmic cycles.

For brevity the presentation of $\mathcal{A}_p$ is not optimized. Slight asymptotic improvements are possible by using more detailed versions of $\mathcal{A}_1$ and $\mathcal{A}_2$. Also we make no attempt to keep the constants small, preferring instead to keep the arithmetic simple.

We will guess a value ℓ^* as the length of C^*. Write

$$a^* = \log \ell^*, \qquad k^* = \log a^* = \log\log \ell^*.$$

For $p \leq p^*$, algorithm $\mathcal{A}_p$ will be given a smaller graph derived from the given one, along with a degree two vertex v. When describing $\mathcal{A}_p$ we will use C^* to denote the longest v-cycle in this recursive call. $\mathcal{A}_p$ uses variables a and k that play roles similar to the roles of a^* and k^* above. The formulas for these quantities are

$$a = \frac{pa^*}{p^*}, \qquad k = k^*. \tag{1}$$

The values of a decrease slowly from a^* as p decreases (in recursive calls). This allows us to find the longest v-cycle possible. The reader should think of variable k as $\log a$, in analogy with the definition of k^*. However since $\log a$ is difficult to compute, we actually define k to be the slightly larger value k^*.

Algorithm $\mathcal{A}_p$ has this length guarantee: given a graph with a v-cycle of length $\geq 2^a$ for a defined by (1), $\mathcal{A}_p$ returns a v-cycle of length at least

$$\alpha_p \left(\frac{a}{k}\right)^p.$$

Here $\alpha_p \leq 1$ is a factor depending on p that we will derive. The length guarantee is proved in Lemma 4.1.

We start with a driving routine. It ensures that a is large, in all calls to routines $\mathcal{A}_{p^*}, \mathcal{A}_{p^*-1}, \ldots, \mathcal{A}_1$.

Main Routine.

Step 1. If $|C^*| \leq 2^{2^8}$, use color coding to find a longest v-cycle, and return it.

Step 2. For k^* taking on consecutive integral values from 8 to $\lfloor \log\log n \rfloor$ set $a^* = 2^{k^*}$, $p^* = \lfloor \sqrt{a^*/24k^*} \rfloor$, and call $\mathcal{A}_{p^*}$. Return the longest v-cycle found by any of these routines.

To implement Step 1 note that for any integer k, color coding can find the longest v-cycle having length $\leq k$ in time $2^{O(k)} n \log n$. Applying this with $k = 2^{2^8}$ almost accomplishes Step 1. But it doesn't suffice, since it doesn't prove $|C^*| \leq k$. In [11] color coding is supplemented with the undirected cycle structure theorem stated in section 1 to accomplish Step 1. Specifically [11] shows that for any undirected graph and any integer k, a v-cycle of length $\geq k$ can be found in time $O(m) + 2^{O(k)} n \log n$, if one exists. This gives a complete implementation of Step 1.

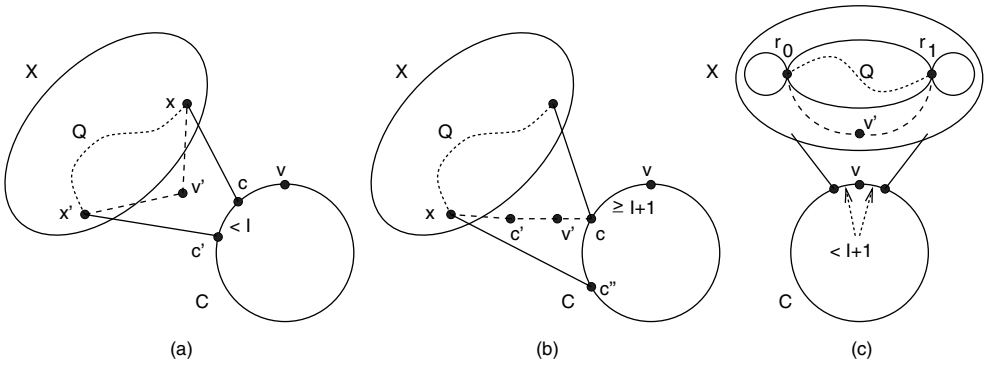


FIG. 5. *Recursive calls of* $\mathcal{A}_{p+1}$ *(see next page):* (a) *Step* 2.1 *calls* $\mathcal{A}_p$*;* (b) *Step* 3.1 *and* (c) *Step* 5.3 *call* $\mathcal{A}_{p+1}$.

Algorithm $\mathcal{A}_1$ is again implemented using the algorithm of [11]. It can find a v-cycle of length $\min\{|C^*|, \log n\}$ in polynomial time. This implies that our length guarantee for $\mathcal{A}_1$ is satisfied with $\alpha_1 = 1$, since clearly $a \le \log n$.

For $p^* > p \ge 1$ Algorithm $\mathcal{A}_{p+1}$ uses $\mathcal{A}_p$. We give an overview of $\mathcal{A}_{p+1}$ before stating it precisely. $\mathcal{A}_{p+1}$ begins by using $\mathcal{A}_p$ to find a v-cycle C. Then $\mathcal{A}_{p+1}$ recurses on each connected component X of $G - V(C)$, to find a long path P illustrated in Figure 1. The idea is to enlarge one of these recursively found paths P, using C or another recursively found cycle, to get a longer cycle $\overline{C}$. Specifically each level of recursion in $\mathcal{A}_{p+1}$ enlarges the cycle (or path) of the previous level by the additive "increment" I defined by

$$\widetilde{a} = \frac{pa}{p+1}, \qquad I = \frac{\alpha_p}{2}\left(\frac{\widetilde{a}}{k}\right)^p. \tag{2}$$

Observe from (1) that $\widetilde{a}$ is the value of a in the recursive calls that $\mathcal{A}_{p+1}$ makes to $\mathcal{A}_p$. So recalling the length guarantee stated above, I is half the length of a recursively found cycle. The enlarged v-cycle $\overline{C}$ gets returned by $\mathcal{A}_{p+1}$.

$\mathcal{A}_{p+1}$ uses several different strategies to construct $\overline{C}$. The first strategy (Step 3 below) is the analogue of Björklund and Husfeldt's algorithm [3] (their algorithm also provides the organization described in the previous paragraph): we apply Lemma 2.1(i) (and Figure 1), using the recursive call to find a long path through X and combining this path with C to get $\overline{C}$. In Step 3 (and Figure 5(b)) the long path through X is denoted c, Q, c'', where $c, c'' \in C$ and $Q \subseteq X$. This method is effective if $d_C(c, v)$ is large.

The second major strategy involves using the gluing principle to piece together two recursively found paths into $\overline{C}$, as in Figure 2. We implement this strategy as follows. Partition the edges of C^* into maximal subpaths that are internally disjoint from C. (Recall that two paths are internally disjoint if any common vertex is an end of both paths.) Call each of these subpaths a *segment* of C^*. Both edges incident to v are themselves segments. In general a segment is either an edge or chord of C or a path traversing a connected component X of $G - V(C)$ (i.e., a path through X plus two connecting edges). Note that more than one segment may traverse a given component X.

In the next paragraph we apply Lemma 2.6 to a segment S that traverses a connected component X. This is interpreted as follows. In the lemma define Y as the vertex complement of X, and P as $S[x_0, x_1]$ for x_0 and x_1 the second and

penultimate vertices, respectively, of S (since the first and last vertices of S belong to C; the variables F_i and b of the lemma are defined in Step 5 below).

Now consider two "long" segments of C^* and their corresponding connected components X. We will apply Lemma 2.6 (to each X and its segment) to find a separation pair r_0, r_1 and corresponding r_0r_1-tricomponent T contained in $X \cup \{r_0, r_1\}$, where T contains a (long) portion of C^*, $C^*[r_0, r_1]$. We will use recursive calls to approximate $C^*[r_0, r_1]$. (This is done in Step 5.3, Figure 5(c).) To be precise for $i = 1, 2$ let X_i be the two components X, let C_i^* be the subpath $C^*[r_0, r_1]$ contained in X_i, and let Q_i be the r_0r_1-path returned by the recursive call (on the identified r_0r_1-tricomponent T). The gluing principle guarantees the existence of a cycle containing Q_1, Q_2, and v. The algorithm can find such a v-cycle using a routine for subgraph homeomorphism. This is done in Step 6.

To implement this strategy based on the separation pairs r_0, r_1 given by Lemma 2.6, we must ensure that the lemma's hypothesis holds. In particular we must ensure that F_0 and F_1 are b-close. This is done using Lemma 2.3(i), in Step 2 (Figure 5(a)). Then (in Step 5.2) we use Lemma 2.7 to find a tricomponent cover, thereby finding the desired separation pairs r_0, r_1.

A special case of the second strategy occurs when the first strategy is not applicable but C^* has only one long segment (so gluing is not applicable). As before, the long segment traverses a component X as in Figure 5(c). This possibility is easy to handle: it suffices to return a cycle constructed from the cycle found recursively in X (Step 7).

We have now sketched the major pieces of the algorithm. Because our two strategies use differing recursive calls (Figure 5(b)–(c)) the algorithm maintains numeric quantities like I to decide which strategy to use.

We proceed to give a high level statement of the algorithm. The details that are postponed to section 5 are indicated in the algorithm statement.

$\mathcal{A}_{p+1}$ has parameters G the graph, v the vertex of degree two, and ρ the recursion level. It is assumed that G has a v-cycle. Parameter ρ is used to prevent too many levels of recursion (which might violate the time bound). ρ equals 0 in the initial call.

The algorithm knows the values a and k of (1), $\widetilde{a}$ and I of (2). In addition it uses the values

$$b = 2^{\widetilde{a}+1}, \qquad g = \left(\frac{\widetilde{a}}{k}\right)^{p+1}. \tag{3}$$

The algorithm generates a number of v-cycles. It maintains $\overline{C}$ as the longest v-cycle ever generated, and eventually returns $\overline{C}$. If a cycle of length $\geq g$ is ever generated, that cycle is immediately returned.

ALGORITHM $\mathcal{A}_{p+1}(G, v, \rho)$. /* The algorithm also knows the values defined in (1)–(3): $a, k, \widetilde{a}, I, b, g$ */

Step 0. Initialize $\overline{C}$ to any v-cycle. If $\rho \geq a^{p+1}$, then return $\overline{C}$. Otherwise prune G to the (unique) biconnected component containing v. Let w_0, w_1 be the two neighbors of v in G. Thus $W = \{vw_0, vw_1\}$ is a w_0w_1-tricomponent. For every other w_0w_1-tricomponent U, execute all the steps below for the graph G' whose edge set is $U \cup W$. Then return $\overline{C}$.

Step 1. Find a v-cycle C by calling $\mathcal{A}_p(G', v, 0)$.

Step 2. Repeat Step 2.1 until either it makes $|C| \geq g$ or no further enlargement of C is possible. In the former case return $\overline{C} = C$.

Step 2.1 (This step is illustrated in Figure 5(a).) Suppose for some connected component X of $G' - V(C)$ that $E[C, X]$ contains independent edges $cx, c'x'$ such that $|C_{\overline{v}}[c, c']| < I$ and some b-round bicomponent of $G[X]$ separates x and x'. Call $\mathcal{A}_p$ to find an xx'-path Q in X. Specifically create a new vertex v' along with new edges $v'x, v'x'$, and call $\mathcal{A}_p(H, v', 0)$ for H the graph induced by $V[X] \cup \{v'\}$. In C replace $C_{\overline{v}}[c, c']$ by c, Q, c'. (The implementation of this step is given in section 5. Section 4 shows that this replacement enlarges C.)

In the rest of the algorithm the variable X ranges over every connected component of $G' - V(C)$. (There may not be any.)

Step 3. Execute Step 3.1 for each component X that has a neighbor $c \in C$ with $d_C(c, v) \geq I + 1$ (if X has more than one such neighbor choose c arbitrarily).

Step 3.1. (This step is illustrated in Figure 5(b).) Create two new vertices v', c' along with new edges $v'c, v'c'$, plus an edge $c'x$ for each vertex $x \in X$ that is a neighbor of $C - c$. Call $\mathcal{A}_{p+1}(H, v', \rho + 1)$ recursively for H, the graph induced by $V[X] \cup \{c, c', v'\}$. The cycle returned corresponds to a path c, Q, c'' in G', where $Q \subseteq X$ and $c'' \in C - c$. Update $\overline{C}$ for the v-cycle $c, Q, c'', C_v[c'', c]$ of G.

Step 4. For each component X, check whether $E[C, X]$ contains two independent edges $cx, c'x'$ that have $d_X(x, x') \geq g$. If so, let Q be an $x'x$-path in X and return the cycle $x', Q, x, C_v[c, c'], x'$.

Step 5. This step applies Lemma 2.6. Initialize a set of paths $\mathcal{Q}$ to $\emptyset$. For each component X not processed in Step 3 (i.e., every neighbor $c \in C$ of X has $d_C(c, v) < I + 1$) execute Steps 5.1–5.3. (Note that if every component X was processed in Step 3 then Steps 5–7 do nothing.)

Step 5.1. Partition $E[C, X]$ into sets F_0 and F_1, where F_0 contains the edges $cx \in E[C, X]$ with $d_C(c, w_0) \leq d_C(c, w_1)$ and F_1 contains the remaining edges. If F_0 or F_1 is empty, skip Steps 5.2–5.3 and proceed to the next component X.

Step 5.2. As described in section 2 use Lemma 2.7 to construct a tricomponent cover $\mathcal{T}$ on X. Then each triplet (r_0, r_1, T) of $\mathcal{T}$ has $r_0, r_1 \in X \cup C$ and $T \subseteq E(X) \cup E[C, X]$.

(Observe the following property of $\mathcal{T}$. Consider any segment of C^* that traverses X such that Lemma 2.6 guarantees a separation pair r_0, r_1 when applied to X and the portion of the segment in X. Then $\mathcal{T}$ contains (r_0, r_1, T), where T is the r_0r_1-tricomponent in $X \cup \{r_0, r_1\}$ containing $C^*_{\overline{v}}[r_0, r_1]$.)

Step 5.3. (This step is illustrated in Figure 5(c).) For each $(r_0, r_1, T) \in \mathcal{T}$, call $\mathcal{A}_{p+1}$ to find an r_0r_1-path Q. Specifically create a new vertex v' along with new edges $v'r_0, v'r_1$, and call $\mathcal{A}_{p+1}(H, v', \rho+1)$ recursively for H the graph induced by $V[T] \cup \{v'\}$. Add Q to $\mathcal{Q}$.

Step 6. For every pair of paths in $\mathcal{Q}$, search for a cycle containing v and the two paths. Update $\overline{C}$ for every cycle found. (The implementation of this step is given in section 5.)

Step 7. Let Q be a longest path of $\mathcal{Q}$. Extend Q to a v-cycle. (As shown below, this can be done since Q and v are in different r_0r_1-tricomponents). Update $\overline{C}$ for this cycle.

We leave the following implementation details to section 5: checking for the existence of the separating b-round bicomponent in Step 2.1, finding the desired cycle in Step 6, and computing all the numeric quantities like I, b, etc. The rest of the implementation of the algorithm is clear.

To clarify the difference in recursive calls to $\mathcal{A}_p$ and $\mathcal{A}_{p+1}$, observe that $\mathcal{A}_p$ essentially initializes C, in Steps 1 and 2. $\mathcal{A}_{p+1}$ processes C, in Steps 3 and 5.

We close this section by verifying that the algorithm is well specified. First note that each graph G' constructed in Step 0 is biconnected. This follows easily from Fact 1.

Recall that we assume $\mathcal{A}_{p+1}$ is called with a graph that has a v-cycle. It is easy to check that this property is maintained in all the recursive calls. In particular for Step 3.1 the biconnectivity of G' guarantees that X has neighbors in $C - c$.

Finally, we note that the v-cycle of Step 7 always exists. In proof, Step 5.2 shows that v does not belong to the r_0r_1-tricomponent T containing Q. The biconnectivity of G' and Fact 1 show there is an r_1r_0-path R containing v that is edge-disjoint from T. Now Q, R is the desired v-cycle for Step 7.

4. Length analysis. Before diving into the analysis we restate the high level purpose of Algorithm $\mathcal{A}_{p+1}$, for some intuition. Recall the length guarantee given for $\mathcal{A}_p$ at the start of last section. Step 1 initializes C to a v-cycle of length $\geq \alpha_p \left(\frac{\widetilde{a}}{k}\right)^p = 2I$. This can be proved by the same simple argument as given for Claim 2 below. The task for the rest of $\mathcal{A}_{p+1}$ is to enlarge the cycle to the length guarantee's quantity $\alpha_{p+1}\left(\frac{a}{k}\right)^{p+1}$. As mentioned in the last section, this is accomplished by having each level of recursion (of $\mathcal{A}_{p+1}$) increase the cycle length by $\geq I$. This fact is the heart of the analysis. It is stated formally as the Assertion of the Lemma 4.1 below.

We start the formal analysis with some inequalities that result from the basic parameters being large. Step 2 of the Main Routine has

$$k^* \geq 8, \qquad a^* = 2^{k^*} \geq 2^8, \qquad \ell^* = 2^{a^*} \geq 2^{2^8}.$$

Recall that algorithm $\mathcal{A}_p$ is called for all $1 \leq p \leq p^*$, and $\mathcal{A}_p$ uses the value $a = pa^*/p^*$. We claim that $\mathcal{A}_p$ always has

$$a \geq 2^4, \qquad a \geq 24kp^2. \tag{4}$$

For the first inequality the Main Routine's Step 2 shows $p^* \leq \sqrt{a^*}$. Hence $a \geq a^*/p^* \geq \sqrt{a^*} \geq 2^4$. For the second inequality write $a = pp^*a^*/(p^*)^2$. The Main Routine's Step 2 shows $p^* \leq \sqrt{a^*/24k^*}$, so we get $a \geq pp^*a^*/(a^*/24k^*) = 24k^*pp^*$.

We need to verify two inequalities to ensure that the algorithm makes sense: $p^* \geq 1$ (since the Main Routine's Step 2 calls $\mathcal{A}_{p^*}$) and $b > 2$ (since Step 2.1 looks for b-round components). The first inequality is equivalent to $\sqrt{a^*/24k^*} \geq 1$, i.e., $a^*/k^* \geq 24$. This holds since, remembering that $x/\log x$ is an increasing function for $x \geq e$, we have $a^*/k^* = a^*/\log a^* \geq 2^8/8 = 2^5$. The inequality $b > 2$ holds since $b = 2^{\widetilde{a}+1}$, $\widetilde{a} \geq 2^4$.

Define the decreasing sequence α_p for integers $p \geq 1$ by

$$\alpha_p = \frac{1}{24^{p-1}(p!)^2}.$$

Let us show that this implies that every algorithm $\mathcal{A}_{p+1}$, $p \geq 1$, has

$$I \geq 2. \tag{5}$$

By definition (2), $I = \frac{\alpha_p}{2}\left(\frac{\widetilde{a}}{k}\right)^p$. Since $\widetilde{a}$ represents the quantity a in algorithm $\mathcal{A}_p$, the second inequality of (4) amounts to $\widetilde{a} \geq 24kp^2$. Thus

$$I \geq \frac{1}{2 \cdot 24^{p-1}p^{2p}}\left(\frac{24kp^2}{k}\right)^p = 12 > 2.$$

The main task of this section is to prove the following length guarantee (already mentioned at the start of section 3).

LEMMA 4.1. *For any $p^* > p \geq 0$, suppose Algorithm $\mathcal{A}_{p+1}$ is called with graph G having a v-cycle of length $\geq 2^a$, and $\rho = 0$. Here a is defined by the general equation (1), i.e., $a = (p+1)a^*/p^*$. Then $\mathcal{A}_{p+1}$ returns a v-cycle of length $\geq \alpha_{p+1}(a/k)^{p+1}$.*

Proof. We induct on p. The base case $p = 0$ (i.e., Algorithm $\mathcal{A}_1$) was verified after the statement of the Main Routine in section 3. So assume that $p \geq 1$ and algorithm $\mathcal{A}_p$ fulfills the length guarantee. We prove that $\mathcal{A}_{p+1}$ also fulfills the length guarantee.

We begin by showing that the claim made in Step 2.1 is true, as follows.

CLAIM 0. *Every replacement done by Step* 2.1 *gives a longer cycle C.*

Proof. Applying Lemma 2.3(i) to the vertices x and x' of Step 2.1 shows that $D_X(x, x') \geq b/2 = 2^{\widetilde{a}}$. Thus the inductive assumption applies to $\mathcal{A}_p$ and shows it gives an xx'-path of length $\geq \alpha_p \left(\frac{\widetilde{a}}{k}\right)^p - 2 = 2I - 2$. Hence the length of C increases by $> (2I-2) + 2 - I = I > 0$. □

Let $\mathcal{R}$ be the recursion tree of $\mathcal{A}_{p+1}$. As usual, identify each node of $\mathcal{R}$ with the corresponding invocation of $\mathcal{A}_{p+1}$. By definition, $\mathcal{R}$ does not contain nodes for calls to $\mathcal{A}_p$. For any node τ of $\mathcal{R}$ let C_τ denote the cycle returned by τ.

CLAIM 1. *Any child σ of any node τ of $\mathcal{R}$ has $|C_\sigma| < |C_\tau|$.*

Proof. First suppose that σ is called from Step 3.1 of τ. Lemma 2.1(i) implies $|C_\tau| > (|C_\sigma| - 2) + (I + 1) > |C_\sigma|$.

Next suppose that σ is called from Step 5.3. It suffices to assume that C_σ corresponds to the longest path Q in $\mathcal{Q}$. Step 5.3 shows that $|C_\sigma| = |Q| + 2$. Step 7 shows that this quantity is $< |C_\tau|$ unless $\{r_0, r_1\} = \{w_0, w_1\}$, where Q is an r_0r_1-path and w_0, w_1 are the neighbors of v defined in Step 0. So suppose $\{r_0, r_1\} = \{w_0, w_1\}$. Step 0 then implies that the r_0r_1-tricomponent T containing Q equals $G' - \{vw_0, vw_1\}$. But this is impossible since the edge sets T and C are disjoint (Step 5.2 defines $T \subseteq E(X) \cup E[C, X]$). □

Claim 1 implies that if Step 0 ever returns because $\rho \geq a^{p+1}$, the cycle returned by the root of $\mathcal{R}$ has more than the desired length. Now assume $\mathcal{R}$ has height $< a^{p+1}$. We can also assume that no invocation of Step 2 or 4 returns a cycle of length $\geq g$. To check this we need only show $g \geq \alpha_{p+1}(a/k)^{p+1}$, equivalently $\left(\frac{p}{p+1}\right)^{p+1} \geq \alpha_{p+1}$. The latter follows from $p \geq 1$, since the left-hand side equals $(1 - 1/(p+1))^{p+1} \geq (1 - 1/2)^2 = 1/4 > \alpha_2 \geq \alpha_{p+1}$.

Define the values

$$\delta = a^{2(p+1)}, \qquad d = \log \delta = 2(p+1)\log a, \qquad \epsilon = 4/a^{p+1}. \tag{6}$$

The core of the argument is the following.

ASSERTION. *Consider a node τ of $\mathcal{R}$ at depth $\geq j$, where j is an integer in $0 \leq j < a^{p+1}$. Let C^* be a longest v-cycle for τ. Assume for some integer i, $1 \leq i \leq \frac{a}{(p+1)d}$,*

$$|C^*| \geq 2^{\widetilde{a} + id - j\epsilon}.$$

Then $|C_\tau| \geq iI$.

Before proving the assertion let us show that it implies the lemma. The root γ of $\mathcal{R}$ has depth $j = 0$. Set $i_0 = \lfloor \frac{a}{(p+1)d} \rfloor$. We will show that the assertion for $i = i_0$ implies the lemma's conclusion.

We begin by showing that i_0 satisfies the assertion's lower bound on $|C^*|$. Recall that the lemma assumes $|C^*| \geq 2^a$. Since $a = \widetilde{a} + a/(p+1) \geq \widetilde{a} + i_0 d$, this implies the desired lower bound.

Next we show the two inequalities

$$i_0 \geq 1 \quad \text{and} \quad i_0 \geq \frac{a}{4(p+1)^2 k}.$$

The first inequality completes the proof that i_0 satisfies all hypotheses of the assertion. The second inequality is used below.

To prove the inequalities, recall $\log a \leq \log a^* = k$. Thus the definition (6) of d shows that i_0 is the floor of the quantity

$$\frac{a}{(p+1)d} = \frac{a}{2(p+1)^2 \log a} \geq \frac{a}{2(p+1)^2 k}.$$

Thus it suffices to show that the rightmost quantity is at least one (since $x \geq 1$ implies $\lfloor x \rfloor \geq x/2$). Inequality (4) applied to algorithm $\mathcal{A}_{p+1}$ shows $a \geq 24k(p+1)^2$. Hence the rightmost quantity is $\geq 24k(p+1)^2/2(p+1)^2 k = 12 > 1$, as desired.

Since $(1+1/p)^p \leq e < 3$,

$$I = \frac{\alpha_p}{2} \frac{1}{(1+1/p)^p} \left(\frac{a}{k}\right)^p \geq \frac{\alpha_p}{6} \left(\frac{a}{k}\right)^p.$$

Using the assertion along with the second displayed lower bound for i_0 and the lower bound for I gives

$$|C_\gamma| \geq i_0 I \geq \frac{\alpha_p}{24(p+1)^2} \left(\frac{a}{k}\right)^{p+1} = \alpha_{p+1} \left(\frac{a}{k}\right)^{p+1},$$

thus establishing the lemma.

We turn to proving the assertion. We use an inner induction, this time inducting on the quantity $i - j$. (This induction is well founded since $i - j > 1 - a^{p+1}$.) For future reference observe the inequality

$$d \geq j\epsilon.$$

It follows since (6) implies $d \geq 2(p+1) \geq 4 = a^{p+1}\epsilon$. The next claim provides the base case of the inner induction.

CLAIM 2. *For any integers $1 \leq i \leq 2$ and $0 \leq j < a^{p+1}$, $|C_\tau| \geq 2I \geq iI$.*

Proof. Since $id - j\epsilon \geq d - j\epsilon \geq 0$, the assertion's lower bound on $|C^*|$ implies $|C^*| \geq 2^{\widetilde{a}}$. So when Step 1 is executed for the tricomponent containing C^*, $\mathcal{A}_p$ returns a cycle of length $\geq \alpha_p(\widetilde{a}/k)^p = 2I$. Here we have used the lemma for $\mathcal{A}_p$ (true by the inductive assumption on p) and the definition (2) of $\widetilde{a}$ and I. □

To do the inductive step (of the inner induction) assume $i \geq 3$. It suffices to show that some execution of Step 3.1, Step 6, or Step 7 constructs a cycle of length $\geq iI$. Fix C as its value after Step 2 has completed. Call a segment of C^* *large* if it has length $\geq 2|C^*|/\delta$. We establish the inductive step by focusing on the large segments. (Claim 6 below shows that they exist. Also note that $2|C^*|/\delta$ is a relatively big quantity, $|C^*|/\delta \geq 2^{3d-j\epsilon}/2^d = 2^{2d-j\epsilon} \geq 2^d = \delta$, and δ is much bigger than the cycle to be returned by $\mathcal{A}_{p+1}$.) Claims 3, 5, and 6 below exhaust all possibilities for the large segments and show that the assertion holds in each case. This establishes the inductive step.

CLAIM 3. *$|C_\tau| \geq iI$ if some large segment traverses a component X that has a neighbor $c \in C$ with $d_C(c,v) \geq I+1$.*

Proof. Lemma 2.1(ii) shows that in Step 3.1 graph H has a v'-cycle of length $\geq (2|C^*|/2\delta+1)+2 > |C^*|/\delta \geq 2^{\widetilde{a+(i-1)d-j\epsilon}}$. The recursive call to $\mathcal{A}_{p+1}$ has depth $\geq j+1$. Hence the inductive assertion implies that the recursive call finds a cycle of length $\geq (i-1)I$. Lemma 2.1(i) shows that the v-cycle constructed has length $\geq ((i-1)I-2)+d_C(c,v)+1 \geq iI$. □

Now assume that the hypothesis of Claim 3 never holds. Our next goal is to derive the bound (8) below for estimating the performance of Step 5. For the rest of the lemma's proof, the notation $C^*[x,y]$ abbreviates $C^*_{\overline{v}}[x,y]$, i.e., all subpaths of C^* that we will refer to avoid v.

CLAIM 4. *Consider any component X that contains a large segment, say the segment $a_0, C^*[x_0,x_1], a_1$. Define sets F_0 and F_1 as in Step 5.1 and value b as in (3). Let path P be $C^*[x_0,x_1]$. Then all hypotheses of Lemma 2.6 are satisfied.*

Proof. We start with a useful inequality:

$$2 + 2d_X(x_0,x_1)b \leq |C^*|/\delta a^{p+1}. \tag{7}$$

To prove this recall from Step 4 that we have assumed $d_X(x_0,x_1) < g$. Furthermore, $g \leq (a/k)^{p+1} \leq (a/8)^{p+1}$. Hence

$$2 + 2d_X(x_0,x_1)b \leq 4d_X(x_0,x_1)b \leq 8\left(\frac{a}{8}\right)^{p+1} 2^{\widetilde{a}} \leq a^{p+1}2^{\widetilde{a}} = \left(\frac{\delta}{a^{p+1}}\right)2^{\widetilde{a}} = \frac{2^{\widetilde{a+d}}}{a^{p+1}}.$$

Using $i \geq 3$ and $d \geq j\epsilon$ gives $d \leq (i-1)d - j\epsilon$. Hence

$$2^{\widetilde{a+d}} \leq 2^{\widetilde{a+(i-1)d-j\epsilon}} \leq \frac{|C^*|}{\delta}.$$

Combining the last two displayed inequalities gives (7).

We proceed to verify the hypotheses of Lemma 2.6:

(i) $Y(E[X,Y]) \neq Y$. This holds since $v \notin Y(E[X,Y])$.

(ii) Each set F_i is b-close. From Claim 3 we have assumed that any edge $cx \in E[C,X]$ has $d_C(c,v) < I+1$. So the definition of F_i (Step 5.1) shows any two edges $cx, c'x'$ in F_i have $|C_{\overline{v}}[c,c']| < I$. Recalling that Step 2 cannot enlarge C any further, this inequality implies that F_i is b-close.

(iii) $|P| \geq 2d_X(x_0,x_1)b+2$. Since $P = C^*[x_0,x_1]$ comes from a large segment, (7) gives

$$\frac{2|C^*|}{\delta} - 2 \geq \frac{|C^*|}{\delta} \geq \frac{|C^*|}{\delta a^{p+1}} \geq 2 + 2d_X(x_0,x_1)b.$$

(iv) $x_i \in X(F_i)$ for $i = 0,1$. We can assume $x_0 \in X(F_0)$, by possibly renaming vertices w_0 and w_1. So we must show $x_1 \in X(F_1)$. The inequality of (iii) gives $D_X(x_0,x_1) \geq d_X(x_0,x_1)b$, so $D_X(x_0,x_1)/d_X(x_0,x_1) \geq b$. Now Lemma 2.3(ii) shows that x_0 and x_1 are separated by a b-round bicomponent in X. Since F_0 is b-close, the independent edges a_0x_0 and a_1x_1 show $x_1 \notin X(F_0)$, i.e., $x_1 \in X(F_1)$. □

Claim 4 shows that any large segment $a_0, C^*[x_0,x_1], a_1$ has a corresponding triplet (r_0,r_1,T) in the tricomponent cover $\mathcal{T}$ constructed in Step 5.2. Here r_0 and r_1 are the near separators of Lemma 2.6, and T is the r_0r_1-tricomponent containing $C^*[r_0,r_1]$. Thus Step 5.3 adds to $\mathcal{Q}$ a path Q corresponding to the triplet (r_0,r_1,T).

Furthermore, the large segment satisfies

$$|C^*[r_0, r_1]| > |C^*[a_0, a_1]| - \frac{|C^*|}{\delta a^{p+1}}. \tag{8}$$

To prove this first observe that if F_0 is not a star, then Lemma 2.5 shows $|C^*[x_0, r_0]| < d_X(x_0, x_1)b$. A similar statement holds for F_1, so if neither set is a star, then

$$|C^*[r_0, r_1]| > |C^*[x_0, x_1]| - 2d_X(x_0, x_1)b.$$

It is easy to check that the above inequality continues to hold when either set F_i is a star, since in that case r_i is the star's center x_i or a_i. The right-hand side is equal to $|C^*[a_0, a_1]| - 2 - 2d_X(x_0, x_1)b$. Applying (7) to this quantity gives (8).

To help bound the right-hand side of (8) we use the following estimate:

$$|C^*| \left(1 - \frac{1}{a^{p+1}}\right) \geq 2^{\widetilde{a}+id-(j+1)\epsilon}. \tag{9}$$

To prove this recall that $\ln(1-x) \geq -2x$ for $0 \leq x \leq 1/2$. Hence $\log(1 - 1/a^{p+1}) = \log e \ \ln(1 - 1/a^{p+1}) \geq 2(-2/a^{p+1}) = -4/a^{p+1} = -\epsilon$. Combining this with the assertion's assumption $|C^*| \geq 2^{\widetilde{a}+id-j\epsilon}$ gives (9).

We now analyze the two remaining possibilities for the inner inductive step. Recall that the inductive quantity is $i - j$.

Claim 5. *$|C_\tau| \geq iI$ if two large segments exist.*

Proof. (8) and (9) show that each large segment satisfies

$$|C^*[r_0, r_1]| > \left(\frac{2|C^*|}{\delta}\right)\left(1 - \frac{1}{a^{p+1}}\right) \geq 2^{\widetilde{a}+(i-1)d-(j+1)\epsilon+1}.$$

Step 5.3 makes a recursive call for the triplet of $\mathcal{T}$ corresponding to each large segment. This call gives a node of depth $\geq j+1$ in the recursion tree $\mathcal{R}$. Hence by the inner inductive assertion the recursive call finds an r_0r_1-path Q of length $\geq (i-1)I - 2$.

Step 6 eventually considers the pair consisting of these two paths Q. The gluing principle ensures that Step 6 finds a v-cycle containing the two paths. The cycle has length $\geq 2((i-1)I - 2) + 2 = iI + (i-2)I - 2 \geq iI$, where we have used $i \geq 3$ and $I \geq 2$ (from (5)). □

Claim 6. *$|C_\tau| \geq iI$ if at most one large segment exists.*

Proof. Let us first show that a large segment must exist. C^* has $\leq |C|$ segments, and we have assumed (from Step 2) that $|C| \leq g$. So the segments that are not large have total length $\leq (2|C^*|/\delta)(a/k)^{p+1} \leq 2|C^*|/(ak)^{p+1}$. The last quantity is $< |C^*|$. So the longest segment must be large.

Denote the longest segment as $a_0, C^*[x_0, x_1], a_1$. The previous paragraph and the hypothesis of Claim 5 shows that its length is $\geq |C^*| - 2|C^*|/(ak)^{p+1}$. Applying (8) and (9) to this segment, and using the inequalities $\delta, k \geq 2$, and $p \geq 1$, gives

$$|C^*[r_0, r_1]| > \left(|C^*| - \frac{2|C^*|}{(ak)^{p+1}}\right) - \frac{|C^*|}{\delta a^{p+1}} \geq |C^*| \left(1 - \frac{1}{a^{p+1}}\right) \geq 2^{\widetilde{a}+id-(j+1)\epsilon}.$$

Step 5.3 makes a recursive call for the triplet of $\mathcal{T}$ for this segment. The call gives a node of depth $\geq j+1$ in $\mathcal{R}$. Hence by the inner inductive assertion the recursive call finds an r_0r_1-path of length $\geq iI - 2$. Step 7 enlarges this path to a v-cycle of length $\geq (iI - 2) + 2 = iI$, as desired. □

Suppose an iteration of Step 2 of the Main Routine has $|C^*| \geq 2^{a^*}$. We show that $\mathcal{A}_{p^*}$ returns a cycle of length $\geq e^{p^*}$. For notational simplicity we drop the asterisks and write a, p, k for a^*, p^*, k^*.

Lemma 4.1 shows that $\mathcal{A}_p$ returns a cycle of length $\geq \alpha_p(a/k)^p$. The definition of α_p and (4) give

$$\alpha_p \left(\frac{a}{k}\right)^p \geq \frac{1}{24^{p-1}(p!)^2}\,(24p^2)^p \geq 24\frac{p^{2p}}{(p!)^2}.$$

Any $p \geq 1$ satisfies $p! \leq 3\sqrt{p}(p/e)^p$ [5, p. 55]. Hence the rightmost quantity is at least

$$24\frac{p^{2p}}{9p(p/e)^{2p}} \geq \frac{e^{2p}}{p} \geq e^p.$$

This gives the desired inequality.

LEMMA 4.2. *The Main Routine returns a v-cycle having length that is at least* $\exp(c\sqrt{\log \ell / \log\log \ell})$, *where ℓ is the length of a longest v-cycle and c is some positive constant.*

Proof. Assume that Step 1 of the Main Routine does not find a longest cycle, so $\log \ell > 2^8$. Step 2 of the Main Routine eventually chooses k^* so that

$$a^* \leq \log \ell \leq 2a^*.$$

(This is true even if $\ell = n$, since the last value for a^* in Step 2 is $\geq 2^{\log\log n - 1} = (\log n)/2$.) The first inequality gives $|C^*| \geq 2^{a^*}$, so, as shown above, $\mathcal{A}_{p^*}$ returns a cycle of length $\geq e^{p^*}$.

Step 2 shows $p^* \geq \sqrt{a^*/24k^*}/2$ (since $x \geq 1$ implies $\lfloor x \rfloor \geq x/2$). The displayed inequalities imply $a^* \geq (\log \ell)/2$ and $k^* \leq \log\log \ell$. Thus $a^*/k^* \geq (\log \ell)/2\log\log \ell$. Hence $p^* \geq c\sqrt{\log \ell / \log\log \ell}$ for some constant $c > 0$. The lemma follows. □

5. Implementation and timing analysis. This section starts by presenting the remaining implementation details of the algorithm, thus allowing a complete timing analysis. The section ends by showing that the algorithm runs in polynomial time.

Implementation of Step 2.1. We break this step into a number of "passes." Each pass but the last one ends by enlarging C. Each pass examines each bicomponent B of each connected component X, as follows.

Consider a bicomponent B that separates two independent edges $cx, c'x'$ selected as in Step 2.1 (Figure 5(a)), i.e., $|C_{\overline{v}}[c, c']| < I$. (If there is more than one such pair of edges for B, choose arbitrarily. If there is no such pair for B, proceed to the next bicomponent.) Let y be the projection $\pi_B(x)$, i.e., the vertex of $V(B)$ on every minimal path from x to B. Similarly let y' be $\pi_B(x')$. So $y \neq y'$. Call $\mathcal{A}_p$ to find a yy'-path Y in B (creating an artificial vertex v' as in Step 2.1). If $|Y| \geq I$, enlarge Y to a path from cx to $c'x'$, and use that cc'-path to enlarge C. Then begin the next pass. (If $|Y| < I$, just continue to the next bicomponent B.)

Step 2 ends when $|C| \geq g$ or a pass examines every bicomponent B without enlarging C.

The argument for Claim 0 in Lemma 4.1 (applied to y, y') shows that if B is b-round, $|Y| \geq 2I - 2 \geq I$. Hence we enlarge C in this case. It is possible that $|Y| \geq I$ even if B is not b-round. It causes no harm that we enlarge C in this case too. Thus the implementation of Step 2.1 is correct.

For the efficiency analysis, observe that in each pass, an edge of G belongs to the graph of at most one call to $\mathcal{A}_p$.

Implementation of Step 6. Consider the search for a v-cycle containing two paths of $\mathcal{Q}$. Let the paths be Q, corresponding to triplet (r_0, r_1, T), and Q', corresponding to triplet (r_0', r_1', T'). Choose an internal vertex q (resp., q') of Q (resp., Q'). Find a cycle A through v, q, and q', if one exists. The pair r_0, r_1 separates q and v, by construction. Hence A traverses T, the r_0r_1-tricomponent containing q. Replace the subpath $A_q[r_0, r_1]$ by Q. Do the same for Q'. This gives the desired cycle. (In particular, the new cycle is simple.)

A can be found by the algorithm of LaPaugh and Rivest [14]. This algorithm finds a cycle through three given vertices. It uses linear time (with no large hidden constants). For the efficiency analysis observe that $\mathcal{Q}$ contains $\leq n$ paths (recall property (ii) of the definition of tricomponent cover). Hence Step 6 processes $O(n^2)$ pairs of paths.

Implementation of arithmetic. We show how to implement all the tests involving numeric quantities in polynomial time. This will be straightforward since the algorithm has been written to avoid computing $\log a$ (using instead the unchanging value k^*). We will use these simple inequalities:

$$p, p^*, k \leq a^* \leq \log n.$$

Note that p, p^*, k, and a^* are integers. Also in this subsection and the next the largest number we need to estimate is $(a^*)^{2(p^*)^2}$. This number is sublinear (in n):

$$\log (a^*)^{2(p^*)^2} = 2(p^*)^2 \log a^* \leq 2\frac{a^*}{24k^*}k^* = \frac{a^*}{12} \leq \frac{\log n}{12}. \tag{10}$$

In the Main Routine, Step 2 computes $p^* = \lfloor \sqrt{a^*/24k^*} \rfloor$. Equivalently, p^* is the unique integer satisfying $24k^*(p^*)^2 \leq a^* < 24k^*(p^*+1)^2$. Since all variables here are integers $\leq \log n$ we can find p^* in polynomial time by enumeration.

We turn to Algorithm $\mathcal{A}_{p+1}$. We can assume the algorithm is given integers $a^*, p^*, k = k^*$ and p (as well as ρ). We list below all numeric quantities q computed by $\mathcal{A}_{p+1}$. We further observe that the only use of each of these quantities q is in a comparison with a known integral quantity i; i.e., the algorithm needs to determine only which relation $i \prec q$ holds for $\prec \in \{<, =, >\}$:

$$a^{p+1} = \left(\frac{(p+1)a^*}{p^*}\right)^{p+1} \quad \text{(Step 0)}, \qquad g = \left(\frac{pa^*}{kp^*}\right)^{p+1} \quad \text{(Steps 2 and 4)},$$
$$I = \frac{\alpha_p}{2}\left(\frac{pa^*}{kp^*}\right)^{p} \quad \text{(Steps 2.1 and 3)}.$$

Recall also the quantity $\alpha_p = 1/24^{p-1}(p!)^2$.

Recall that $p + 1 \leq p^* \leq \log n$. Treat each of the quantities q as a fraction, computing its numerator and denominator in $O(p^*)$ integer multiplications. We will show that each numerator and denominator is an integer $\leq n$. So, performing the comparison with i presents no problem.

The numerator and denominator of the first two quantities a^{p+1} and g are $\leq (a^*)^{2p^*}$. So (10) gives the desired conclusion. For the third quantity I the numerator is again $\leq (a^*)^{2p^*}$. The denominator is the product of three integers each $\leq (a^*)^{2p^*}$, since $24 < 2^8 \leq a^*$ and $(p!)^2 \leq p^{2p} \leq (a^*)^{2p^*}$. So again (10) gives the desired conclusion.

Wrapup.

THEOREM 5.1. *The Main Routine returns a v-cycle having length that is $\exp(\Omega(\sqrt{\log \ell / \log\log \ell}))$ in polynomial time, where ℓ is the length of a longest v-cycle.*

Proof. Lemma 4.2 proves the length guarantee, so we need only establish the polynomial time bound. For this we need only show that each execution of $\mathcal{A}_{p^*}$ by Step 2 of the Main Routine uses polynomial time. Throughout this argument, G^* denotes the given graph.

First observe (Figure 5) that every invocation of a routine $\mathcal{A}_p$, $p \le p^*$, is given a graph G that consists of two edges incident to v plus edges that are either in G^* or images of such edges. (An image of an edge in G^* is created in Step 3.1: an edge $c'x$ comes from an edge from $C - c$ to x.) This follows by a simple induction.

The pseudocode in section 3 plus the implementation details of this section show that if we ignore the recursive calls that $\mathcal{A}_{p+1}$ makes to itself or $\mathcal{A}_p$, Algorithm $\mathcal{A}_{p+1}$ takes polynomial time. Specifically note that Step 2.1 repeats at most g times, and g is sublinear (by (10)). Step 4 uses breadth-first search, and Step 5.2 finds a tricomponent cover using a routine for biconnected components [5, 17]. Step 6 uses polynomial time, as we noted in its implementation.

Hence it suffices to show there are a polynomial number of invocations of all routines $\mathcal{A}_p$, $p \le p^*$. We have shown that each invocation of a routine $\mathcal{A}_p$ uses a graph that contains an edge of G^* (or its image). Hence it suffices to show that each edge e of G^* is in the graph of a polynomial number of invocations $\mathcal{A}_p$, $p \le p^*$.

Take any $p \le p^*$. Let $\mathcal{R}$ be the recursion tree of $\mathcal{A}_p$. As in the previous section, $\mathcal{R}$ contains nodes only for calls to $\mathcal{A}_p$. Suppose that edge e of G^* (or its image) belongs to the graph of a node τ of $\mathcal{R}$. Then e belongs to the graph of at most one child of τ. In proof, the recursive calls to $\mathcal{A}_p$ occur in Steps 3.1 and 5.3. A component X is processed in only one of the Steps 3 or 5. Step 5.2 ensures that all tricomponents T are pairwise disjoint (property (ii) of the definition of tricomponent cover). So a given edge is involved in only one recursive call in Step 5.3.

We conclude that e occurs in $\le a^p$ nodes τ of $\mathcal{R}$. (Since we're discussing $\mathcal{A}_p$ rather than $\mathcal{A}_{p+1}$, we reduce p in the pseudocode of section 3 by 1; e.g., Step 0 guarantees that $\rho \le a^p$.)

In each such node τ, e occurs in $\le g \le a^p$ graphs for calls to $\mathcal{A}_{p-1}$ in Steps 1 and 2.1. This follows since each pass of Step 2.1 except the last enlarges C, and e belongs to ≤ 1 graph per pass. We conclude that e occurs in at most $a^p \times a^p = a^{2p}$ calls from this invocation of $\mathcal{A}_p$ to $\mathcal{A}_{p-1}$.

Any routine $\mathcal{A}_p$ has $a \le a^*$, $p \le p^*$. So overestimating, edge e is in the initial graph of $\le ((a^*)^{2(p^*)})^i$ recursion trees for routine $\mathcal{A}_{p^*-i}$, for any $0 \le i < p^*$. In each of these recursion trees, we have already noted that e occurs in $\le (a^*)^{p^*}$ nodes. This gives a total of $\le (a^*)^{2p^*i} \times (a^*)^{p^*} \le (a^*)^{2p^*(i+1)}$ nodes in recursion trees for calls to $\mathcal{A}_{p^*-i}$.

Summing for i going from 0 to $p^* - 1$, we have shown that e occurs in a total of $\le p^*(a^*)^{2(p^*)^2}$ nodes (i.e., recursive invocations). Using $p^* \le \log n$ and (10), this bound is polynomial (in fact sublinear), as desired. ∎

6. Conclusions. Despite many levels of recursion, we leave the long cycle problem with a large gap between our guaranteed length bound (which is a cycle of length $\exp(\Omega(\sqrt{\log \ell / \log\log \ell}))$) and the best-known hardness result (the quasi-NP-hardness of a factor $2^{O(\log^{1-\epsilon} n)}$ approximation to the longest path).

Since the initial appearance of this paper [10] some surprising progress has been made. Feder and Motwani [7] have shown how to find a cycle whose length is

$\exp(\Omega(\log n/\log\log n))$ in Hamiltonian graphs, in time $O(n^3)$. Their results also improve our length guarantee in graphs that are very close to Hamiltonian ($\ell \geq n/\exp(o(\sqrt{\log n \log\log n}))$) as well as in some other cases. Their algorithm, like ours, combines two recursively found paths through tricomponents, using the algorithm of [14] for a cycle through three given vertices. In addition, they use a variety of other new ideas, starting with a result of Chen, Xu, and Yu [6]: in a 3-connected graph of maximum degree d, a cycle of length $\geq n^{\log_b 2}$ exists and can be found efficiently, for $b = 2(d-1)^2 + 1$.

REFERENCES

[1] N. Alon, R. Yuster, and U. Zwick, *Color-coding*, J. ACM, 42 (1995), pp. 844–856.

[2] H. L. Bodlaender, *Minor tests with depth-first search*, J. Algorithms, 14 (1993), pp. 1–23.

[3] A. Björklund and T. Husfeldt, *Finding a path of superlogarithmic length*, SIAM J. Comput., 32 (2003), pp. 1395–1402.

[4] A. Björklund, T. Husfeldt, and S. Khanna, *Approximating longest directed paths and cycles*, in Proceedings of the 31st International Conference on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 3142, Springer, New York, 2004, pp. 222–233.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed., McGraw–Hill, New York, 2001.

[6] G. Chen, J. Xu, and X. Yu, *Circumference of graphs with bounded degree*, SIAM J. Comput., 33 (2004), pp. 1136–1170.

[7] T. Feder and R. Motwani, *Finding large cycles in Hamiltonian graphs*, in Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, Vancouver, BC, 2005, SIAM, Phiadelphia, pp. 166–175.

[8] T. Feder, R. Motwani, and C. Subi, *Approximating the longest cycle problem in sparse graphs*, SIAM J. Comput., 31 (2002), pp. 1596–1607.

[9] M. R. Fellows and M. A. Langston, *Nonconstructive tools for proving polynomial-time decidability*, J. ACM, 35 (1988), pp. 727–739.

[10] H. N. Gabow, *Finding paths and cycles of superpolylogarithmic length*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, 2004, ACM, New York, pp. 407–416.

[11] H. N. Gabow and S. Nie, *Finding a long directed cycle*, in Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA, 2004, SIAM, Philadelphia, pp. 49–58.

[12] J. E. Hopcroft and R. E. Tarjan, *Dividing a graph into triconnected components*, SIAM J. Comput., 2 (1973), pp. 135–158.

[13] D. Karger, R. Motwani, and G. D. S. Ramkumar, *On approximating the longest path in a graph*, Algorithmica, 18 (1997), pp. 82–98.

[14] A. S. LaPaugh and R. L. Rivest, *The subgraph homeomorphism problem*, J. Comput. System Sci., 20 (1980), pp. 133–149.

[15] B. Monien, *How to find long paths efficiently*, Ann. Discrete Math., 25 (1985), pp. 239–254.

[16] N. Robertson and P. D. Seymour, *Graph minors.* XIII: *The disjoint paths problem*, J. Combin. Theory B, 63 (1995), pp. 65–110.

[17] R. Tarjan, *Depth-first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.

[18] M. E. Watkins and D. M. Mesner, *Cycles and connectivity in graphs*, Canadian J. Math., 19 (1967), pp. 1319–1328.

[19] D. B. West, *Introduction to Graph Theory*, 2nd ed., Prentice–Hall, Upper Saddle River, NJ, 2001.

SIAM J. COMPUT.
Vol. 36, No. 6, pp. 1672–1695

AN OPTIMAL CACHE-OBLIVIOUS PRIORITY QUEUE AND ITS APPLICATION TO GRAPH ALGORITHMS*

LARS ARGE†, MICHAEL A. BENDER‡, ERIK D. DEMAINE§,
BRYAN HOLLAND-MINKLEY¶, AND J. IAN MUNRO‖

Abstract. We develop an optimal cache-oblivious priority queue data structure, supporting insertion, deletion, and delete-min operations in $O(\frac{1}{B}\log_{M/B}\frac{N}{B})$ amortized memory transfers, where M and B are the memory and block transfer sizes of any two consecutive levels of a multilevel memory hierarchy. In a cache-oblivious data structure, M and B are not used in the description of the structure. Our structure is as efficient as several previously developed external memory (cache-aware) priority queue data structures, which all rely crucially on knowledge about M and B. Priority queues are a critical component in many of the best known external memory graph algorithms, and using our cache-oblivious priority queue we develop several cache-oblivious graph algorithms.

Key words. cache-oblivious algorithms, priority queue

AMS subject classification. 68P05

DOI. 10.1137/S0097539703428324

1. Introduction.

As the memory systems of modern computers become more complex, it is increasingly important to design algorithms that are sensitive to the structure of memory. One of the characteristic features of modern memory systems is that they are made up of a hierarchy of several levels of cache, main memory, and disk. While traditional theoretical computational models have assumed a "flat" memory with uniform access time, the access times of different levels of memory can vary by several orders of magnitude in current machines. Thus algorithms for hierarchical memory have received considerable attention in recent years. Very recently, the *cache-oblivious* model was introduced as a way of achieving algorithms that are efficient in arbitrary memory hierarchies without the use of complicated multilevel memory models. In this paper we develop an optimal cache-oblivious priority queue and use it to develop several cache-oblivious graph algorithms.

*Received by the editors May 23, 2003; accepted for publication (in revised form) August 4, 2006; published electronically March 19, 2007. An extended abstract version of this paper was presented at the 2002 ACM Symposium on Theory of Computation (STOC'02), AMC Press, New York, 2002, pp. 268–276.

http://www.siam.org/journals/sicomp/36-6/42832.html

†Department of Computer Science, University of Aarhus, DK-8200 Aarhus N, Denmark (large@daimi.au.dk). This work was done while this author was at Duke University. This author was supported in part by the National Science Foundation through ESS grant EIA–9870734, RI grant EIA–9972879, CAREER grant CCR–9984099, and ITR grant EIA–0112849.

‡Department of Computer Science, SUNY Stony Brook, Stony Brook, NY 11794 (bender@cs.sunysb.edu). This author was supported in part by the National Science Foundation through ITR grant EIA–0112849 and by HRL Laboratories and Sandia National Laboratories.

§Laboratory for Computer Science, MIT, Cambridge, MA 02139 (edemaine@mit.edu). This author was supported in part by the National Science Foundation through ITR grant EIA–0112849.

¶Department of Computer Science, Duke University, Durham, NC 27708 (bhm@cs.duke.edu). This author was supported in part by the National Science Foundation through ITR grant EIA–0112849.

‖School of Computer Science, University of Waterloo, Waterloo, ON, N2L 3G1, Canada (imunro@uwaterloo.ca). This author was supported in part by the Natural Science and Engineering Council through grant RGPIN 8237-97 and the Canada Research Chair in Algorithm Design.

1.1. Background and previous results. Traditionally, most algorithmic work has been done in the *random access machine* (RAM) model of computation, which models a "flat" memory with uniform access time. Recently, some attention has turned to the development of theoretical models and algorithms for modern complicated hierarchical memory systems; refer, e.g., to [3, 4, 5, 7, 53, 59]. Developing models that are both simple and realistic is a challenging task since a memory hierarchy is described by many parameters. A typical hierarchy consists of a number of memory levels, with memory level ℓ having size M_ℓ and being composed of M_ℓ/B_ℓ blocks of size B_ℓ. In any memory transfer from level ℓ to $\ell-1$, an entire block is moved atomically. Each memory level also has an associated *replacement strategy*, which is used to decide what block to remove in order to make room for a new block being brought into that level of the hierarchy. Further complications are the limited *associativity* of some levels of the hierarchy, meaning that a given block can only be loaded into a limited number of memory positions, as well as the complicated *prefetching strategies* employed by many memory systems. In order to avoid the complications of multilevel memory models, a body of work has focused on two-level memory hierarchies. Most of this work has been done in the context of problems involving massive datasets, because the extremely long access times of disks compared to the other levels of the hierarchy means that I/O between main memory and disk is often the bottleneck in such problems.

1.1.1. Two-level I/O model. In the two-level *I/O model* (or *external memory model*) introduced by Aggarwal and Vitter [6], the memory hierarchy consists of an internal memory of size M and an arbitrarily large external memory partitioned into blocks of size B. The efficiency of an algorithm in this model (a so-called *I/O* or *external memory* algorithm) is measured in terms of the number of block transfers it performs between these two levels (here called *memory transfers*). An algorithm has complete control over the placement of blocks in main memory and on disk. The simplicity of the I/O model has resulted in the development of a large number of external memory algorithms and techniques. See, e.g., [10, 59] for recent surveys.

The number of memory transfers needed to read N contiguous elements from disk is $scan(N) = \Theta(\frac{N}{B})$ (the *linear* or *scanning* bound). Aggarwal and Vitter [6] showed that $\Theta(\frac{N}{B}\log_{M/B}\frac{N}{B})$ memory transfers are necessary and sufficient to sort N elements. In this paper, we use $\mathrm{sort}(N)$ to denote $\frac{N}{B}\log_{M/B}\frac{N}{B}$ (the *sorting* bound). The number of memory transfers needed to search for an element among a set of N elements is $\Omega(\log_B N)$ (the *searching* bound), and this bound is matched by the B-tree, which also supports updates in $O(\log_B N)$ memory transfers [17, 36, 40, 39]. An important consequence of these bounds is that, unlike in the RAM model, one cannot sort optimally with a search tree—inserting N elements in a B-tree takes $O(N\log_B N)$ memory transfers, which is a factor of $(B\log_B N)/(\log_{M/B}\frac{N}{B})$ from optimal. Finally, permuting N elements according to a given permutation takes $\Theta(\min\{N, \mathrm{sort}(N)\})$ memory transfers, and for all practical values of N, M, and B this is $\Theta(\mathrm{sort}(N))$ [6]. This represents another fundamental difference between the RAM and I/O model, since N elements can be permuted in $O(N)$ time in the RAM model.

The I/O model can also be used to model the two-level hierarchy between cache and main memory; refer, e.g., to [54, 42, 44, 60, 50, 52]. Some of the main shortcomings of the I/O model on this level are the lack of explicit application control of placement of data in the cache and the low associativity of many caches. However, as demonstrated by Sen, Chatterjee, and Dumir [54], I/O model results can often be used to obtain results in more realistic two-level cache main memory models. Algorithms that are

efficient on the two-level cache main memory levels have also been considered by, e.g., LaMarca and Ladner [42, 43].

1.1.2. Cache-oblivious model. One of the main disadvantages of two-level memory models is that they force the algorithm designer to focus on a particular level of the hierarchy. Nevertheless, the I/O model has been widely used because it is convenient to consider only two levels of the hierarchy. Very recently, a new model that combines the simplicity of the I/O mode with the realism of more complicated hierarchical models was introduced by Frigo et al. [38]. The idea in the *cache-oblivious model* is to design and analyze algorithms in the I/O model but without using the parameters M and B in the algorithm description. It is assumed that when an algorithm accesses an element that is not stored in main memory, the relevant block is automatically fetched into memory with a *memory transfer*. If the main memory is full, the *ideal* block in main memory is elected for replacement based on the future characteristics of the algorithm; that is, an *optimal* offline paging strategy is assumed. While this model may seem unrealistic, Frigo et al. [38] showed that it can be simulated by essentially any memory system with only a small constant-factor overhead. For example, the least recently used (LRU) block-replacement strategy approximates the omniscient strategy within a constant factor, given a cache larger by a constant factor [38, 55]. The main advantage of the cache-oblivious model is that it allows us to reason about a simple two-level memory model, but prove results about an unknown, multilevel memory hierarchy; because an analysis of an algorithm in the two-level model holds for any block and main memory size, it holds for *any* level of the memory hierarchy. As a consequence, if the algorithm is optimal in the two-level model, it is optimal on *all* levels of a multilevel memory hierarchy.

Frigo et al. [38] developed optimal cache-oblivious algorithms for matrix multiplication, matrix transposition, fast Fourier transform, and sorting. Subsequently, several authors developed dynamic cache-oblivious B-trees with a search and update cost of $O(\log_B N)$ matching the standard (cache-aware) B-tree [21, 27, 22, 49, 20]. Recently, several further results have been obtained, e.g., [23, 57, 24, 25, 26, 18, 2, 14, 16, 19, 28, 33]. See also the survey in [13]. Some of these results assume that $M \geq B^2$ (the *tall-cache* assumption), and we will also make that assumption in this paper. Brodal and Fagerberg [26] showed that an assumption of this type (actually $M = \Omega(B^{1+\epsilon})$ for some $\epsilon > 0$) is necessary to obtain the I/O model sorting bound in the cache-oblivious model.

1.1.3. Priority queues. A priority queue maintains a set of elements each with a priority (or key) under *insert* and *delete-min* operations, where a delete-min operation finds and deletes the element with the minimum key in the queue. Sometimes *delete* of an arbitrary element is also supported, often, as in this paper, assuming that the key of the element to be deleted is known. The heap is a standard implementation of a priority queue, and a balanced search tree can, of course, also easily be used to implement a priority queue. In the I/O model, a priority queue based on a B-tree would support all operations in $O(\log_B N)$ memory transfers. The standard heap can also be easily modified (to have fanout B) so that all operations are supported in the same bound (see, e.g., [44]). The existence of a cache-oblivious B-tree immediately implies the existence of an $O(\log_B N)$ cache-oblivious priority queue.

As discussed in section 1.1.1, the use of an $O(\log_B N)$ search tree (or priority queue) to sort N elements results in an I/O model algorithm that is a factor of $(B \log_B N)/(\log_{M/B}(N/B))$ from optimal. To sort optimally we need a data structure

supporting the relevant operations in $O(\frac{1}{B}\log_{M/B}\frac{N}{B})$ memory transfers. Note that for reasonable values of N, M, and B, this bound is less than 1 and we can, therefore, only obtain it in an amortized sense. To obtain such a bound, Arge developed the *buffer tree technique* and showed how it can be used on a B-tree to obtain a priority queue supporting all operations in $O(\frac{1}{B}\log_{M/B}\frac{N}{B})$ amortized memory transfers [11]. This structure seems hard to make cache-oblivious since it involves periodically finding the $\Theta(M)$ smallest key elements in the structure and storing them in internal memory. Efficient I/O model priority queues have also been obtained by using the buffer tree technique on heap structures [37, 41]. The heap structure by Fadel et al. [37] seems hard to make cache-oblivious because it requires collecting $\Theta(M)$ insertions and performing them all on the structure at the same time. The structure by Kumar and Schwabe [41] avoids this by using the buffer technique on a tournament tree data structure. However, they only obtained $O(\frac{1}{B}\log_2 N)$ bounds, mainly because their structure was designed to also support an *update* operation. Finally, Brodal and Katajainen [29] developed a priority queue structure based on an M/B-way merging scheme. Very recently, and after the appearance of the conference version of this paper, Brodal and Fagerberg [25] managed to develop a similar merging based cache-oblivious priority queue based on ideas they developed in [24]. Brodal et al. [28], as well as Chowdhury and Ramachandran [33], have also developed cache-oblivious priority queues that support updates in the same bound as the I/O-efficient structure of Kumar and Schwabe [41].

1.1.4. I/O model graph algorithms. The superlinear lower bound on permutation in the I/O model has important consequences for the I/O complexity of graph algorithms, because the solution of almost any graph problem involves somehow permuting the V vertices or E edges of the graph. Thus $\Omega(\min\{V, \text{sort}(V)\})$ is in general a lower bound on the number of memory transfers needed to solve most graph problems. Refer to [9, 31, 46]. As mentioned in section 1.1.1, this bound is $\Omega(\text{sort}(V))$ in all practical cases. Still, even though a large number of I/O model graph algorithms have been developed (see [59, 61] and references therein), not many algorithms match this bound. Below we review the results most relevant to our work.

As with PRAM graph algorithms [51], list ranking—the problem of ranking the elements in a linked list stored as an unordered sequence in memory—is the most fundamental I/O model graph problem. Using PRAM techniques, Chiang et al. [31] developed the first efficient I/O model list ranking algorithm. Using an I/O-efficient priority queue, Arge [11] showed how to solve the problem in $O(\text{sort}(V))$ memory transfers. The list ranking algorithm and PRAM techniques can be used in the development of $O(\text{sort}(V))$ algorithms for many problems on trees, such as computing an Euler tour, breadth-first search (BFS), depth-first search (DFS), and centroid decomposition [31]. The best known DFS and BFS algorithms for general directed graphs use $O(V + \frac{EV}{BM})$ [31] or $O((V + E/B)\log_2 V + \text{sort}(E))$ [30] memory transfers. For undirected graphs, improved $O(V + \text{sort}(E))$ and $O(\sqrt{\frac{V \cdot E}{B}} + \text{sort}(E))$ BFS algorithms have been developed [46, 45]. The best known algorithms for computing the connected components and the minimum spanning forest of a general undirected graph both use $O(\text{sort}(E) \cdot \log_2 \log_2(\frac{VB}{E}))$ or $O(V + \text{sort}(E))$ memory transfers [46, 15].

1.2. Our results. The main result of this paper is an optimal cache-oblivious priority queue. Our structure supports insert, delete, and delete-min operations in $O(\frac{1}{B}\log_{M/B}\frac{N}{B})$ amortized memory transfers and $O(\log N)$ amortized computation time; it is described in section 2. The structure is based on a combination of several

TABLE 1.1
Summary of our results (priority queue bounds are amortized).

Problem	Our cache-oblivious result	Best cache-aware result	
Priority queue	$O(\frac{1}{B}\log_{M/B}\frac{N}{B})$	$O(\frac{1}{B}\log_{M/B}\frac{N}{B})$	[11]
List ranking	$O(\text{sort}(V))$	$O(\text{sort}(V))$	[31, 11]
Tree algorithms	$O(\text{sort}(V))$	$O(\text{sort}(V))$	[31]
Directed BFS and DFS	$O((V+E/B)\log_2 V + \text{sort}(E))$	$O((V+E/B)\log_2 V + \text{sort}(E))$	[30]
		$O(V + \frac{EV}{BM})$	[31]
Undirected BFS	$O(V + \text{sort}(E))$	$O(V + \text{sort}(E))$	[46]
		$O(\sqrt{\frac{V\cdot E}{B}} + \text{sort}(E))$	[45]
MSF	$O(\text{sort}(E)\cdot\log_2\log_2 V)$	$O(\text{sort}(E)\cdot\log_2\log_2\frac{VB}{E})$	[15]
	$O(V + \text{sort}(E))$	$O(V + \text{sort}(E))$	[15]

new ideas with ideas used in previous recursively defined cache-oblivious algorithms and data structures [38, 21], the buffer technique of Arge [11, 41], and the M/B-way merging scheme utilized by Brodal and Katajainen [29]. When the conference version of this paper appeared, our structure was the only cache-oblivious priority queue to obtain the same bounds as in the cache-aware case.

In the second part of the paper, section 3, we use our priority queue to develop several cache-oblivious graph algorithms. We first show how to solve the list ranking problem in $O(\text{sort}(V))$ memory transfers. Using this result we develop $O(\text{sort}(V))$ algorithms for fundamental problems on trees, such as the Euler tour, BFS, and DFS problems. The complexity of all of these algorithms matches the complexity of the best known cache-aware algorithms. Next we consider DFS and BFS on general graphs. Using modified versions of the data structures used in the $O((V + E/B)\log_2 V + \text{sort}(E))$ DFS and BFS algorithms for directed graphs [30], we make these algorithms cache-oblivious. We also discuss how the $O(V+\text{sort}(E))$ BFS algorithm for undirected graphs [46] can be made cache-oblivious. Very recently, and after the appearance of the conference version of this paper, Brodal et al. [28] developed two other cache-oblivious algorithms for undirected BFS based on the ideas in [45]. Finally, we develop two cache-oblivious algorithms for computing a minimum spanning forest (MSF), and thus also for computing connected components, of an undirected graph using $O(\text{sort}(E)\cdot\log_2\log_2 V)$ and $O(V + \text{sort}(E))$ memory transfers, respectively. The two algorithms can be combined to compute the MSF in $O(\text{sort}(E)\cdot\log_2\log_2\frac{V}{V'} + V')$ memory transfers for any V' independent of B and M. Table 1.1 summarizes our results. Note that, recently, cache-oblivious algorithms for undirected shortest path computation have also been developed [28, 33].

2. Priority queue. In this section we describe our optimal cache-oblivious priority queue. In section 2.1 we define the data structure and in section 2.2 we describe the supported operations.

2.1. Structure.

2.1.1. Levels. Our priority queue data structure containing N elements consists of $\Theta(\log\log N_0)$ *levels* whose sizes vary from $N_0 = \Theta(N)$ to some small size $c > 1$ beneath a constant threshold c_t. The size of a level corresponds (asymptotically) to the number of elements that can be stored within it. The ith level from above has size

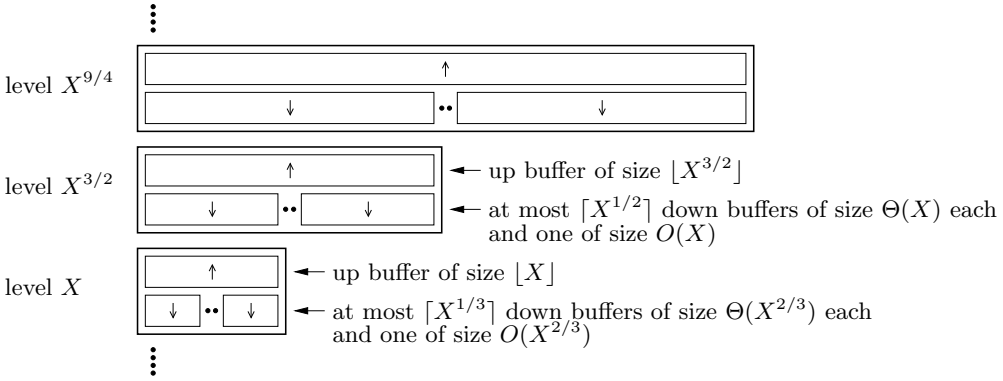


FIG. 2.1. *Levels X, $X^{3/2}$, and $X^{9/4}$ of the priority queue data structure.*

$N_0^{(2/3)^{i-1}}$, and for convenience we refer to the levels by their size. Thus the levels from largest to smallest are level N_0, level $N_0^{2/3}$, level $N_0^{4/9}$, ..., level $X^{9/4}$, level $X^{3/2}$, level X, level $X^{2/3}$, level $X^{4/9}$, ..., level $c^{9/4}$, level $c^{3/2}$, and level c. Intuitively, smaller levels store elements with smaller keys or elements that were recently inserted. In particular, the element with minimum key and the most recently inserted element in the structure are in the smallest (lowest) level c. Both insertions and deletions are initially performed on the smallest level and may propagate up through the levels.

2.1.2. Buffers. A level stores elements in a number of *buffers*, which are also used to transfer elements between levels. Refer to Figure 2.1. Level $X^{3/2}$ consists of one *up buffer* $u^{X^{3/2}}$ and at most $\lceil X^{1/2} \rceil + 1$ *down buffers* $d_1^{X^{3/2}}, \ldots, d_{\lceil X^{1/2} \rceil + 1}^{X^{3/2}}$. The up buffer can store up to $\lfloor X^{3/2} \rfloor$ elements and the first down buffer can store up to $2\lfloor X \rfloor - 1$ elements, while each of the other down buffers can store between $\lfloor X \rfloor$ and $2\lfloor X \rfloor - 1$ elements. We refer to the maximum possible number of elements that can be stored in a buffer or level as its *size*; we refer to the number of elements currently in a buffer or level as the *occupancy*. Thus the size of level $X^{3/2}$ is $\Theta(X^{3/2})$. Note that the size of a down buffer at one level matches the size (up to a constant factor) of the up buffer one level down.

We maintain three invariants about the relationships between the elements in buffers of various levels.

INVARIANT 1. *At level $X^{3/2}$, elements are sorted* among *the down buffers, that is, elements in $d_i^{X^{3/2}}$ have smaller keys than elements in $d_{i+1}^{X^{3/2}}$, but the elements within $d_i^{X^{3/2}}$ are unordered.*

The element with largest key in each down buffer $d_i^{X^{3/2}}$ is called a *pivot element*. Pivot elements mark the boundaries between the ranges of the keys of elements in down buffers.

INVARIANT 2. *At level $X^{3/2}$, elements in the down buffers have smaller keys than the elements in the up buffer $u^{X^{3/2}}$.*

INVARIANT 3. *The elements in the down buffers at level $X^{3/2}$ have smaller keys than the elements in the down buffers at the next higher level $X^{9/4}$.*

The three invariants ensure that the keys of the elements in the down buffers get larger as we go from smaller to larger levels of the structure. Furthermore, there is an order between the buffers on one level; keys of elements in the up buffer are larger than keys of elements in down buffers. Therefore, down buffers are drawn below up

buffers in Figure 2.1. However, the keys of the elements in an up buffer are unordered relative to the keys of the elements in down buffers one level up. Intuitively, up buffers store elements that are "on their way up"; that is, they have yet to be resolved as belonging to a particular down buffer in the next (or higher) level. Analogously, down buffers store elements that are "on their way down"—these elements are partitioned into several clusters so that we can quickly find the cluster of elements with smallest keys of size roughly equal to the next level down. In particular, the first down buffer of level X contains the smallest element in level X and higher levels.

2.1.3. Layout. We store the priority queue in a linear array as follows. The levels are stored consecutively from smallest to largest with each level occupying a single region of memory. For level $X^{3/2}$ we reserve space for the up buffers of size $\lfloor X^{3/2} \rfloor$ and for $\lceil X^{1/2} \rceil + 1$ possible down buffers of size $2\lfloor X \rfloor$. The up buffer is stored first, followed by the down buffers stored in an arbitrary order but linked together to form an ordered linked list. Thus the total size of the array is $\sum_{i=0}^{\log_{3/2} \log_c N_0} O(N_0^{(2/3)^i}) = O(N_0)$.

2.2. Operations. To implement the priority queue operations we will use two general operations, *push* and *pull*. Push inserts $\lfloor X \rfloor$ elements (with larger keys than all elements in the down buffers of level X) into level $X^{3/2}$, and pull removes and returns the $\lfloor X \rfloor$ elements with smallest keys from level $X^{3/2}$ (and above). Generally, whenever an up buffer on level X overflows we push the $\lfloor X \rfloor$ elements in the buffer into level $X^{3/2}$, and whenever the down buffers on level X become too empty we pull $\lfloor X \rfloor$ elements from level $X^{3/2}$. In sections 2.2.1 and 2.2.2 below, the push and pull operations are described in detail. Note that we will assume that we never push elements from level N_0 and that we never try to pull more elements from level N_0 than it contains, that is, that level N_0 never runs full or empty. As described in section 2.2.3, we will ensure this by periodically rebuilding the entire structure.

2.2.1. Push. We push $\lfloor X \rfloor$ elements (with larger keys than all elements in the down buffers of level X) into level $X^{3/2}$ as follows: We first sort the elements. Then we distribute them into the down buffers of level $X^{3/2}$ by scanning through the sorted list and simultaneously visiting the down buffers in (linked) order. More precisely, we append elements to the end of the current down buffer $d_i^{X^{3/2}}$, and advance to the next down buffer $d_{i+1}^{X^{3/2}}$ as soon as we encounter an element with larger keys than the pivot of $d_i^{X^{3/2}}$. Elements with larger keys than the pivot of the last down buffer are inserted in the up buffer $u^{X^{3/2}}$. During the distribution of elements a down buffer may become overfull, that is, contain $2\lfloor X \rfloor$ elements. In this case, we split the buffer into two down buffers each containing $\lfloor X \rfloor$ elements. If the level has at most $\lceil X^{1/2} \rceil + 1$ down buffers after the split, we place the new buffer in any free down-buffer spot for the level and update the linked list accordingly. Otherwise, we first remove the last down buffer by moving its at most $2\lfloor X \rfloor - 1$ elements into the up buffer; then we place the new buffer in the free down-buffer spot and update the linked list. If the up buffer runs full during the process, that is, contains more than $\lfloor X^{3/2} \rfloor$ elements, we recursively push $\lfloor X^{3/2} \rfloor$ elements into level $X^{9/4}$ (the next level up), leaving at most $\lfloor X^{3/2} \rfloor$ elements in the up buffer.

The invariants are all maintained during a push of $\lfloor X \rfloor$ elements into level $X^{3/2}$. Because we sort the elements to distribute them among the down buffers, it is clear we maintain Invariant 1. Only elements with keys larger than the pivot of the last down buffer are placed in the up buffer, so Invariant 2 is also maintained. Finally, Invariant 3 is maintained since (by definition) the $\lfloor X \rfloor$ elements all have keys larger than the

elements in the down buffers of level X, and since by Invariant 2 all recursively pushed elements have keys larger than all elements in the down buffers (as required to perform the recursive push).

Ignoring the cost of recursive push operations, a push of $\lfloor X \rfloor$ elements into level $X^{3/2}$ can be performed cache-obliviously in $O(\frac{X}{B} \log_{M/B} \frac{X}{B})$ memory transfers and $O(X \log_2 X)$ time using $O(M)$ of main memory. First note that since $M = \Omega(B^2)$ (the tall-cache assumption), all levels of size less than B^2 (of total size $O(B^2)$) fit in memory. If all these levels are kept in main memory at all times, all push costs associated with them would be eliminated. The optimal paging strategy is able to do so. Thus we only need to consider push costs when $X^{3/2} > B^2$, that is, when $X > B^{4/3}$ (note that in that case $\frac{X}{B} > 1$). When performing the push operation, the initial sort of the $\lfloor X \rfloor$ elements can then be performed cache-obliviously using $O(\frac{X}{B} \log_{M/B} \frac{X}{B})$ memory transfers and $O(X \log_2 X)$ time [38]. The scan of the elements in the distribution step then takes $O(X/B)$ memory transfers and $O(X)$ time. However, even though we do not insert elements in every down buffer, we still might perform a memory transfer for each of the $\lceil X^{1/2} \rceil + 1$ possible buffers; a block of each buffer may have to be loaded and written back without transferring a full block of elements into the buffer. If $X \geq B^2$, we trivially have that $\lceil X^{1/2} \rceil + 1 = O(\frac{X}{B})$. If, on the other hand, $B^{4/3} < X < B^2$, the $\lceil X^{1/2} \rceil + 1$ term can dominate the memory transfer bound and we have to analyze the cost more carefully. In this case we are working on a level $X^{3/2}$ where $B^2 < X^{3/2} < B^3$. There is only one such level and because $X^{1/2} < B$ and $M = \Omega(B^2)$ (the tall-cache assumption), a block for each of its down buffers can fit into main memory. Consequently, if a fraction of the main memory is used to keep a partially filled block of each buffer of level $X^{3/2}$ ($B^2 \leq X^{3/2} \leq B^3$) in memory at all times, and full blocks are written to disk, the $X^{1/2} + 1$ cost is eliminated at this level. The optimal paging strategy is able to do so. Thus the total cost of distributing the $\lfloor X \rfloor$ elements is $O(X/B)$ memory transfers and $O(X + X^{1/2}) = O(X)$ time.

The split of an overfull down buffer during the distribution, that is, split of a buffer of occupancy $2\lfloor X \rfloor$, can be performed in $O(X/B)$ memory transfers and $O(X)$ time by first finding the median of the elements in the buffer in $O(X/B)$ transfers and $O(X)$ time [38], and then partitioning the elements into the two new buffers of occupancy $\lfloor X \rfloor$ in a simple scan. Since we maintain that any new down buffer has occupancy $\lfloor X \rfloor$, and thus that $\lfloor X \rfloor$ elements have to be inserted in it before it splits, the amortized splitting cost per element is $O(1/B)$ transfers and $O(1)$ time. Thus, in total, the amortized number of memory transfers and time used on splitting buffers while distributing the $\lfloor X \rfloor$ elements are $O(X/B)$ and $O(X)$, respectively.

LEMMA 2.1. *Using $O(M)$ main memory, a push of $\lfloor X \rfloor$ elements into level $X^{3/2}$ can be performed in $O(\frac{X}{B} \log_{M/B} \frac{X}{B})$ memory transfers and $O(X \log_2 X)$ amortized time, not counting the cost of any recursive push operations, while maintaining Invariants* 1–3.

2.2.2. Pull. To pull the $\lfloor X \rfloor$ elements with smallest keys from level $X^{3/2}$ (and above), we consider three different cases.

If the occupancy of the first down buffer of level $X^{3/2}$ is $Y \geq \lfloor X \rfloor$, we sort the $Y < 2\lfloor X \rfloor$ elements in the down buffer, remove the $\lfloor X \rfloor$ elements with smallest keys, and leave the remaining $Y - \lfloor X \rfloor$ elements in the buffer. We return the $\lfloor X \rfloor$ removed elements, since by Invariants 1–3 they are the elements with smallest keys in level $X^{3/2}$ (and above). It is easy to see that Invariants 1–3 are maintained is this case.

If the occupancy of the first down buffer of level $X^{3/2}$ is $Y < \lfloor X \rfloor$ but level $X^{3/2}$ has at least one other down buffer, we first remove the Y elements in the first buffer.

Next we sort the between $\lfloor X \rfloor$ and $2\lfloor X \rfloor - 1$ elements in the new first down buffer, remove the $\lfloor X \rfloor - Y$ elements with smallest keys, and leave the remaining elements in the buffer. We return the $\lfloor X \rfloor$ removed elements, since by Invariants 1–3 they are the elements with smallest keys in level $X^{3/2}$ (and above). As in the first case, it is easy to see that Invariants 1–3 are maintained.

Finally, if the occupancy of the first down buffer of level $X^{3/2}$ is $Y < \lfloor X \rfloor$ and level $X^{3/2}$ has no other down buffers, we remove the Y elements and then we recursively pull the $\lfloor X^{3/2} \rfloor$ elements with smallest keys from level $X^{9/4}$ (and above). Because these $\lfloor X^{3/2} \rfloor$ elements do not necessarily have smaller keys than the U elements in the up buffer $u^{X^{3/2}}$, we then sort all the $\lfloor X^{3/2} \rfloor + U \leq 2\lfloor X^{3/2} \rfloor$ elements, insert the U elements with largest keys in the up buffer, and remove the $\lfloor X \rfloor - Y$ elements with smallest keys. Finally, we distribute the remaining $\lfloor X^{3/2} \rfloor + Y - \lfloor X \rfloor < \lfloor X^{3/2} \rfloor \leq X \cdot X^{1/2} \leq (\lfloor X \rfloor + 1) \cdot X^{1/2} = \lfloor X \rfloor \cdot X^{1/2} + X^{1/2} \leq \lfloor X \rfloor \cdot \lceil X^{1/2} \rceil + \lfloor X \rfloor$ sorted elements into one down buffer with occupancy between 1 and $\lfloor X \rfloor$ and at most $\lceil X^{1/2} \rceil$ down buffers of occupancy $\lfloor X \rfloor$ each. As in the first two cases, we return the $\lfloor X \rfloor$ removed elements, since Invariant 1 and the sorting of the recursively pulled elements and the elements in the up buffer ensure that they are the elements with the smallest keys in level $X^{3/2}$ (and above). As in the first two cases, it is relatively easy to see that Invariants 1–3 are also maintained in this case: Invariant 1 is fulfilled since we place elements in down buffers in sorted order, Invariant 2 is fulfilled since we explicitly place the U elements (among the original U elements and the $\lfloor X^{3/2} \rfloor$ recursively pulled elements) with largest keys in the up buffer, and Invariant 3 is fulfilled since all the elements we place in down buffers have smaller keys than elements in level $X^{9/4}$.

To analyze a pull operation we assume, as in the push case, that all levels of size $O(B^2)$ are kept in main memory (so that we only need to consider levels $X^{3/2}$ with $X^{3/2} > B^2$, that is, the case where $X > B^{4/3}$). The first two cases are both performed by sorting and scanning $O(X)$ elements using $O(\frac{X}{B} \log_{M/B} \frac{X}{B}) + O(\frac{X}{B})$ memory transfers and $O(X \log_2 X) + O(X)$ time. In the third case we also sort and scan (distribute) $O(X^{3/2})$ elements. However, the cost of doing so is dominated by the cost of the recursive pull operation itself. Thus, ignoring these costs (charging them to the recursive pull), we have the following lemma.

LEMMA 2.2. *Using $O(M)$ main memory, a pull of $\lfloor X \rfloor$ elements from level $X^{3/2}$ can be performed in $O(\frac{X}{B} \log_{M/B} \frac{X}{B})$ memory transfers and $O(X \log_2 X)$ amortized time, not counting the cost of any recursive pull operations, while maintaining Invariants* 1–3.

2.2.3. Insert and delete-min. To support insert and delete-min operations using push and pull on our structure we (or rather, the optimal paging strategy does) maintain an insertion and a deletion buffer of at most $\lfloor c^{2/3} \rfloor$ elements each in main memory. The deletion buffer contains elements that have smaller keys than all other elements in the structure, while intuitively the insertion buffer contains the most recently inserted elements. We perform an insertion simply by comparing the key of the element to be inserted with the maximal key of an element in the deletion buffer: if the key of the new element is largest, we simply insert it into the insertion buffer; otherwise we insert it into the deletion buffer and move the element with largest key from the deletion buffer to the insertion buffer. In both cases, the occupancy of the insertion buffer is increased by one, and if it runs full we empty it by pushing its $\lfloor c^{2/3} \rfloor$ elements into level c. Similarly, we perform a delete-min by deleting and returning the element with smallest key in the deletion buffer; if the deletion buffer becomes

empty, we pull $\lfloor c^{2/3} \rfloor$ elements from level c and fill up the deletion buffer with the $\lfloor c^{2/3} \rfloor$ smallest of these elements and the elements in the insertion buffer (without changing the occupancy of the insertion buffer). The correctness of the insert and delete-min operations follows directly from Invariants 1–3 and the definition of the push and pull operations.

Except for the possible push and pull operations on level c, which may require recursive pushes or pulls on higher levels, the insert and delete-min operations are performed in constant time without incurring any memory transfers. Below we will use a credit argument to prove that including all push and pull operations the amortized cost of an insert or delete-min is $O(\frac{1}{B} \log_{M/B} \frac{N_0}{B})$ memory transfers; then we will argue that the operations take $O(\log_2 N_0)$ amortized computation time.

We define a *level-X push coin* as well as a *level-$X^{3/2}$ pull coin* to be worth $\Theta(\frac{1}{B} \log_{M/B} \frac{X}{B})$ memory transfers each, that is, $\lfloor X \rfloor$ level-X push coins can pay for a push of $\lfloor X \rfloor$ elements into level $X^{3/2}$, and $\lfloor X \rfloor$ level-$X^{3/2}$ coins can pay for a pull of $\lfloor X \rfloor$ elements from level $X^{3/2}$. We maintain the following *coin invariants*.

INVARIANT 4. *On level $X^{3/2}$, each element in the first half of a down buffer has a pull coin for level $X^{3/2}$ and each level below.*

INVARIANT 5. *On level $X^{3/2}$, each element in the second half of a down buffer and in the up buffer has a push coin for level $X^{3/2}$ and each level above, as well as pull coins for all levels.*

INVARIANT 6. *Each element in the insertion buffer has a push and a pull coin for each level.*

Intuitively, the first two coin invariants mean that an element in the first half of a down buffer can pay for being pulled down through all lower levels, while elements in the second half of a down buffer and in an up buffer can pay for being pushed up through all higher levels and pulled down through all levels.

To fulfill Invariant 6, we simply have to give each inserted element a push and a pull coin for each level, since an insert operation increases the occupancy of the insertion buffer by one and a delete-min does not change the occupancy. We will show that we can then pay for all recursive push and pull operations with released coins while maintaining Invariants 4 and 5. Thus a delete-min is free amortized and an insertion costs $O(\sum_{i=0}^{\infty} \frac{1}{B} \log_{M/B}(N_0^{(2/3)^i}/B)) = O(\frac{1}{B} \log_{M/B} \frac{N_0}{B})$ amortized memory transfers.

First consider an insertion that results in a sequence of push operations. We will show that we can maintain the coin invariants while paying for each push operation with released coins, if we require that when pushing $\lfloor X \rfloor$ elements into level $X^{3/2}$, each of the pushed elements has a push coin for level X and each level above, as well as a pull coin for all levels (the *push requirement*). Note that Invariants 5 and 6 ensure that the push requirement is fulfilled, since the pushed elements come from the insertion buffer or the up buffer of level X.

When performing a push on level $X^{3/2}$ we first distribute the $\lfloor X \rfloor$ elements into the down and the up buffers. In the worst case (when all elements are placed in the second half of a down buffer or in the up buffer) each element needs a push coin for level $X^{3/2}$ and each level above and pull coins for all levels to fulfill Invariants 4 and 5. Since they have a push coin for level X and each level above and pull coins for all levels, this leaves us with $\lfloor X \rfloor$ level-X push coins, which we can use to pay the $O(\frac{X}{B} \log_{M/B} \frac{X}{B})$ push cost (Lemma 2.1). If a down buffer of occupancy $2\lfloor X \rfloor$ splits into two buffers of occupancy $\lfloor X \rfloor$ during the distribution process, $\lfloor X \rfloor$ push coins for level $X^{3/2}$ and each level above and $\lfloor X \rfloor$ pull coins for level $X^{9/4}$ and each level above

are released, since the $\lfloor X \rfloor$ elements in the second half of the full buffer must fulfill Invariant 5 before the split but only Invariant 4 after the split. On the other hand, if splitting a down buffer results in the movement of the elements in the last down buffer to the up buffer, the at most $\lfloor X \rfloor$ elements in the first half of the down buffer need $\lfloor X \rfloor$ push coins for level $X^{3/2}$ and each level above and $\lfloor X \rfloor$ pull coins for level $X^{9/4}$ and each level above, since they must fulfill Invariant 5 after the move but only Invariant 4 before the move. Since we never move elements from a down buffer to the up buffer without having split a down buffer, we can reestablish Invariants 4 and 5 with the released coins. Finally, as mentioned above, if the up buffer runs full and we perform a recursive push of $\lfloor X^{3/2} \rfloor$ elements into level $X^{9/4}$, each of the pushed elements has a push coin for level $X^{3/2}$ and each level above, as well as pull coins for all levels, as required (Invariant 5).

Next consider a delete-min that results in a sequence of pull operations. We will show that we can maintain the coin invariants while paying for each pull operation with released coins, if we require that when pulling $\lfloor X \rfloor$ elements from level $X^{3/2}$ down to level X, each of the pulled elements has a pull coin for level X and each level below (the *pull requirement*).

When performing a pull on level $X^{3/2}$ with at least $\lfloor X \rfloor$ elements in the down buffers (the first two cases), we effectively remove the $\lfloor X \rfloor$ smallest elements from the first two down buffers. It is straightforward to see that the remaining elements still fulfill Invariants 4 and 5. From Invariants 4 and 5 we know that each of the removed (pulled) elements (at least) has a pull coin for level $X^{3/2}$ and each level below. Thus, since they only need a pull coin for level X and each level below to fulfill the pull requirement, this leaves us with $\lfloor X \rfloor$ level-$X^{3/2}$ pull coins, which we can use to pay the $O(\frac{X}{B} \log_{M/B} \frac{X}{B})$ pull cost (Lemma 2.2). If, on the other hand, level $X^{3/2}$ contains $Y < \lfloor X \rfloor$ elements in the down buffers (the third case), we perform a recursive pull of $\lfloor X^{3/2} \rfloor$ elements from level $X^{9/4}$, and effectively remove the $\lfloor X \rfloor$ elements with smallest keys among the $\lfloor X^{3/2} \rfloor + Y$ elements. Before the recursive pull, level $X^{3/2}$ has one down buffer with Y elements and an up buffer with U elements; after the recursive pull and removal of the $\lfloor X \rfloor$ elements, it has one down buffer with fewer than $\lfloor X \rfloor$ elements, at most $\lceil X^{1/2} \rceil$ down buffers with $\lfloor X \rfloor$ elements, and an up buffer with U elements. The coins on the U elements in the up buffer before the recursive pull can be used to fulfill Invariant 5 for the U elements in the up buffer after the recursive pull. By the pull requirement, each of the $\lfloor X^{3/2} \rfloor$ pulled elements has a pull coin for level $X^{3/2}$ and each level below. The same is true for each of the Y original elements (Invariant 4). Thus we have enough coins for all the elements in the down buffers after the pull, since each down buffer contains at most $\lfloor X \rfloor$ elements and each element therefore only needs to fulfill Invariant 4. Similarly, since each of the $\lfloor X \rfloor$ removed (pulled) elements only needs a pull coin for level X and each level below to fulfill the pull requirement, the pull cost can (as in the first two cases) be payed by the remaining $\lfloor X \rfloor$ level-$X^{3/2}$ pull coins.

The above argument shows that all pushes and pulls can be paid if each inserted element is given a push and a pull coin for each level of the structure, that is, that a delete-min is free amortized and an insertion costs $O(\frac{1}{B} \log_{M/B} \frac{N_0}{B})$ amortized memory transfers. By a completely analogous argument, it is easy to see that the operations are performed in $O(\sum_{i=0}^{\infty} \log_2(N_0^{(2/3)^i})) = O(\log_2 N_0)$ amortized time.

Finally, to maintain that $N_0 = \Theta(N)$ we completely rebuild the structure bottom-up after every $N_0/4$ operations (often referred to as *global rebuilding* [47]). We choose the size N_0 of the largest level to be $2N$ and compute the largest value $c < c_t$ such

that $c^{(3/2)^i} = 2N$ for some integer i. Then we construct levels $c, c^{3/2}, \ldots, N_0^{2/3}$ such that all up buffers are empty and such that each level-$X^{3/2}$ has exactly $\lceil X^{1/2} \rceil$ down buffers of size $\lfloor X \rfloor$. The remaining elements are placed in level N_0 such that it has at most $\lceil N_0^{1/3} \rceil$ down buffers of size exactly $\lfloor N_0^{2/3} \rfloor$ and one of size less than $\lfloor N_0^{2/3} \rfloor$. We can easily perform the rebuilding in a sorting and a scanning step using a total of $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ memory transfers. At the same time, we can place pull coins on the elements in order to fulfill Invariant 4; no elements need to fulfill Invariant 5—in particular, no push coins are needed. The cost of these pull coins is bounded by the cost if all elements were placed in level N_0, that is, by $O(N_0 \cdot \sum_{i=1}^{\infty} \frac{1}{B} \log_{M/B}(N_0^{(2/3)^i}/B)) = O(\frac{N_0}{B} \log_{M/B} \frac{N_0}{B})$ memory transfers. Thus the rebuilding adds only $O(\frac{1}{B} \log_{M/B} \frac{N_0}{B})$ amortized transfers to the cost of an insert or delete-min operation. In the same way we can argue that it adds only $O(\log_2 N_0)$ amortized time per operation.

Since the up buffer of level N_0 is of size $\lfloor 2N \rfloor = 2N$ after the rebuilding, we will not need further levels during the next $N_0/4 = N/2$ operations (insertions). At the same time, the size N_0 of the largest level remains $\Theta(N)$ during the next $N/2$ operations (deletions). Thus the size of our structure remains linear in N and it supports insert and delete-min operations in $O(\frac{1}{B} \log_{M/B} \frac{N}{B})$ amortized memory transfers and $O(\log_2 N)$ amortized computation time.

LEMMA 2.3. *Using $O(M)$ main memory, a set of N elements can be maintained in a linear-space cache-oblivious priority queue data structure supporting each insert and delete-min operation in $O(\frac{1}{B} \log_{M/B} \frac{N}{B})$ amortized memory transfers and $O(\log_2 N)$ amortized computation time.*

2.2.4. Delete. Using standard techniques we can also easily support a delete operation in $O(\frac{1}{B} \log_{M/B} \frac{N}{B})$ memory transfers and $O(\log_2 N)$ amortized time, provided that we are given the id and key of the element to be deleted.

To perform a deletion, we simply insert a special delete element in the priority queue, with id and key equal to the element to be deleted. Furthermore, whenever two elements in the queue need to be compared (for example when sorting elements during push or pull operations) their keys are first compared; in case of equal keys their id's are then compared. When performing a delete-min we first remove the smallest key element from the deletion buffer in main memory. Then we consider the removed element and the new smallest key element in the deletion buffer: If the two elements have the same id's (which means that one of the elements is a special delete element) we also remove the new smallest key element (and delete both elements without reporting any of them). We repeat this process of looking at the two smallest key elements until they have different id's; in that case we return and delete the removed element. Finally, we modify the rebuilding algorithm (performed after every $N_0/4$ operations) to first remove all special delete elements and the elements they should delete, before choosing a new N_0 and building the new structure. We can easily do so in a simple sorting and scanning step without changing the asymptotic rebuilding cost, since the way we compare elements (keys) ensures that an element that should have been deleted and its corresponding special deletion element appear consecutively in the list of sorted elements. Note that this also means that the structure cannot contain more than a constant fraction of special elements and elements that should have been deleted (that is, N_0 is always $\Theta(N)$).

The correctness of the delete (and delete-min) operation(s) follows directly from the discussion in the previous subsection and the fact that if the smallest key element

in the deletion buffer should really have been deleted, then the elements and the corresponding special delete element will appear consecutively in the buffer. To analyze the modified structure, we first note that a delete operation basically behaves like an insertion. As previously, all pushes and pulls can therefore be paid if each special delete element is also given a push and a pull coin for each level of the structure. Thus the cost of an insertion or deletion is $O(\frac{1}{B}\log_{M/B}\frac{N}{B})$ amortized memory transfers. That the modified delete-min operation is free amortized then follows, since it basically consists of a sequence of the unmodified delete-min operations (where the deleted elements are not reported).

THEOREM 2.4. *Using $O(M)$ main memory, a set of N elements can be maintained in a linear-space cache-oblivious priority queue data structure supporting each insert, delete-min, and delete operation in $O(\frac{1}{B}\log_{M/B}\frac{N}{B})$ amortized memory transfers and $O(\log_2 N)$ amortized computation time.*

3. Graph algorithms. In this section we discuss how our cache-oblivious priority queue can be used to develop several cache-oblivious graph algorithms. We first consider the simple list ranking problem and algorithms on trees, and then we go on and consider BFS, DFS, and MSF algorithms for general graphs.

3.1. List ranking. In the list ranking problem we are given a linked list of V nodes stored as an unordered sequence. More precisely, we have an array with V nodes, each containing the position of the next node in the list (an edge). The goal is to determine the *rank* of each node v, that is, the number of edges from v to the end of the list. In a more general version of the problem, each edge has a weight and the goal is to find for each node v the sum of the weights of edges from v to the end of the list. Refer to Figure 3.1.

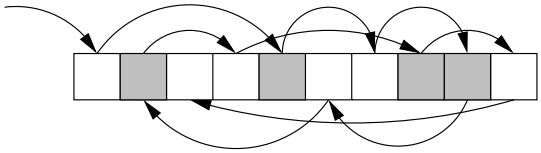


FIG. 3.1. *List ranking problem. An independent set of size* 4 *is marked. There are two forward lists (on top) and two backward lists (on bottom).*

Based on ideas from efficient PRAM algorithms [8, 34], Chiang et al. [31] designed an $O(\text{sort}(V))$ I/O model list ranking algorithm. The main idea in the algorithm is as follows. An *independent set* of $\Theta(V)$ nodes (nodes without edges to each other) is found, nodes in the independent set are "bridged out" (edges incident to nodes in this set are contracted), the remaining list is recursively ranked, and finally the contracted nodes are reintegrated into the list (their ranks are computed). The main innovation in the algorithm by Chiang et al. [31] was an $O(\text{sort}(V))$ memory transfer algorithm for computing an independent set of size V/c for some constant $c > 0$. The rest of the nonrecursive steps of the algorithm can easily be performed in $O(\text{sort}(V))$ memory transfers using a few scans and sorts of the nodes of the list as follows. To bridge out the nodes in the independent set, we first identify nodes with a successor in the independent set. We do so by creating a copy of the list of nodes, sorting it by successor position, and simultaneously scanning the two lists. During this process, we can also mark each predecessor of an independent set node v with the position of the successor w of v, as well as with the weight of the edge (v, w). Next, in a simple scan, we create a new list where the two edges incident to each independent set node

v have been replaced with an edge from the predecessor of v to the successor of v. The new edge has weight equal to the sum of that of the two old edges. Finally, we create the list to be ranked recursively by removing the independent set nodes and "compressing" the remaining nodes, that is, storing them in an array of size $V(1-1/c)$. We do so by scanning through the list while creating a list of the nodes not in the independent set, as well as a list that indicates the old and new position of each node (that is, the position of each node in the old and new array). Then we update the successor fields of the first list by sorting it by successor position, and simultaneously scanning it and the list of new positions. After having ranked the compressed list recursively, we reintegrate the removed nodes while computing their ranks. The rank of an independent set node v is simply the rank of its successor w minus the weight of edge (v, w). The reintegration of the independent set nodes can be performed in a few scans and sorts similar to the way we bridged out the independent set. Overall, not counting the independent set computation, the number of memory transfers used to rank a V node list is $T(V) = O(\text{sort}(V)) + T(V/c) = O(\text{sort}(V))$.

Since we only use scans and sorts in the above algorithm, all that remains in order to obtain a cache-oblivious list ranking algorithm is to develop a cache-oblivious independent set algorithm. Under different assumptions about the memory and block size, Chiang et al. [31] developed several independent set algorithms based on 3-coloring; in a 3-coloring every node is colored with one of three colors such that adjacent nodes have different colors. The independent set (of size at least $V/3$) then consists of the set of nodes with the most popular color. Arge [11] and Kumar and Schwabe [41] later removed the main memory and block assumptions.

One way of computing a 3-coloring is as follows [11, 31]: We call an edge (v, w) a *forward* edge if v appears before w in the (unordered) sequence of nodes—otherwise it is called a *backward* edge. First we imagine splitting the list into two sets consisting of forward running segments (forward lists) and backward running segments (backward lists). Each node is included in at least one of these sets, and nodes at the head or tail of a segment (nodes at which there is a reversal of the direction) will be in both sets. Refer to Figure 3.1. Next we color the nodes in the forward lists red or blue by coloring the head nodes red and the other nodes alternately blue and red. Similarly, the nodes in the backward lists are colored green and blue, with the head nodes being colored green. In total, every node is colored with one color, except for the heads/tails, which have two colors. It is easy to see that we obtain a 3-coloring if we color each head/tail node the color it was colored as the head of a list [31].

In the above 3-coloring algorithm we can cache-obliviously color the forward lists as follows (the backward lists can be colored similarly). In a single sort and scan we identify the head nodes, and for each such node v we insert a red element in a cache-oblivious priority queue with key equal to the position of v in the unordered list. We then repeatedly extract the minimal key element e from the queue. If e corresponds to a node v, we access v in the list, color it the same color as e, and insert an element corresponding to its successor in the queue. We color the inserted element in the opposite color of e. After processing all elements in the queue we have colored all forward lists. The initial sort and scan is performed cache-obliviously in $O(\text{sort}(V))$ memory transfers, and since we use a cache-oblivious priority queue we can also perform the $O(V)$ priority queue operations in $O(\text{sort}(V))$ memory transfers. Apart from this, we also perform what appears to be random accesses to the $O(V)$ nodes in the list. However, since we only process the forward list nodes in position order, the accesses overall end up corresponding to a scan of the list. Thus they only

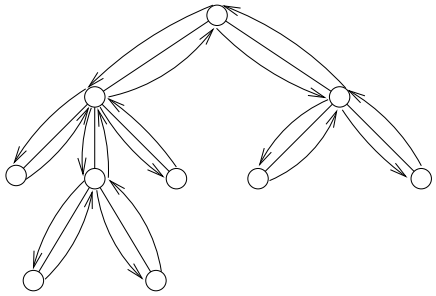


FIG. 3.2. *An undirected tree and an Euler tour of the corresponding directed graph.*

require $O(V/B)$ transfers. Therefore, we can compute a 3-coloring cache-obliviously in $O(\text{sort}(V))$ memory transfers, and thus overall we obtain the following.

THEOREM 3.1. *The list ranking problem on a V node list can be solved cache-obliviously in $O(\text{sort}(V))$ memory transfers.*

3.2. Algorithms on trees. Many efficient PRAM algorithms on undirected trees use Euler tour techniques [56, 58]. An Euler tour of a graph is a cycle that traverses each edge exactly once. Not every graph has an Euler tour, but a tree where each undirected edge has been replaced with two directed edges does (refer to Figure 3.2). When, in the following, we refer to an Euler tour of an undirected tree, we mean a tour in the graph obtained by replacing each edge in the tree with two directed edges.

To cache-obliviously compute an Euler tour of an undirected tree, that is, to compute an ordered list of the edges along the tour, we use ideas from similar PRAM algorithms. Consider imposing a (any) cyclic order on the nodes adjacent to each node v in the tree. In [51] it is shown that an Euler tour is obtained if we traverse the tree such that a visit to v from u (through the incoming edge (u, v)) is followed by a visit to the node w following u in the cyclic order (through the outgoing edge (v, w)). Thus we can compute the successor edge of each edge e, that is, the edge following e in the Euler tour, as follows: We first construct a list of incoming edges to each node v sorted according to the cyclic order. If two edges (u, v) and (w, v) are stored next to each other in this list, the successor edge for the (incoming) edge (u, v) is simply the (outgoing) edge (v, w). Therefore, we can compute all successor edges in a scan of the list. Given a list of all edges augmented with their successor edge, we can then compute the Euler tour simply by ranking the list and the sorting the edges by their rank. Thus overall we compute an Euler tour of an undirected tree using a few sorts and scans and a list ranking step, that is, using $O(\text{sort}(V))$ memory transfers.

Using our cache-oblivious Euler tour algorithm, we can easily compute a DFS numbering of the nodes of a tree starting at a source node s [51]. First note that if we start a walk of the Euler tour in the source node s it is actually a DFS tour of the tree. To compute the numbering, we therefore first classify each edge as being either a *forward* or a *backward* edge; an edge is a forward edge if it is traversed before its reverse in the tour. After numbering the edges along the tour and sorting the list of edges such that reverse edges appear consecutively, we can classify each edge in a simple scan of the list. Then we assign each forward edge weight 1 and each backward edge weight 0. The DFS number of a node v is then simply the sum of the weights on the edges from s to v. Thus we can obtain the DFS numbering by solving the general version of list ranking. Since we only use Euler tours computation, list

ranking, sorting, and scanning, we compute the DFS numbering cache-obliviously in a total of $O(\text{sort}(V))$ memory transfers.

Using an Euler tour, list ranking, and sorting, we can also compute a BFS numbering of the nodes of a tree cache-obliviously in $O(\text{sort}(V))$ memory transfers in a similar way [51]. Using standard PRAM ideas, and the tools developed here, we can also, e.g., compute the centroid decomposition of a tree in $O(\text{sort}(V))$ memory transfers [56, 58, 31]. The centroid of a tree is the node that, if removed, minimizes the size of the largest of the remaining subtrees. The centroid decomposition of a tree is a recursive partition of a tree into subtrees around the centroid.

THEOREM 3.2. *The Euler tour, BFS, DFS, and centroid decomposition problems on a tree with V nodes can be solved cache-obliviously in $O(\text{sort}(V))$ memory transfers.*

3.3. DFS and BFS. We now consider the DFS and BFS numbering problems for general graphs. We first describe a cache-oblivious DFS algorithm for directed graphs and then we modify it to compute a BFS numbering. Finally, we develop an improved BFS algorithm for undirected graphs.

3.3.1. Depth-first search. In the RAM model, directed DFS can be solved in linear time using a stack S containing vertices v that have not yet been visited but have an edge (w, v) incident to a visited vertex w, as well as an array A containing an element for each vertex v, indicating if v has been visited or not. The top vertex v of S is repeatedly popped and A is accessed to determine if v has already been visited. If v has not yet been visited, it is marked as visited in A and all vertices adjacent to v are pushed onto S. It is easy to realize that if the stack S is implemented using a doubling array then a *push* or *pop* requires $O(1/B)$ amortized cache-oblivious memory transfers, since the optimal paging strategy can always keep the last block of the array (accessed by both push and pop) in main memory. However, each access to A may require a separate memory transfer resulting in $\Omega(E)$ memory transfers in total.

In the I/O model, Chiang et al. [31] modified the above algorithm to obtain an $O(V + \frac{E}{B}\frac{V}{M})$ algorithm. In their algorithm all visited vertices (marked vertices in array A) are stored in main memory. Every time the number of visited vertices grows larger than the main memory, all visited vertices and all their incident edges are removed from the graph. Since this algorithm relies crucially on knowledge of the main memory size, it seems hard to make it cache-oblivious. Buchsbaum et al. [30] described another $O((V + \frac{E}{B}) \log_2 V + \text{sort}(E))$ I/O model algorithm. In the following we describe how to make it cache-oblivious. The algorithm uses a number of data structures: V priority queues, a stack, and a so-called *buffered repository tree*. As discussed above, a stack can trivially be made cache-oblivious. Below we first describe how to make the buffered repository tree cache-oblivious. Next we describe what we call a *buffered priority tree* that we use in the algorithm rather than our cache-oblivious priority queue; we cannot simply use our priority queue since it requires $O(M)$ space. Finally, we describe the algorithm by Buchsbaum et al. [30] and how the use of the cache-oblivious structures leads to a cache-oblivious version of it.

Buffered repository tree. A buffered repository tree (BRT) maintains $O(E)$ elements with keys in the range $[1..V]$ under operations *insert* and *extract*. The insert operation inserts a new element, while the extract operation reports and deletes all elements with a certain key.

Our cache-oblivious version of the BRT consists of a static binary tree with the keys 1 through V in sorted order in the leaves. A buffer is associated with each node and leaf of the tree. The buffer of a leaf v contains elements with key v and the buffers

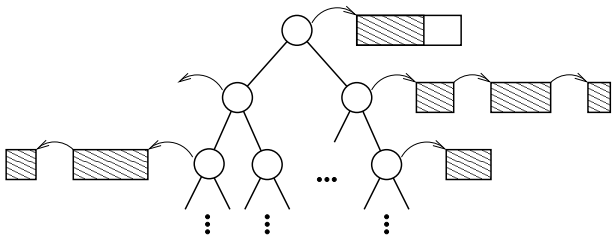


FIG. 3.3. *Buffered repository tree (BRT). Each nonroot node has a buffer of elements stored as a linked list of buckets. The root node buffer is implemented using a doubling array.*

of the internal nodes are used to perform insertions in a batched manner. We perform an insertion simply by inserting the new element into the root buffer. To perform an extraction of elements with key v we traverse the path from the root to the leaf containing v. At each node μ on this path we scan the associated buffer and report and delete elements with key v. During the scan we also distribute the remaining elements among the two buffers associated with the children of μ; we distribute an element with key w to the buffer of the child of μ on the path to w. We place elements inserted in a buffer during the same scan consecutively in memory (but not necessarily right after the other elements in the buffer). This way the buffer of a node μ can be viewed as consisting of a linked list of *buckets* of elements in consecutive memory locations, with the number of buckets being bounded by the number of times the buffer of μ's parent buffer has been emptied since the last time μ's buffer was emptied. To avoid performing a memory transfer on each insertion, we implement the root buffer slightly differently, namely as a doubling array (like a stack). Since only the root buffer is implemented this way, the optimal paging strategy can keep the last block of the array in main memory and we obtain an $O(1/B)$ amortized root buffer insertion bound. Refer to Figure 3.3 for an illustration of a BRT.

LEMMA 3.3. *A cache-oblivious BRT uses $\Theta(B)$ main memory space and supports insert and extract operations in $O(\frac{1}{B}\log_2 V)$ and $O(\log_2 V)$ amortized memory transfers, respectively.*

Proof. As discussed, an insertion in the root buffer requires $O(1/B)$ amortized memory transfers. During an extract operation, we use $O(X/B+K)$ memory transfers to empty the buffer of a node μ containing X elements in K buckets (since we access each element and each bucket). We charge the X/B-term to the insert operations that inserted the X elements in the BRT. Since each element is charged at most once on each level of the tree, an insert is charged $O(\frac{1}{B}\log_2 V)$ transfers. We charge the K-term to the extract operations that created the K buckets. Since an extract operation creates two buckets on each level of the tree, it is charged a total of $O(\log_2 V)$ memory transfers. □

Buffered priority tree. A buffered priority tree is constructed on E_v elements with E_v distinct keys, and maintains these elements under *buffered delete* operations. Given E' elements, all of which are currently stored in the structure, a buffered delete operation first deletes the E' elements and then deletes and reports the minimal key element among the remaining elements in the structure.

The buffered priority tree is implemented similarly to the BRT. It consists of a static binary tree with the E_v keys in sorted order in the leaves, where a buffer is associated with each internal node. Initially, the E_v elements are stored in the leaves containing their keys. The buffers are used to store elements intended to be

deleted from the structure, and with each node v we maintain a counter storing the number of (undeleted) elements stored in the tree rooted in v. To perform a buffered delete we first insert the E' elements in the buffer of the root and update its counter. Next we traverse the path from the root to the leaf l containing the minimal key elements (among the undeleted elements), while emptying the buffers associated with the encountered nodes: At the root, we scan the buffer and distribute the elements among the two buffers associated with the children (exactly as in the BRT). During the distribution, we also update the counters in the two children. If the counter of the left child v_l is still greater than zero, i.e., the minimal element is in the tree rooted in v_l, we then recursively visit v_l. Otherwise we visit the right child v_r. (Note that when reaching leaf l it has an empty buffer.) After finding l, we report and delete the element stored in l. Finally, we decrement the counters on the path from the root to l.

LEMMA 3.4. *Using no permanent main memory, a cache-oblivious buffered priority tree supports buffered deletes of E' elements in $O((\frac{E'}{B}+1)\log_2 V)$ amortized memory transfers. It can be constructed in $O(\mathrm{sort}(E_v))$ memory transfers.*

Proof. The number of transfers needed to construct the tree is dominated by the $O(\mathrm{sort}(E_v))$ memory transfers needed to sort the E_v elements and construct the leaves; after that, the tree can be constructed in $O(E_v/B)$ transfers level-by-level bottom-up. The amortized cost of a buffered delete is equal to the cost of inserting E' elements into a BRT, plus the cost of extracting an element from a BRT, that is, $O((\frac{E'}{B}+1)\log_2 V)$ memory transfers. □

DFS algorithm. As mentioned, the directed DFS numbering algorithm by Buchsbaum et al. [30] utilizes a number of data structures: a stack S containing vertices on the path from the root of the DFS tree to the current vertex, a priority queue $P(v)$ for each vertex v containing edges (v, w) connecting v with a possibly unvisited vertex w, as well as one BRT D containing edges (v, w) incident to a vertex w that has already been visited but where (v, w) is still present in $P(v)$. The key of an edge (v, w) is v in D and w in $P(v)$. For the priority queues $P(v)$ we use our buffered priority tree.

Initially each $P(v)$ is constructed on the E_v edges (v, w) incident to v, the source vertex is placed on the stack S, and the BRT D is empty. To compute a DFS numbering, the vertex u on the top of the stack is repeatedly considered. All edges in D of the form (u, w) are extracted and deleted from $P(u)$ using a buffered delete operation. If $P(u)$ is now empty, so that no minimal element (edge (u, v)) is returned, all neighbors of u have already been visited and u is popped off S. Otherwise, if edge (u, v) is returned, vertex v is visited next: It is numbered and pushed on S, and all edges (w, v) (with $w \neq u$) incident to it are inserted in D. In [30] it is shown that this algorithm correctly computes a DFS numbering.

To analyze the above algorithm, we first note that each vertex is considered on the top of S a number of times equal to one greater than its number of children in the DFS tree. Thus the total number of times we consider a vertex on top of S is $2V - 1$. When considering a vertex u we first perform a stack operation on S, an extract operation on D, and a buffered delete operation on $P(u)$. The stack operation requires $O(1/B)$ memory transfers and the extraction $O(\log_2 V)$ transfers, since the optimal paging strategy can keep the relevant $O(1)$ blocks of S and D in main memory at all times. Thus overall these costs add up to $O(V \log_2 V)$. The buffered delete operation requires $O((1+\frac{E'}{B})\log_2 V)$ memory transfers if E' is the number of deleted elements (edges). Each edge is deleted once, so overall the buffered deletes costs add up to $O((V+\frac{E}{B})\log_2 V)$. Next we insert the, say E'', edges incident to v in D. This

requires $O(1 + \frac{E''}{B} \log_2 V)$ memory transfers, or $O(V + \frac{E}{B} \log_2 V)$ transfers over the whole algorithm. (Note that the E'' edges are not immediately deleted directly from the relevant $P(w)$'s since that could cost a memory transfer per edge.) In addition, the initial construction of the buffered priority trees requires $O(\text{sort}(E))$ memory transfers. Thus overall the algorithm uses $O((V + \frac{E}{B}) \log_2 V + \text{sort}(E))$ memory transfers.

3.3.2. Breadth-first search. The DFS algorithm described above can be modified to perform a BFS simply by replacing the stack S with a queue. Queues, like stacks, can be implemented using a doubling array, and the optimal paging strategy can keep the two partial blocks in use by the *enqueue* and *dequeue* operations in memory, such that each queue operation requires $O(1/B)$ amortized memory transfers. Thus we also obtain an $O((V + \frac{E}{B}) \log_2 V)$ directed BFS numbering algorithm.

Our directed DFS and BFS algorithms can of course also be used on undirected graphs. For undirected graphs, improved $O(V + \text{sort}(E))$ and $O(\sqrt{\frac{V \cdot E}{B}} + \text{sort}(E))$ I/O model algorithms have been developed [46, 45]. The idea in the algorithm by Munagala and Ranade [46], which can immediately be made cache-oblivious, is to visit the vertices in "layers" of vertices of equal distance from the source vertex s. The algorithm utilizes the fact that in an undirected graph any vertex adjacent to a vertex in layer i is either in layer $i-1$, layer i, or layer $i+1$. It maintains two sorted lists of vertices in the last two layers i and $i-1$. To create a sorted list of vertices in layer $i+1$, a list of possible layer $i+1$ vertices is first produced by collecting all vertices with a neighbor in level i (using a scan of the adjacency lists of layer i vertices). Then this list is sorted, and in a scan of the list and the (sorted) lists of vertices in level i and $i-1$ all previously visited vertices are removed. Apart from the sorting steps, overall this algorithm uses $O(V + E/B)$ memory transfers to access the edge lists for all vertices, as well as $O(E/B)$ transfers to scan the lists. Since each vertex is included in a sort once for each of its incident edges, the total cost of all sorting steps is $O(\text{sort}(E))$. Thus in total the algorithm uses $O(V + \text{sort}(E))$ memory transfers. Since it uses only scans and sorts, it is cache-oblivious without modification. Refer to [46] for full details. As mentioned in the introduction, Brodal et al. [28] have developed other undirected BFS algorithms based on the ideas in [45].

THEOREM 3.5. *The DFS or BFS numbering of a directed graph can be computed cache-obliviously in* $O((V + \frac{E}{B}) \log_2 V + \text{sort}(E))$ *memory transfers. The BFS numbering of an undirected graph can be computed cache-obliviously in* $O(V + \text{sort}(E))$ *memory transfers.*

3.4. Minimum spanning forest. In this section we consider algorithms for computing the MSF of an undirected weighted graph. Without loss of generality, we assume that all edge weights are distinct. In the I/O model, a sequence of algorithms has been developed for the problem [31, 1, 41, 15], culminating in an algorithm using $O(\text{sort}(E) \cdot \log_2 \log_2(\frac{VB}{E}))$ memory transfers developed by Arge, Brodal, and Toma [15]. This algorithm consists of two phases. In the first phase, an edge contraction algorithm inspired by PRAM algorithms [32, 35] is used to reduce the number of vertices to $O(E/B)$. In the second phase, a modified version of Prim's algorithm [48] is used to complete the MSF computation. Using our cache-oblivious priority queue we can relatively easily modify both of the phases to work cache-obliviously. However, since we cannot decide cache-obliviously when the first phase has reduced the number of vertices to $O(E/B)$, we are not able to combine the two phases as effectively as in the I/O model. Below, we first describe how to make the algorithms used in the two phases cache-oblivious. Then we discuss their combination.

3.4.1. Phase 1. The basic edge contraction based MSF algorithm proceeds in stages [32, 31, 41]. In each stage the minimum weight edge incident to each vertex is selected and output as part of the MSF, and the vertices connected by the selected edges are contracted into supervertices (that is, the connected components of the graph of selected edges are contracted). See, e.g., [15] for a proof that the selected edges along with the edges in a MSF of the contracted graph constitute a MSF for the original graph.

In the following we sketch how we can perform a contraction stage on a graph G cache-obliviously in $O(\text{sort}(E))$ memory transfers as in [15]. We can easily select the minimum weight edges in $O(\text{sort}(E))$ memory transfers using a few scans and sorts. To perform the contraction, we select a *leader vertex* in each connected component of the graph G_s of selected edges, and replace every edge (u, v) in G with the edge $(\mathit{leader}(u), \mathit{leader}(v))$. To select the leaders, we use the fact that the connected components of G_s are trees, except that one edge in each component (namely, the minimal weight edge) appears twice [15]. In each component, we simply use one of the vertices incident to the edge appearing twice as leader. This way we can easily identify all the leaders in $O(\text{sort}(E))$ memory transfers using a few sorts and scans (by identifying all edges that appear twice). We can then use our cache-oblivious tree algorithms developed in section 3.2 to distribute the identity of the leader to each vertex in each component in $O(\text{sort}(V))$ memory transfers: We add an edge between each leader in G_s and a pseudo root vertex s and perform a DFS numbering of the resulting tree starting in s; since all vertices in the same connected component (tree) will have consecutive DFS numbers, we can then mark each vertex with its leader using a few sorts and scans. Finally, after marking each vertex v with $\mathit{leader}(v)$, we can easily replace each edge (u, v) in G with $(\mathit{leader}(u), \mathit{leader}(v))$ in $O(\text{sort}(E))$ memory transfers using a few sort and scan steps on the vertices and edges.

Since each contraction stage reduces the number of vertices by a factor of two, and since a stage is performed in $O(\text{sort}(E))$ memory transfers, we can reduce the number of vertices to $V' = V/2^i$ in $O(\text{sort}(E) \cdot \log_2(V/V'))$ memory transfers by performing i stages after each other. Thus we obtain an $O(\text{sort}(E) \cdot \log_2 V)$ algorithm by continuing the contraction until we are left with a single vertex. In the I/O model, Arge, Brodal, and Toma [15] showed how to improve this bound to $O(\text{sort}(E) \cdot \log_2 \log_2 V)$ by grouping the stages into "superstages" and working only on a subset of the edges of G in each superstage. The extra steps involved in their improvement are all sorting or scanning of the edges and vertices, and therefore the improvement is immediately cache-oblivious.

LEMMA 3.6. *The MFS of an undirected weighted graph can be computed cache-obliviously in* $O(\text{sort}(E) \cdot \log_2 \log_2 V)$ *memory transfers.*

3.4.2. Phase 2. Prim's algorithm [48] grows a minimum spanning tree (MST) of a connected graph iteratively from a source vertex using a priority queue P on the vertices not already included in the MST. The key of a vertex v in P is equal to the weight of the minimal weight edge connecting v to the current MST. In each step of the algorithm a delete-min is used to obtain the next vertex u to add to the MST, and the keys of all neighbors of u in P are (possibly) updated. A standard implementation of this algorithm uses $\Omega(E)$ memory transfers, since a transfer is needed to obtain the current key of each neighbor vertex. In the I/O model, Arge, Brodal, and Toma [15] showed how to modify the algorithm to use $O(V + \text{sort}(E))$ memory transfers by storing edges rather than vertices in the priority queue. Below we describe this algorithm in order to show that it can be implemented cache-obliviously.

Like Prim's algorithm, the algorithm in [15] grows the MST iteratively. We maintain a priority queue P containing (at least) all edges connecting vertices in the current MST with vertices not in the tree; P can also contain edges between two vertices in the MST. Initially it contains all edges incident to the source vertex. In each step of the algorithm we extract the minimum weight edge (u, v) from P. If v is already in the MST we discard the edge; otherwise we include v in the MST and insert all edges incident to v, except (v, u), in the priority queue. We can efficiently determine if v is already in the MST, since if u and v are both already in the MST, then (u, v) must be in the priority queue twice; thus, if the next edge we extract from P is (v, u), then v is already in the MST.

The correctness of the above algorithm follows immediately from the correctness of Prim's algorithm. During the algorithm we access the edges in the adjacency list of each vertex v once (when v is included in the MST) for a total of $O(V+E/B)$ memory transfers. We also perform $O(E)$ priority queue operations, for a total of $O(\text{sort}(E))$ memory transfers. Thus the algorithm uses $O(V + \text{sort}(E))$ memory transfers. All of the above can easily be modified to compute a MSF for an unconnected graph rather than a MST for a connected graph. Thus we have obtained the following.

LEMMA 3.7. *The MSF of an undirected weighted graph can be computed cache-obliviously in $O(V + \text{sort}(E))$ memory transfers.*

3.4.3. Combined algorithm. In the I/O model, an $O(\text{sort}(E) \cdot \log_2 \log_2(\frac{VB}{E}))$ MSF algorithm can be obtained by running the phase 1 algorithm until the number of vertices has been reduced to $V' = E/B$ using $O(\text{sort}(E) \cdot \log_2 \log_2(\frac{VB}{E}))$ memory transfers, and then finishing the MSF in $O(V' + \text{sort}(E)) = O(\text{sort}(E))$ memory transfers using the phase 2 algorithm. As mentioned, we cannot combine the two phases as effectively in the cache-oblivious model. In general, however, we can combine the two algorithms to obtain an $O(\text{sort}(E) \cdot \log_2 \log_2(V/V') + V')$ algorithm for any V' independent of B and M.

THEOREM 3.8. *The MSF of an undirected weighted graph can be computed cache-obliviously in $O(\text{sort}(E) \cdot \log_2 \log_2(V/V') + V')$ memory transfers for any V' independent of B and M.*

4. Conclusions. In this paper, we presented an optimal cache-oblivious priority queue and used it to develop efficient cache-oblivious algorithms for several graph problems. We believe that the ideas utilized in the development of the priority queue and our graph algorithms will prove useful in the development of other cache-oblivious data structures.

Many important problems still remain open in the area of cache-oblivious algorithms and data structures. In the area of graph algorithms, for example, it remains open to develop a cache-oblivious MSF algorithm with complexity matching the best known cache-aware algorithm.

Acknowledgments. The authors wish to thank an anonymous reviewer for pointing out a mistake in an earlier draft of this paper, and all the reviewers for numerous suggestions for improving the presentation.

REFERENCES

[1] J. ABELLO, A. L. BUCHSBAUM, AND J. R. WESTBROOK, *A functional approach to external graph algorithms*, Algorithmica, 32 (2002), pp. 437–458.

[2] P. K. AGARWAL, L. ARGE, A. DANNER, AND B. HOLLAND-MINKLEY, *Cache-oblivious data structures for orthogonal range searching*, in Proceedings of the ACM Symposium on Computational Geometry, 2003, pp. 237–245.

[3] A. Aggarwal, B. Alpern, A. K. Chandra, and M. Snir, *A model for hierarchical memory*, in Proceedings of the ACM Symposium on Theory of Computation, 1987, pp. 305–314.

[4] A. Aggarwal and A. K. Chandra, *Virtual memory algorithms*, in Proceedings of the ACM Symposium on Theory of Computation, 1988, pp. 173–185.

[5] A. Aggarwal, A. K. Chandra, and M. Snir, *Hierarchical memory with block transfer*, in Proceedings of the IEEE Symposium on Foundations of Computer Science, 1987, pp. 204–216.

[6] A. Aggarwal and J. S. Vitter, *The input/output complexity of sorting and related problems*, Commun. ACM, 31 (1988), pp. 1116–1127.

[7] B. Alpern, L. Carter, E. Feig, and T. Selker, *The uniform memory hierarchy model of computation*, Algorithmica, 12 (1994), pp. 72–109.

[8] R. J. Anderson and G. L. Miller, *A simple randomized parallel algorithm for list-ranking*, Inform. Process. Lett., 33 (1990), pp. 269–273.

[9] L. Arge, *The I/O-complexity of ordered binary-decision diagram manipulation*, in Algorithms and Computations, Lecture Notes in Comput. Sci. 1004, Springer, Berlin, 1995, pp. 82–91. A complete version appears as BRICS Technical Report RS-96-29, University of Aarhus, Denmark.

[10] L. Arge, *External memory data structures*, in Handbook of Massive Data Sets, J. Abello, P. M. Pardalos, and M. G. C. Resende, eds., Kluwer Academic Publishers, Norwell, MA, 2002, pp. 313–358.

[11] L. Arge, *The buffer tree: A technique for designing batched external data structures*, Algorithmica, 37 (2003), pp. 1–24.

[12] L. Arge, M. Bender, E. Demaine, B. Holland-Minkley, and J. I. Munro, *Cache-oblivious priority queue and graph algorithm applications*, in Proceedings of the ACM Symposium on Theory of Computation, 2002, pp. 268–276.

[13] L. Arge, G. S. Brodal, and R. Fagerberg, *Cache-oblivious data structures*, in Handbook on Data Structures and Applications, D. Mehta and S. Sahni, eds., CRC Press, Boca Raton, FL, 2005.

[14] L. Arge, G. S. Brodal, R. Fagerberg, and M. Laustsen, *Cache-oblivious planar orthogonal range searching and counting*, in Proceedings of the ACM Symposium on Computational Geometry, 2005, pp. 160–169.

[15] L. Arge, G. S. Brodal, and L. Toma, *On external memory MST, SSSP and multi-way planar graph separation*, J. Algorithms, 53 (2004), pp. 186–206.

[16] L. Arge, M. de Berg, and H. Haverkort, *Cache-oblivious R-trees*, in Proceedings of the ACM Symposium on Computational Geometry, 2005, pp. 170–179.

[17] R. Bayer and E. McCreight, *Organization and maintenance of large ordered indexes*, Acta Inform., 1 (1972), pp. 173–189.

[18] M. Bender, R. Cole, E. Demaine, and M. Farach-Colton, *Scanning and traversing: Maintaining data for traversals in memory hierarchy*, in Proceedings of the European Symposium on Algorithms, 2002, Lecture Notes in Comput. Sci. 2461, Springer, Berlin, 2002, pp. 152–164.

[19] M. A. Bender, G. S. Brodal, R. Fagerberg, D. Ge, S. He, H. Hu, J. Iacono, and A. López-Ortiz, *The cost of cache-oblivious searching*, in Proceedings of the IEEE Symposium on Foundations of Computer Science, 2003, pp. 271–282.

[20] M. A. Bender, R. Cole, and R. Raman, *Exponential structures for cache-oblivious algorithms*, in Proceedings of the 29th International Colloquium on Automata, Languages, and Programming, Malaga, Spain, 2002, Lecture Notes in Comput. Sci. 2380, Springer, Berlin, pp. 195–207.

[21] M. A. Bender, E. D. Demaine, and M. Farach-Colton, *Cache-oblivious B-trees*, in Proceedings of the IEEE Symposium on Foundations of Computer Science, 2000, pp. 339–409.

[22] M. A. Bender, Z. Duan, J. Iacono, and J. Wu, *A locality-preserving cache-oblivious dynamic dictionary*, in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2002, pp. 29–38.

[23] R. D. Blumofe, M. Frigo, C. F. Joerg, C. E. Leiserson, and K. H. Randall, *An analysis of dag-consistent distributed shared-memory algorithms*, in Proceedings of the ACM Symposium on Parallel Algorithms and Architectures, 1996, pp. 297–308.

[24] G. S. Brodal and R. Fagerberg, *Cache oblivious distribution sweeping*, in Proceedings of the 29th International Colloquium on Automata, Languages, and Programming, Malaga, Spain, 2002, Lecture Notes in Comput. Sci. 2380, Springer, Berlin, pp. 426–438.

[25] G. S. Brodal and R. Fagerberg, *Funnel heap—A cache oblivious priority queue*, in Algorithms and Computation, Lecture Notes in Comput. Sci. 2518, Springer, Berlin, 2002, pp. 219–228.

[26] G. S. Brodal and R. Fagerberg, *On the limits of cache-obliviousness*, in Proceedings of the ACM Symposium on Theory of Computation, 2003, pp. 307–315.

[27] G. S. Brodal, R. Fagerberg, and R. Jacob, *Cache oblivious search trees via binary trees of small height*, in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2002, pp. 39–48.

[28] G. S. Brodal, R. Fagerberg, U. Meyer, and N. Zeh, *Cache-oblivious data structures and algorithms for undirected breadth-first search and shortest paths*, in Algorithm Theory—SWAT 2004, Lecture Notes in Comput. Sci. 3111, Springer, Berlin, 2004, pp. 480–492.

[29] G. S. Brodal and J. Katajainen, *Worst-case efficient external-memory priority queues*, in Algorithm Theory—SWAT '98, Lecture Notes in Comput. Sci. 1432, Springer, Berlin, 1998, pp. 107–118.

[30] A. L. Buchsbaum, M. Goldwasser, S. Venkatasubramanian, and J. R. Westbrook, *On external memory graph traversal*, in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2000, pp. 859–860.

[31] Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, and J. S. Vitter, *External-memory graph algorithms*, in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1995, pp. 139–149.

[32] F. Chin, J. Lam, and I. Chen, *Efficient parallel algorithms for some graph problems*, Commun. ACM, 25 (1982), pp. 659–665.

[33] R. A. Chowdhury and V. Ramachandran, *Cache-oblivious shortest paths in graphs using buffer heap*, in Proceedings of the ACM Symposium on Parallel Algorithms and Architectures, ACM Press, New York, 2004, pp. 245–254.

[34] R. Cole and U. Vishkin, *Deterministic coin tossing with applications to optimal parallel list-ranking*, Inform. and Control, 70 (1986), pp. 32–53.

[35] R. Cole and U. Vishkin, *Approximate parallel scheduling.* II. *Applications to logarithmic-time optimal parallel graph algorithms*, Inform. and Comput., 92 (1991), pp. 1–47.

[36] D. Comer, *The ubiquitous B-tree*, ACM Computing Surveys, 11 (1979), pp. 121–137.

[37] R. Fadel, K. V. Jakobsen, J. Katajainen, and J. Teuhola, *Heaps and heapsort on secondary storage*, Theoret. Comput. Sci., 220 (1999), pp. 345–362.

[38] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran, *Cache-oblivious algorithms*, in Proceedings of the IEEE Symposium on Foundations of Computer Science, 1999, pp. 285–298.

[39] S. Huddleston and K. Mehlhorn, *A new data structure for representing sorted lists*, Acta Inform., 17 (1982), pp. 157–184.

[40] D. E. Knuth, *Sorting and Searching*, The Art of Computer Programming, Vol. 3, 2nd ed., Addison-Wesley, Reading, MA, 1998.

[41] V. Kumar and E. Schwabe, *Improved algorithms and data structures for solving graph problems in external memory*, in Proceedings of the IEEE Symposium on Parallel and Distributed Processing, 1996, pp. 169–177.

[42] R. Ladner, J. Fix, and A. LaMarca, *Cache performance analysis of traversals and random accesses*, in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1999, pp. 613–622.

[43] A. LaMarca and R. Ladner, *The influence of cache on the performance of sorting*, J. of Algorithms, 31 (1999), pp. 66–104.

[44] A. LaMarca and R. E. Ladner, *The influence of caches on the performance of heaps*, ACM J. Exp. Algorithmics, 1 (1996) (electronic).

[45] K. Mehlhorn and U. Meyer, *External-memory breadth-first search with sublinear I/O*, in Algorithms—ESA 2002, Lecture Notes in Comput. Sci. 2461, Springer, New York, 2002, pp. 723–735.

[46] K. Munagala and A. Ranade, *I/O-complexity of graph algorithms*, in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1999, pp. 687–694.

[47] M. H. Overmars, *The Design of Dynamic Data Structures*, Lecture Notes in Comput. Sci. 156, Springer-Verlag, New York, 1983.

[48] R. C. Prim, *Shortest connection networks and some generalizations*, Bell Syst. Tech. J., 36 (1957), pp. 1389–1401.

[49] N. Rahman, R. Cole, and R. Raman, *Optimised predecessor data structures for internal memory*, in Algorithm Engineering, Lecture Notes in Comput. Sci. 2141, Springer, New York, 2001, pp. 67–78.

[50] N. Rahman and R. Raman, *Analysing cache effects in distribution sorting*, in Algorithm Engineering, Lecture Notes in Comput. Sc. 1668, Springer-Verlag, Berlin, 1999, pp. 183–197.

[51] J. H. Reif, ed., *Synthesis of Parallel Algorithms*, Morgan Kaufmann, San Francisco, 1993, pp. 61–114.

[52] P. Sanders, *Fast priority queues for cached memory*, in Algorithm Engineering and Experimentation, Lecture Notes in Comput. Sci. 1619, Springer, Berlin, 1999, pp. 312–327.
[53] J. E. Savage, *Extending the Hong-Kung model to memory hierarchies*, in Computing and Combinatorics, Lecture Notes in Comput. Sci. 959, Springer, New York, 1995, pp. 270–281.
[54] S. Sen, S. Chatterjee, and N. Dumir, *Towards a theory of cache-efficient algorithms*, J. ACM, 49 (2002), pp. 828–858.
[55] D. D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, Commun. ACM, 28 (1985), pp. 202–208.
[56] R. E. Tarjan and U. Vishkin, *An efficient parallel biconnectivity algorithm*, SIAM J. Comput., 14 (1985), pp. 862–874.
[57] S. Toledo, *Locality of reference in LU decomposition with partial pivoting*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 1065–1081.
[58] U. Vishkin, *On efficient parallel strong orientation*, Inform. Processing Lett., 20 (1985), pp. 235–240.
[59] J. S. Vitter, *External memory algorithms and data structures: Dealing with MASSIVE data*, ACM Computing Surveys, 33 (2001), pp. 209–271.
[60] R. Wickremesinghe, L. Arge, J. S. Chase, and J. S. Vitter, *Efficient sorting using registers and caches*, ACM J. Exp. Algorithmics, 7 (2002) (electronic).
[61] N. Zeh, *I/O-Efficient Algorithms for Shortest Path Related Problems*, Ph.D. thesis, Carleton University, Ottawa, ON, 2002.

SIAM J. COMPUT.
Vol. 36, No. 6, pp. 1696–1708

ONLINE LEARNING AND RESOURCE-BOUNDED DIMENSION: WINNOW YIELDS NEW LOWER BOUNDS FOR HARD SETS*

JOHN M. HITCHCOCK†

Abstract. We establish a relationship between the online mistake-bound model of learning and resource-bounded dimension. This connection is combined with the Winnow algorithm to obtain new results about the density of hard sets under adaptive reductions. This improves previous work of Fu [*SIAM J. Comput.*, 24 (1995), pp. 1082–1090] and Lutz and Zhao [*SIAM J. Comput.*, 30 (2000), pp. 1197–1210], and solves one of Lutz and Mayordomo's "twelve problems in resource-bounded measure" [*Bull. Eur. Assoc. Theor. Comput. Sci. EATSC*, 68 (1999), pp. 64–80].

Key words. computational complexity, polynomial-time reductions, resource-bounded dimension, resource-bounded measure, sparse sets

AMS subject classification. 68Q15

DOI. 10.1137/050647517

1. Introduction.

This paper has two main contributions: (i) establishing a close relationship between resource-bounded dimension and Littlestone's online mistake-bound model of learning, and (ii) using this relationship along with the Winnow algorithm to resolve an open problem in computational complexity. In this introduction we briefly describe these contributions.

1.1. Online learning and dimension.

Lindner, Schuler, and Watanabe [15] studied connections between computational learning theory and resource-bounded measure, primarily working with the probably approximately correct (PAC) model. They also included the observation that any "admissible" subclass of P/poly that is polynomial-time learnable in Angluin's exact learning model [2] must have p-measure 0. The proof of this made use of the essential equivalence between Angluin's model and Littlestone's online mistake-bound model [16].

In the online mistake-bound model, a learner is presented a sequence of examples and is asked to predict whether or not they belong to some unknown target concept. The concept is drawn from some concept class, which is known to the learner, and the examples may be chosen by an adversary. After making a prediction about each example, the learner is told the correct classification for the example, and learner may use this knowledge in making future predictions. The mistake bound of the learner is the maximum number of incorrect predictions the learner will make, over any choice of target concept and sequence of examples.

We push the observation of [15] much further, developing a powerful, general framework for showing that classes have resource-bounded *dimension* 0. Resource-bounded measure and dimension involve betting on the membership of strings in an unknown set. To prove that a class has dimension 0, we show that it suffices to give a reduction to a family of concept classes that has a good mistake-bound learning algorithm. It is possible that the reduction can take exponential-time and that the

*Received by the editors December 13, 2005; accepted for publication (in revised form) September 20, 2006; published electronically March 19, 2007. This research was supported in part by National Science Foundation grant 0515313.

http://www.siam.org/journals/sicomp/36-6/64751.html

†Department of Computer Science, University of Wyoming, Laramie, WY 82071 (jhitchco@cs.uwyo.edu).

learning algorithm can also take exponential-time, as long as the mistake bound of the algorithm is subexponential. If we have a reduction from the unknown set to a concept in the learnable concept class, we can view the reduction as generating a sequence of examples, apply the learning algorithm to these examples, and use the learning algorithm's predictions to design a good betting strategy. Formal details of this framework are given in section 3.

1.2. Density of hard sets. The two most common notions of polynomial-time reductions are many-one ($\leq^{\mathrm{p}}_{\mathrm{m}}$) and Turing ($\leq^{\mathrm{p}}_{\mathrm{T}}$). A many-one reduction from A to B maps instances of A to instance of B, preserving membership. A Turing reduction from A to B makes many, possibly adaptive, queries to B in order to solve A. Many-one reductions are a special case of Turing reductions. In between $\leq^{\mathrm{p}}_{\mathrm{m}}$ and $\leq^{\mathrm{p}}_{\mathrm{T}}$ is a wide variety of polynomial-time reductions of different strengths.

A common use of reductions is to demonstrate hardness for a complexity class. Let $\leq^{\mathrm{p}}_{\tau}$ be a polynomial-time reducibility. For any set B, let $\mathrm{P}_{\tau}(B) = \{A \mid A \leq^{\mathrm{p}}_{\tau} B\}$ be the class of all problems that $\leq^{\mathrm{p}}_{\tau}$-reduce to B. We say that B is $\leq^{\mathrm{p}}_{\tau}$-*hard* for a complexity class $\mathcal{C}$ if $\mathcal{C} \subseteq \mathrm{P}_{\tau}(B)$, that is, every problem in $\mathcal{C}$ $\leq^{\mathrm{p}}_{\tau}$-reduces to B. For a class $\mathcal{D}$ of sets, a useful notation is $\mathrm{P}_{\tau}(\mathcal{D}) = \bigcup_{B\in\mathcal{D}} \mathrm{P}_{\tau}(B)$.

A problem B is *dense* if there exists $\epsilon > 0$ such that $|B_{\leq n}| > 2^{n^{\epsilon}}$ for all but finitely many n. All known hard sets for the exponential-time complexity classes $\mathrm{E} = \mathrm{DTIME}(2^{O(n)})$ or $\mathrm{EXP} = \mathrm{DTIME}(2^{n^{O(1)}})$ are dense. Whether every hard set must be dense has been often studied. First, Meyer (see his result reported by Berman and Hartmanis [5, pp. 317–318]) showed that every $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard set for E must be dense, and he observed that proving the same for $\leq^{\mathrm{p}}_{\mathrm{T}}$ reductions would imply that E does not have polynomial-size circuits. Since then, a line of research has obtained results for a variety of reductions between $\leq^{\mathrm{p}}_{\mathrm{m}}$ and $\leq^{\mathrm{p}}_{\mathrm{T}}$, specifically the conjunctive ($\leq^{\mathrm{p}}_{\mathrm{c}}$) and disjunctive ($\leq^{\mathrm{p}}_{\mathrm{d}}$) reductions, and for various functions $f(n)$, the bounded query $\leq^{\mathrm{p}}_{f(n)-\mathrm{tt}}$ and $\leq^{\mathrm{p}}_{f(n)-\mathrm{T}}$ reductions:

1. Watanabe [27, 10] showed that every hard set for E under the $\leq^{\mathrm{p}}_{\mathrm{c}}$, $\leq^{\mathrm{p}}_{\mathrm{d}}$, or $\leq^{\mathrm{p}}_{O(\log n)-\mathrm{tt}}$ reductions is dense.
2. Lutz and Mayordomo [21] showed that for all $\alpha < 1$, the class $\mathrm{P}_{n^{\alpha}-\mathrm{tt}}(\mathrm{DENSE}^c)$ has p-measure 0, where DENSE is the class of all dense sets. Since E does not have p-measure 0, their result implies that every $\leq^{\mathrm{p}}_{n^{\alpha}-\mathrm{tt}}$-hard set for E is dense.
3. Fu [8] showed that for all $\alpha < 1/2$, every $\leq^{\mathrm{p}}_{n^{\alpha}-\mathrm{T}}$-hard set for E is dense, and that for all $\alpha < 1$, every $\leq^{\mathrm{p}}_{n^{\alpha}-\mathrm{T}}$-hard set for EXP is dense.
4. Lutz and Zhao [23] gave a measure-theoretic strengthening of Fu's results, showing that for all $\alpha < 1/2$, $\mathrm{P}_{n^{\alpha}-\mathrm{T}}(\mathrm{DENSE}^c)$ has p-measure 0, and that for all $\alpha < 1$, $\mathrm{P}_{n^{\alpha}-\mathrm{T}}(\mathrm{DENSE}^c)$ has p_2-measure 0.

This contrast between E and EXP in the last two references was left as a curious open problem and stated as follows by Lutz and Mayordomo [22, Problem 6] as one of their "twelve problems in resource-bounded measure."

> *For* $\frac{1}{2} \leq \alpha < 1$, *is it the case that* $\mathrm{P}_{n^{\alpha}-\mathrm{T}}(\mathrm{DENSE}^c)$ *has* p-measure 0 (*or at least, that* $\mathrm{E} \not\subseteq \mathrm{P}_{n^{\alpha}-\mathrm{T}}(\mathrm{SPARSE})$)?

We resolve this problem, showing the much stronger conclusion that the classes in question have p-dimension 0. But first, in section 4, we prove a theorem about disjunctive reductions that illustrates the basic idea of our technique. We show that the class $\mathrm{P}_{\mathrm{d}}(\mathrm{DENSE}^c)$ has p-dimension 0. The proof uses the learning framework

of section 3 and Littlestone's Winnow algorithm [16]. Suppose that $A \leq^{\mathrm{p}}_{\mathrm{d}} S$, where S is a nondense set. Then there is a reduction g mapping strings to sets of strings such that $x \in A$ if and only if at least one string in $g(x)$ belongs to S. We view the reduction g as generating examples that we can use to learn a disjunction based on S. Because S is subexponentially dense, the target disjunction involves a subexponential number of variables out of exponentially many variables. This is truly a case "when irrelevant attributes abound" [16], and the Winnow algorithm perfoms exceedingly well to establish our dimension result. In the same section we also use the learning framework to show that $\mathrm{P_c}(\mathrm{DENSE}^c)$ has p-dimension 0. These results give new proofs of Watanabe's aforementioned theorems about $\leq^{\mathrm{p}}_{\mathrm{d}}$-hard and $\leq^{\mathrm{p}}_{\mathrm{c}}$-hard sets for E.

Our main theorem, presented in section 5, is that for all $\alpha < 1$, $\mathrm{P}_{n^\alpha-\mathrm{T}}(\mathrm{DENSE}^c)$ has p-dimension 0. This substantially improves the results of [21, 8, 23]. The resource-bounded measure proofs in [21, 23] use the concept of *weak stochasticity*. As observed by Mayordomo [25], this stochasticity approach can be extended to show a $-1st$-*order scaled dimension* [12] result, but it seems a different technique is needed for an (unscaled) dimension result. Our learning framework turns out to be just what is needed. We reduce the class $\mathrm{P}_{n^\alpha-\mathrm{T}}(\mathrm{DENSE}^c)$ to a family of learnable disjunctions. For this, we make use of a technique that Allender et al. [1] used to prove a surprising result converting bounded-query reductions to sparse sets into disjunctive reductions to sparse sets: $\mathrm{P_{btt}}(\mathrm{SPARSE}) \subseteq \mathrm{P_d}(\mathrm{SPARSE})$. Carefully applying the same technique on a sublinear-query Turing-reduction to a nondense set results in a disjunction with a nearly exponential blowup, but it can still be learned by Winnow in our dimension setting.

The density of complete and hard sets for NP has also been studied often, with motivation coming originally from the Berman–Hartmanis isomorphism conjecture [5]: all many-one complete sets are dense if the isomorphism conjecture holds. Since no absolute results about the density of NP-complete or NP-hard sets can be proved without separating P from NP, the approach has been to prove conditional results under a hypothesis on NP. Mahaney [24] showed that if $\mathrm{P} \neq \mathrm{NP}$, then no sparse set is $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard for NP. Ogiwara and Watanabe [26] extended Mahaney's theorem to the $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard sets. Deriving a result from $\mathrm{P} \neq \mathrm{NP}$ about NP-hard sets under unbounded truth-table reductions is still an open problem, but a measure-theoretic assumption yields very strong consequences. Lutz and Zhao [23] showed that under the hypothesis "NP does not have p-measure 0," every $\leq^{\mathrm{p}}_{n^\alpha-\mathrm{T}}$-hard set for NP must be dense, for all $\alpha < 1$. In section 6 we present the same conclusion under the weaker hypothesis "NP has positive p-dimension," and some additional consequences.

2. Preliminaries. The set of all binary strings is $\{0,1\}^*$. The length of a string $x \in \{0,1\}^*$ is $|x|$. We write λ for the empty string. For $n \in \mathbb{N}$, $\{0,1\}^n$ is the set of strings of length n and $\{0,1\}^{\leq n}$ is the set of strings of length at most n. We write $s_0 = \lambda$, $s_1 = 0$, $s_2 = 1$, $s_3 = 00$, ... for the standard lexicographic enumeration of $\{0,1\}^*$.

A *language* is a subset $L \subseteq \{0,1\}^*$. We write $L_{\leq n} = L \cap \{0,1\}^{\leq n}$ and $L_{=n} = L \cap \{0,1\}^n$. We say that L is *sparse* if there is a polynomial $p(n)$ such that for all n, $|L_{=n}| \leq p(n)$. We say that L is *(exponentially) dense* if there is a constant $\epsilon > 0$ such that $|L_{\leq n}| > 2^{n^\epsilon}$ for all sufficiently large n. We write SPARSE and DENSE for the classes of all sparse languages and all dense languages. The complement DENSE^c of DENSE is the class of all *nondense* languages.

2.1. Polynomial-time reductions. We use the following standard notions of polynomial-time reducibilities:

- *Turing reducibility:* $A \leq_{\mathrm{T}}^{\mathrm{p}} B$ if there is a polynomial-time oracle machine M such that $A = L(M^B)$.
- *Truth-table reducibility:* $A \leq_{\mathrm{tt}}^{\mathrm{p}} B$ if there is a polynomial-time oracle machine M that makes nonadaptive queries such that $A = L(M^B)$.
- *Disjunctive reducibility:* $A \leq_{\mathrm{d}}^{\mathrm{p}} B$ if there is a polynomial-time computable $f : \{0,1\}^* \to \mathcal{P}(\{0,1\}^*)$ such that for all x, $x \in A$ if and only if $f(x) \cap B \neq \emptyset$.
- *Conjunctive reducibility:* $A \leq_{\mathrm{c}}^{\mathrm{p}} B$ if there is a polynomial-time computable $f : \{0,1\}^* \to \mathcal{P}(\{0,1\}^*)$ such that for all x, $x \in A$ if and only if $f(x) \subseteq B$.

We write $\leq_{q(n)-\mathrm{T}}^{\mathrm{p}}$ or $\leq_{q(n)-\mathrm{tt}}^{\mathrm{p}}$ to indicate that the reduction makes at most $q(n)$ queries on any input of length n. The bounded reducibility $A \leq_{\mathrm{btt}}^{\mathrm{p}} B$ means $A \leq_{k-\mathrm{tt}}^{\mathrm{p}} B$ for some constant k.

Let $\leq_\tau^{\mathrm{p}}$ be a polynomial-time reducibility. For any language B, we define $\mathrm{P}_\tau(B) = \{A \mid A \leq_\tau^{\mathrm{p}} B\}$. A language B is $\leq_\tau^{\mathrm{p}}$*-hard* for a class $\mathcal{C}$ if $\mathcal{C} \subseteq \mathrm{P}_\tau(B)$. For any class $\mathcal{D}$ of languages, $\mathrm{P}_\tau(\mathcal{D}) = \bigcup_{B \in \mathcal{D}} \mathrm{P}_\tau(B)$.

2.2. Resource-bounded measure and dimension. Resource-bounded measure and dimension were introduced by Lutz [17, 19], and resource-bounded strong dimension by Athreya et al. [4]. Here we briefly review the definitions and basic properties. We refer to the original sources and also the surveys [18, 22, 20, 13] for more information.

The *Cantor space* is $\mathbf{C} = \{0,1\}^\infty$. Each language $A \subseteq \{0,1\}^*$ is identified with its characteristic sequence $\chi_A \in \mathbf{C}$ according to the standard (lexicographic) enumeration of $\{0,1\}^*$. We typically write A in place of χ_A. In this way a complexity class $\mathcal{C} \subseteq \mathcal{P}(\{0,1\}^*)$ is viewed as a subset $\mathcal{C} \subseteq \mathbf{C}$. We use the notation $S \upharpoonright n$ to denote the first n bits of a sequence $S \in \mathbf{C}$.

Let $s > 0$ be a real number. An *s-gale* is a function $d : \{0,1\}^* \to [0, \infty)$ such that for all $w \in \{0,1\}^*$,

$$d(w) = \frac{d(w0) + d(w1)}{2^s}.$$

A *martingale* is a 1-gale.

The goal of an s-gale is to obtain large values on sequences as follows.

DEFINITION. *Let d be an s-gale and $S \in \mathbf{C}$.*

1. *d succeeds on S if* $\limsup_{n\to\infty} d(S \upharpoonright n) = \infty$.
2. *d succeeds strongly on S if* $\liminf_{n\to\infty} d(S \upharpoonright n) = \infty$.
3. *The* success set *of d is* $S^\infty[d] = \{S \in \mathbf{C} \mid d \text{ succeeds on } S\}$.
4. *The* strong success set *of d is* $S^\infty_{\mathrm{str}}[d] = \{S \in \mathbf{C} \mid d \text{ succeeds strongly on } S\}$.

Notice that the smaller s is, the more difficult it is for an s-gale to obtain large values. Succeeding martingales ($s = 1$) imply measure 0, and the infimum s for which an s-gale can succeed (or strongly succeed) gives the dimension (or strong dimension) as follows.

DEFINITION. *Let $X \subseteq \mathbf{C}$.*

1. *X has* p-measure 0*, written $\mu_{\mathrm{p}}(X) = 0$, if there is a polynomial-time computable martingale d such that $X \subseteq S^\infty[d]$.*
2. *The* p-dimension *of X, written $\dim_{\mathrm{p}}(X)$, is the infimum of all s such that there exists a polynomial-time computable s-gale d with $X \subseteq S^\infty[d]$.*
3. *The* strong p-dimension *of X, written $\mathrm{Dim}_{\mathrm{p}}(X)$, is the infimum of all s such that there exists a polynomial-time computable s-gale d with $X \subseteq S^\infty_{\mathrm{str}}[d]$.*

We now summarize some of the basic properties of the p-dimensions and p-measure.

PROPOSITION 2.1 (see [19, 4]). *Let* $X, Y \subseteq \mathbf{C}$.

1. $0 \leq \dim_{\mathrm{p}}(X) \leq \mathrm{Dim}_{\mathrm{p}}(X) \leq 1$.
2. *If* $\dim_{\mathrm{p}}(X) < 1$, *then* $\mu_{\mathrm{p}}(X) = 0$.
3. *If* $X \subseteq Y$, *then* $\dim_{\mathrm{p}}(X) \leq \dim_{\mathrm{p}}(Y)$ *and* $\mathrm{Dim}_{\mathrm{p}}(X) \leq \mathrm{Dim}_{\mathrm{p}}(Y)$.

The following theorem indicates that the p-dimensions are useful for studies within the complexity class E.

THEOREM 2.2 (see [17, 19, 4]).

1. $\mu_{\mathrm{p}}(\mathrm{E}) \neq 0$. *In particular,* $\dim_{\mathrm{p}}(\mathrm{E}) = \mathrm{Dim}_{\mathrm{p}}(\mathrm{E}) = 1$.
2. *For all* $c \in \mathbb{N}$, $\mathrm{Dim}_{\mathrm{p}}(\mathrm{DTIME}(2^{cn})) = 0$.

2.3. Online mistake-bound model of learning. A *concept* is a set $C \subseteq U$, where U is some *universe*. A concept C is often identified with its characteristic function $f_C : U \to \{0,1\}$. In this paper the universe is always a set of binary strings. A *concept class* is a set of concepts $\mathcal{C} \subseteq \mathcal{P}(U)$.

Given a concept class $\mathcal{C}$ and a universe U, a learning algorithm tries to learn an unknown target concept $C \in \mathcal{C}$. The algorithm is given a sequence of examples x_1, x_2, ... in U. When given each example x_i, the algorithm must predict if $x_i \in C$ or $x_i \notin C$. The algorithm is then told the correct answer and given the next example. The algorithm makes a *mistake* if its prediction for membership of x_i in C is wrong. This proceeds until every member of U is given as an example.

The goal is to minimize the number of mistakes. The *mistake bound* of a learning algorithm A for a concept class $\mathcal{C}$ is the maximum over all $C \in \mathcal{C}$ of the number of mistakes A makes when learning C, over all possible sequences of examples. The *running time* of A on $\mathcal{C}$ is the maximum time A takes to make a prediction.

2.4. Disjunctions and Winnow. An interesting concept class is the class of monotone disjunctions, which can be efficiently learned by Littlestone's Winnow algorithm [16]. A monotone disjunction on $\{0,1\}^n$ is a formula of the form $\phi_V = \bigvee_{i \in V} x_i$, where $V \subseteq \{1, \ldots, n\}$ and we write a string $x \in \{0,1\}^n$ as $x = x_1 \cdots x_n$. The concept ϕ_V can also be viewed as the set $\{x \in \{0,1\}^n \mid \phi_V(x) = 1\}$ or equivalently as $\{A \subseteq \{1, \ldots, n\} \mid A \cap V \neq \emptyset\}$.

The Winnow algorithm has two parameters α (a weight update multiplier) and θ (a threshold value). Initially, each variable x_i has a weight $w_i = 1$. To classify a string x, the algorithm predicts that x is in the concept if $\sum_i w_i x_i > \theta$, and not in the concept otherwise. The weights are updated as follows whenever a mistake is made.

- If a negative example x is incorrectly classified, then set $w_i := 0$ for all i such that $x_i = 1$. (Certainly these x_i's are not in the disjunction.)
- If a positive example x is incorrectly classified, then set $w_i := \alpha \cdot w_i$ for all i such that $x_i = 1$. (It is considered more likely that these x_i's are in the disjunction.)

A useful setting of the parameters is $\alpha = 2$ and $\theta = n/2$. With these parameters, Littlestone proved that Winnow will make at most $2k \log n + 2$ mistakes when the target disjunction has at most k literals. Also, the algorithm uses $O(n)$ time to classify each example and update the weights.

3. Learning and dimension. In this section we present a framework relating online learning to resource-bounded dimension. This framework is based on reducibility to learnable concept class families.

DEFINITION. *A sequence* $\mathcal{C} = (\mathcal{C}_n \mid n \in \mathbb{N})$ *of concept classes is called a* concept class family.

We consider two types of reducibility: a strong reduction that works for almost all input lengths and a weak reduction that is only required to work infinitely often.

DEFINITION. *Let* $L \subseteq \{0,1\}^*$, $\mathcal{C} = (\mathcal{C}_n \mid n \in \mathbb{N})$ *be a concept class family, and* $r(n)$ *be a time bound.*

1. *We say* L strongly reduces to $\mathcal{C}$ in $r(n)$ time*, and we write* $L \leq^r_{str} \mathcal{C}$*, if there exist a sequence of target concepts* $(c_n \in \mathcal{C}_n \mid n \in \mathbb{N})$ *and a reduction* f *computable in* $O(r(n))$ *time such that for all but finitely many* n*, for all* $x \in \{0,1\}^n$*,* $x \in L$ *if and only if* $f(x) \in c_n$*.*
2. *We say* L weakly reduces to $\mathcal{C}$ in $r(n)$ time*, and we write* $L \leq^r_{wk} \mathcal{C}$*, if there exists a reduction* f *computable in* $O(r(n))$ *time such that for infinitely many* n*, there is a concept* $c_n \in \mathcal{C}_n$ *such that for all* $x \in \{0,1\}^{\leq n}$*,* $x \in L$ *if and only if* $f(0^n, x) \in c_n$*.*

It is necessary to quantify both the time complexity and mistake bound for learning a concept class family.

DEFINITION. *Let* $t, m : \mathbb{N} \to \mathbb{N}$ *and let* $\mathcal{C} = (\mathcal{C}_n \mid n \in \mathbb{N})$ *be a concept class family. We say that* $\mathcal{C} \in \mathcal{L}(t,m)$ *if there is an algorithm that learns* $\mathcal{C}_n$ *in* $O(t(n))$ *time with mistake bound* $m(n)$*.*

Combining the two previous definitions we arrive at our central technical concept.

DEFINITION. *Let* $r, t, m : \mathbb{N} \to \mathbb{N}$*.*

1. $\mathcal{RL}_{\mathrm{str}}(r,t,m)$ *is the class of all languages that* $\leq^r_{str}$*-reduce to some concept class family in* $\mathcal{L}(t,m)$*.*
2. $\mathcal{RL}_{\mathrm{wk}}(r,t,m)$ *is the class of all languages that* $\leq^r_{wk}$*-reduce to some concept class family in* $\mathcal{L}(t,m)$*.*

A remark about the parameters in this definition is in order. If $A \in \mathcal{RL}_{\mathrm{str}}(r,t,m)$, then $A \leq^r_{str} \mathcal{C}$ for some concept class family $\mathcal{C} = (\mathcal{C}_n \mid n \in \mathbb{N})$. Then $x \in A_{=n}$ if and only if $f(x) \in c_n$, where $c_n \in \mathcal{C}_n$ is the target concept and f is the reduction. We emphasize that the complexity of learning $\mathcal{C}_n$ is measured in terms of $n = |x|$, and not the size of c_n or $f(x)$. Instead $\mathcal{C}_n$ is learnable in time $O(t(n))$ with mistake bound $m(n)$.

The following theorem is the main technical tool in this paper. Here we consider exponential-time reductions to concept classes that can be learned in exponential-time, but with subexponentially many mistakes.

THEOREM 3.1. *Let* $c \in \mathbb{N}$*.*

1. $\mathcal{RL}_{\mathrm{str}}(2^{cn}, 2^{cn}, o(2^n))$ *has strong* p*-dimension* 0*.*
2. $\mathcal{RL}_{\mathrm{wk}}(2^{cn}, 2^{cn}, o(2^n))$ *has* p*-dimension* 0*.*

Proof. We prove only that $\mathcal{RL}_{\mathrm{wk}}(2^{cn}, 2^{cn}, o(2^n))$ has p-dimension 0. The other part of the theorem is proved similarly. Let $s > 0$ such that 2^s is rational. It suffices to show that the class has p-dimension at most s.

Let $A \in \mathcal{RL}_{\mathrm{wk}}(2^{cn}, 2^{cn}, o(2^n))$. Then there is a concept class family $\mathcal{C} = \{\mathcal{C}_n \mid n \in \mathbb{N}\} \in \mathcal{L}(2^{cn}, o(2^n))$ such that $A \leq^{2^{cn}}_{wk} \mathcal{C}$. Let f be this reduction from A to $\mathcal{C}$. Then for infinitely many n, there is a target concept $c_n \in \mathcal{C}_n$ such that

$$x \in A_{\leq n} \iff f(x) \in c_n.$$

Let J be the set of all n such that this concept exists. Let $\mathcal{A}$ be a 2^{cn}-time learning algorithm for $\mathcal{C}$ with mistake bound $o(2^n)$.

Fix an n and let $N = 2^{n+1} - 2$. We view the reduction f as generating a sequence

of examples

$$f(s_0), f(s_1), \ldots, f(s_N),$$

one for each string in $\{0,1\}^{\leq n}$. The idea is to run the algorithm $\mathcal{A}$ on this sequence of examples, trying to learn c_n. We will use $\mathcal{A}$'s predictions to define an s-gale d_n inductively as follows.

1. Let $N_0 = 2^{n/2}$. For all strings w with $|w| < N_0$, $d_n(w) = 2^{(s-1)|w|}$.
2. Let ϵ be a small rational number to be determined later. Let w be any string with $N_0 \leq |w| < N$. Run $\mathcal{A}$ on the sequence of examples $f(s_{N_0}), \ldots, f(s_{|w|})$, telling $\mathcal{A}$ that for each i, $N_0 \leq i < |w|$,
   - if $w[i] = 1$, then $f(s_i)$ is a positive example.
   - if $w[i] = 0$, then $f(s_i)$ is a negative example.

   At the end $\mathcal{A}$ will output a prediction for $f(s_{|w|})$:
   - If $\mathcal{A}$ predicts that $f(s_{|w|})$ is a member of the target concept c_n, then we let
     - $d_n(w1) = 2^s(1-\epsilon)d(w)$,
     - $d_n(w0) = 2^s\epsilon d(w)$.
   - Otherwise, $\mathcal{A}$ predicts that $f(s_{|w|})$ is not a member of the target concept c_n, and we let
     - $d_n(w0) = 2^s(1-\epsilon)d(w)$,
     - $d_n(w1) = 2^s\epsilon d(w)$.
3. For w with $|w| \geq N$, we set $d_n(w0) = d_n(w1) = 2^{(s-1)}d_n(w)$.

The reason for making d_n wait until N_0 to bet is computational efficiency. For $|w| < N_0$, $d_n(w)$ is computable in $O(|w|)$ time. If $|w| \geq N_0$, then to compute $d_n(w)$ we need to execute $\mathcal{A}$ on at most $|w|$ examples, each execution taking $O(2^{cn})$ time to compute the example and $O(2^{cn})$ to compute the label, for a total time of $O(|w|2^{cn})$. Because $|w| \geq 2^{n/2}$, this simplifies to $O(|w|^{2c+1})$.

Each time $\mathcal{A}$ makes a correct prediction, the value of the s-gale is increased by a $2^s(1-\epsilon)$ factor. When $\mathcal{A}$ makes a mistake, the value is multiplied by $2^s\epsilon$. Let w_n be the length N prefix of A's characteristic sequence and suppose that $n \in J$. In the computation of $d_n(w_n)$, observe that $\mathcal{A}$ is told the correct labels for the examples according to the target concept c_n. Let m_n be the number of mistakes that $\mathcal{A}$ makes on this sequence of examples when learning c_n; we know that $m_n = o(2^n)$. Then

$$\begin{aligned} d_n(w_n) &= 2^{s(N-N_0)} \cdot (1-\epsilon)^{N-N_0-m_n} \cdot \epsilon^{m_n} \cdot 2^{(s-1)N_0} \\ &= 2^{sN+[(N-N_0-m_n)\log(1-\epsilon)]+[m_n \log \epsilon]-N_0} \\ &\geq 2^{sN-\left(N \log \frac{1}{1-\epsilon}+m_n \log \frac{1-\epsilon}{\epsilon}\right)-N_0}. \end{aligned}$$

We choose $\epsilon \in \mathbb{Q}$ so that $\log \frac{1}{1-\epsilon} < s$ and let $0 < \delta < s - \log \frac{1}{1-\epsilon}$. Then since m_n and N_0 are both $o(N)$, when $n \in J$ is large enough we have

$$d_n(w_n) \geq 2^{\delta N}.$$

Let d be the s-gale $d = \sum_{n=1}^{\infty} 2^{-n} d_n$. Then $A \in S^\infty[d]$. A standard technique is that taking the first $|w| + r$ terms of the sum, we can approximate $d(w)$ to precision 2^{-r} in time $O((|w|+r) \cdot \max\{|w|+r, |w|^{2c+1}\})$. Such an s-gale can be defined for every set in $\mathcal{RL}_{\mathrm{wk}}(2^{cn}, 2^{cn}, o(2^n))$. These gales are all computable within the same time bound, so we can apply a union lemma [19] to conclude that $\mathcal{RL}_{\mathrm{wk}}(2^{cn}, 2^{cn}, o(2^n))$ has p-dimension at most s. □

4. Disjunctive and conjunctive reductions. In this section, as a warmup to our main theorem, we present two basic applications of Theorem 3.1. First, we consider disjunctive reductions.

THEOREM 4.1. $\mathrm{P_d}(\mathrm{DENSE}^c)$ *has* p-*dimension* 0.

Proof. We will show that $\mathrm{P_d}(\mathrm{DENSE}^c) \subseteq \mathcal{RL}_{\mathrm{wk}}(2^{2n}, 2^{2n}, o(2^n))$. For this, let $A \in \mathrm{P_d}(\mathrm{DENSE}^c)$ be arbitrary. Then there is a set $S \in \mathrm{DENSE}^c$ and a reduction $f : \{0,1\}^* \to \mathcal{P}(\{0,1\}^*)$ computable in polynomial time $p(n)$ such that for all $x \in \{0,1\}^*$, $x \in A$ if and only if $f(x) \cap S \neq \emptyset$. Note that on an input of length n, all queries of f have length bounded by $p(n)$. Also, since S is nondense, for any $\epsilon > 0$ there are infinitely many n such that

$$|S_{\leq p(n)}| \leq 2^{n^\epsilon}. \tag{4.1}$$

Let $Q_n = \bigcup_{|x|\leq n} f(x)$ be the set of all queries made by f up through length n. Then $|Q_n| \leq 2^{n+1}p(n)$. Enumerate Q_n as $q_1, \ldots, q_N$. Then each subset of $R \subseteq Q_n$ can be identified with its characteristic string $\chi_R \in \{0,1\}^N$ according to this enumeration. We define $\mathcal{C}_n$ to be the concept class of all monotone disjunctions on $\{0,1\}^N$ that have at most 2^{n^ϵ} literals. Our target disjunction is

$$\phi_n = \bigvee_{i:q_i \in S} q_i,$$

which is a member of $\mathcal{C}_n$ whenever (4.1) holds. For any $x \in \{0,1\}^{\leq n}$,

$$x \in A \iff \phi_n(\chi_{f(x)}) = 1.$$

Given x, $\chi_{f(x)}$ can be computed in $O(2^{2n})$ time. Therefore $A \leq_{wk}^{O(2^{2n})} \mathcal{C} = (\mathcal{C}_n \mid n \in \mathbb{N})$. Since Winnow [16] learns $\mathcal{C}_n$ making at most $2 \cdot 2^{n^\epsilon} \log |Q_n| + 2 = o(2^n)$ mistakes in time $O(2^{2n})$, it follows that $A \in \mathcal{RL}_{\mathrm{wk}}(2^{2n}, 2^{2n}, o(2^n))$. □

Next, we consider conjunctive reductions.

THEOREM 4.2. $\mathrm{P_c}(\mathrm{DENSE}^c)$ *has* p-*dimension* 0.

Proof. We will show that $\mathrm{P_c}(\mathrm{DENSE}^c) \subseteq \mathcal{RL}_{\mathrm{wk}}(2^n, 2^{2n}, o(2^n))$. For this, let $A \leq_{\mathrm{c}}^{\mathrm{p}} S \in \mathrm{DENSE}^c$. Then there is a reduction $f : \{0,1\}^* \to \mathcal{P}(\{0,1\}^*)$ computable in polynomial time $p(n)$ such that for all $x \in \{0,1\}^*$, $x \in A$ if and only if $f(x) \subseteq S$.

Fix an input length n, and let $Q_n = \bigcup_{|x|\leq n} f(x)$. Let $\epsilon > 0$ and consider the concept class

$$\mathcal{C}_n = \{\mathcal{P}(X) \mid X \subseteq Q_n \text{ and } |X| \leq 2^{n^\epsilon}\}.$$

Our target concept is

$$C_n = \mathcal{P}(S \cap Q_n).$$

For infinitely many n, $|S \cap Q_n| \leq |S_{\leq p(n)}| \leq 2^{n^\epsilon}$, in which case $C_n \in \mathcal{C}_n$. For any $x \in \{0,1\}^{\leq n}$, we have

$$x \in A \iff f(x) \in C_n.$$

Therefore $A \leq_{wk}^{p(n)} \mathcal{C} = (\mathcal{C}_n \mid n \in \mathbb{N})$.

The class $\mathcal{C}_n$ can be learned by a simple algorithm that makes at most $|X|$ mistakes when learning $\mathcal{P}(X)$. The hypothesis for X is simply the union of all positive examples

seen so far. More explicitly, the algorithm begins with the hypothesis $H = \emptyset$. In any stage, given an example Q, the algorithm predicts "yes" if $Q \subseteq H$ and "no" otherwise. If the prediction is no, but Q is revealed to be a positive example, then the hypothesis is updated as $H := H \cup Q$. The algorithm will never make a mistake on a negative example, and can make at most $|X|$ mistakes on positive examples.

This algorithm shows that $\mathcal{C} \in \mathcal{L}(2^{2n}, o(2^n))$, so $A \in \mathcal{RL}_{\mathrm{wk}}(p(n), 2^{2n}, o(2^n))$. It follows that $\mathrm{P_c}(\mathrm{DENSE}^c) \subseteq \mathcal{RL}_{\mathrm{wk}}(2^n, 2^{2n}, o(2^n))$. □

Since $\dim_{\mathrm{p}}(\mathrm{E}) = 1$, we have new proofs of the following results of Watanabe.

COROLLARY 4.3 (Watanabe [27]). $\mathrm{E} \not\subseteq \mathrm{P_d}(\mathrm{DENSE}^c)$ *and* $\mathrm{E} \not\subseteq \mathrm{P_c}(\mathrm{DENSE}^c)$. *That is, every* $\leq^{\mathrm{p}}_{\mathrm{d}}$*-hard or* $\leq^{\mathrm{p}}_{\mathrm{c}}$*-hard set for* E *is dense.*

5. Adaptive reductions. In this section we prove our main theorem, which concerns adaptive reductions that make a sublinear number of queries to a nondense set. It turns out that this problem can also be reduced to learning disjunctions.

In a surprising result (refuting a conjecture of Ko [14]), Allender et al. [1] showed that $\mathrm{P_{btt}}(\mathrm{SPARSE}) \subseteq \mathrm{P_d}(\mathrm{SPARSE})$. The disjunctive reduction they obtain will not be polynomial-time computable if the original reduction has more than a constant number of queries. However, in the proof of the following theorem we are still able to exploit their technique, and obtain an exponential-time reduction to a disjunction. Then we can apply the Winnow algorithm as in the previous section.

THEOREM 5.1. *For all* $\alpha < 1$, $\mathrm{P}_{n^\alpha - \mathrm{T}}(\mathrm{DENSE}^c)$ *has* p*-dimension* 0.

Proof. Let $L \leq^{\mathrm{p}}_{n^\alpha - \mathrm{T}} S \in \mathrm{DENSE}^c$ via some oracle machine M. We will show how to reduce L to a class of disjunctions and obtain $\mathrm{P}_{n^\alpha - \mathrm{T}}(\mathrm{DENSE}^c) \subseteq \mathcal{RL}_{\mathrm{wk}}(2^{2n}, 2^{2n}, o(2^n))$.

Fix an input length n. For an input $x \in \{0,1\}^{\leq n}$, consider using each $z \in \{0,1\}^{n^\alpha}$ as the sequence of yes/no answers to M's queries. Each z causes M to produce a sequence of queries $w_0^{x,z}, \ldots, w_{k(x,z)}^{x,z}$, where $k(x,z) < n^\alpha$, and an accepting or rejecting decision. Let $Z_x \subseteq \{0,1\}^{n^\alpha}$ be the set of all query answer sequences that cause M to accept x. Then we have $x \in L$ if and only if

$$(\exists z \in Z_x)(\forall 0 \leq j \leq k(x,z))\ S[w_j^{x,z}] = z[j],$$

which is equivalent to

$$(\exists z \in Z_x)(\forall 0 \leq j \leq k(x,z))\ z[j] \cdot w_j^{x,z} \in S^c \oplus S,$$

where $S^c \oplus S$ is the disjoint union $\{0x \mid x \in S^c\} \cup \{1x \mid x \in S\}$.

A key part of the proof that $\mathrm{P_{btt}}(\mathrm{SPARSE}) \subseteq \mathrm{P_d}(\mathrm{SPARSE})$ in [1] is to show that $\mathrm{P_{1-tt}}(\mathrm{SPARSE})$ is contained in $\mathrm{P_d}(\mathrm{SPARSE})$. The same argument yields that

$$\mathrm{P_{1-tt}}(\mathrm{DENSE}^c) \subseteq \mathrm{P_d}(\mathrm{DENSE}^c).$$

Therefore, there is a set $U \in \mathrm{DENSE}^c$ such that $S^c \oplus S \leq^{\mathrm{p}}_{\mathrm{d}} U$. Letting g be this polynomial-time disjunctive reduction, we have $x \in L$ if and only if

$$(\exists z \in Z_x)(\forall 0 \leq j \leq k(x,z))\ g(z[j] \cdot w_j^{x,z}) \cap U \neq \emptyset.$$

For each $z \in Z_x$, let

$$H_{x,z} = \{\langle u_0, \ldots, u_{k(x,z)} \rangle \mid (\forall 0 \leq j \leq k(x,z))\ u_j \in g(z[j] \cdot w_j^{x,z})\}.$$

Let $r(n)$ be a polynomial bounding the running time of g on inputs of the form $z[j] \cdot w_j^{x,z}$, where $|x| \leq n$. Define

$$A_n = \{\langle u_0, \ldots, u_k \rangle \mid k < n^\alpha \text{ and } (\forall 0 \leq j \leq k)\ u_j \in U_{\leq r(n)}\}.$$

Then we have $x \in L$ if and only if

$$(\exists z \in Z_x)(\exists v \in H_{x,z})\ v \in A_n.$$

Letting

$$H_x = \bigcup_{z \in Z_x} H_{x,z},$$

we can rewrite this as

$$x \in L \iff H_x \cap A_n \neq \emptyset. \tag{5.1}$$

Because $|H_{x,z}| \leq r(n)^{n^\alpha}$ we have

$$|H_x| \leq |Z_x| \cdot r(n)^{n^\alpha} \leq 2^{n^\alpha \cdot (1+\log r(n))}. \tag{5.2}$$

Also,

$$|A_n| \leq n^\alpha \cdot |U_{\leq r(n)}|^{n^\alpha}.$$

Let $\epsilon \in (0, 1-\alpha)$, and let $\delta \in (\alpha+\epsilon, 1)$. Then since U is nondense, for infinitely many n, we have $|U_{\leq r(n)}| \leq 2^{n^\epsilon}$. This implies

$$(\exists^\infty n)\ |A_n| \leq n^\alpha \cdot 2^{n^{\alpha+\epsilon}} \leq 2^{n^\delta}. \tag{5.3}$$

Let

$$H_n = \bigcup_{x \in \{0,1\}^{\leq n}} H_x.$$

Then from (5.2), $|H_n| \leq 2^{2n}$ if n is sufficiently large.

Enumerate H_n as $h_1, \ldots, h_N$. We identify any $R \subseteq H_n$ with its characteristic string $\chi_R^{(n)} \in \{0,1\}^N$ according to this enumeration. Let $\mathcal{C}_n$ be the concept class of all monotone disjunctions on $\{0,1\}^N$ that have at most 2^{n^δ} literals.

Define the disjunction

$$\phi_n = \bigvee_{i : h_i \in A_n} h_i,$$

which by (5.3) is in $\mathcal{C}_n$ for infinitely many n. For any $x \in \{0,1\}^{\leq n}$, from (5.1) it follows that

$$x \in L \iff \phi_n(\chi_{H_x}^{(n)}) = 1.$$

Given $x \in \{0,1\}^{\leq n}$, we can compute $\chi_{H_x}^{(n)}$ in $O(2^n \cdot \mathrm{poly}(n) + |H_n|)$ time. Therefore, letting $\mathcal{C} = (\mathcal{C}_n \mid n \in \mathbb{N})$, we have $L \leq_{wk}^{2^{2n}} \mathcal{C}$. Since $\mathcal{C}_n$ is learnable by Winnow with at most $2 \cdot 2^{n^\delta} \cdot \log |H_n| + 2 = o(2^n)$ mistakes, it follows that $L \in \mathcal{RL}_{\mathrm{wk}}(2^{2n}, 2^{2n}, o(2^n))$. □

As a corollary, we have a positive answer to the question of Lutz and Mayordomo [22] mentioned in our introduction.

COROLLARY 5.2. *For all* $\alpha < 1$, $\mathrm{P}_{n^\alpha\text{-T}}(\mathrm{DENSE}^c)$ *has* p*-measure* 0.

COROLLARY 5.3. *For all* $\alpha < 1$, $\mathrm{E} \not\subseteq \mathrm{P}_{n^\alpha-\mathrm{T}}(\mathrm{DENSE}^c)$. *That is, every* $\leq^{\mathrm{p}}_{n^\alpha-\mathrm{T}}$*-hard set for* E *is dense.*

If we scale down from nondense sets to sparse sets, the same proof technique can handle more queries.

THEOREM 5.4. $\mathrm{P}_{o(n/\log n)-\mathrm{T}}(\mathrm{SPARSE})$ *has strong* p*-dimension* 0.

Proof. Let $L \leq^{\mathrm{p}}_{f(n)-\mathrm{T}} S \in \mathrm{SPARSE}$ via some oracle machine M, where $f(n) = o(n/\log n)$.

Fix an input length n. For an input $x \in \{0,1\}^n$, each query answer sequence $z \in \{0,1\}^{f(n)}$ causes M to produce a sequence of queries $w_0^{x,z}, \ldots, w_{k(x,z)}^{x,z}$, where $k(x,z) < f(n)$, and an accepting or rejecting decision. Let $Z_x \subseteq \{0,1\}^{f(n)}$ be the set of all query answer sequences that cause M to accept x. Then we have $x \in L$ if and only if

$$(\exists z \in Z_x)(\forall 0 \leq j \leq k(x,z))\ z[j] \cdot w_j^{x,z} \in S^c \oplus S.$$

Since $\mathrm{P}_{1-\mathrm{tt}}(\mathrm{SPARSE}) \subseteq \mathrm{P}_{\mathrm{d}}(\mathrm{SPARSE})$, there is a set $U \in \mathrm{SPARSE}$ such that $S^c \oplus S \leq^{\mathrm{p}}_{\mathrm{d}} U$. Letting g be this polynomial-time disjunctive reduction, we have $x \in L$ if and only if

$$(\exists z \in Z_x)(\forall 0 \leq j \leq k(x,z))\ g(z[j] \cdot w_j^{x,z}) \cap U \neq \emptyset.$$

As before, we can define sets H_x and A_n so that

$$x \in L \iff H_x \cap A_n \neq \emptyset.$$

Let $r(n)$ be a polynomial bounding the running time of g outputs on inputs of the form $z[j] \cdot w_j^{x,z}$, where $|x| = n$. Then

$$|H_x| \leq |Z_x| \cdot r(n)^{f(n)} \leq 2^{f(n)\cdot(1+\log r(n))},$$

so we have $|H_x| \leq 2^n$ if n is sufficiently large because $f(n) = o(n/\log n)$. Letting $H_n = \bigcup_{x\in\{0,1\}^n} H_x$, we have $|H_n| \leq 2^{2n}$.

Also,

$$|A_n| \leq f(n) \cdot |U_{\leq r(n)}|^{f(n)}.$$

Let $q(n)$ be a polynomial such that $|U_{\leq r(n)}| \leq q(n)$ for all n. Then

$$|A_n| \leq f(n) \cdot q(n)^{f(n)} \leq 2^{f(n)\log q(n) + \log f(n)}.$$

Let $v(n) = f(n)\log q(n) + \log f(n)$. Notice that $v(n) = o(n)$ because $f(n) = o(n/\log n)$.

As before, we enumerate H_n as $h_1, \ldots, h_N$ and identify any $R \subseteq H_n$ with its characteristic string $\chi_R^{(n)} \in \{0,1\}^N$. Let $\mathcal{C}_n$ be the concept class of all monotone disjunctions on $\{0,1\}^N$ that have at most $2^{v(n)}$ literals. The disjunction $\phi_n = \bigvee_{i:h_i\in A_n} h_i$ is in $\mathcal{C}_n$ for every n. For any $x \in \{0,1\}^n$, we have

$$x \in L \iff \phi_n(\chi_{H_x}^{(n)}) = 1.$$

Given $x \in \{0,1\}^n$, we can compute $\chi_{H_x}^{(n)}$ in $O(2^n \cdot \mathrm{poly}(n) + |H_n|)$ time. Therefore, letting $\mathcal{C} = (\mathcal{C}_n \mid n \in \mathbb{N})$, we have $L \leq^{2^{2n}}_{str} \mathcal{C}$. Since $\mathcal{C}_n$ is learnable by

Winnow with at most $2 \cdot 2^{v(n)} \cdot \log |H_n| + 2 = o(2^n)$ mistakes, it follows that $L \in \mathcal{RL}_{\mathrm{str}}(2^{2n}, 2^{2n}, o(2^n))$. □

The following corollary improves the result of Fu [8] that $\mathrm{E} \not\subseteq \mathrm{P}_{o(n/\log n)-\mathrm{T}}(\mathrm{TALLY})$.

COROLLARY 5.5. $\mathrm{E} \not\subseteq \mathrm{P}_{o(n/\log n)-\mathrm{T}}(\mathrm{SPARSE})$.

Since Wilson constructed an oracle relative to which $\mathrm{E} \subseteq \mathrm{P}_{O(n)-\mathrm{tt}}(\mathrm{SPARSE})$ [28, 21], Corollary 5.5 is near the limits of relativizable techniques.

In Theorem 5.4, we used strong dimension, which raises a technical point. The results about reductions to DENSE^c cannot be strengthened to strong p-dimension simply because the class DENSE^c itself has strong dimension 1. This is because being nondense is an infinitely often property [9]. However, if we replace DENSE^c with by SPARSE in any of our results, the proofs can be adapted to show that the resulting class has strong p-dimension 0. We can also obtain strong dimension results by substituting the larger class $\mathrm{DENSE}^c_{\mathrm{i.o.}}$, where $\mathrm{DENSE}_{\mathrm{i.o.}}$ is the class of all L that satisfy $(\exists \epsilon > 0)(\exists^\infty n)\ |L_{\leq n}| > 2^{n^\epsilon}$.

6. Hard sets for NP. The hypothesis "NP has positive p-dimension," written $\dim_{\mathrm{p}}(\mathrm{NP}) > 0$, was first used in [11] to study the inapproximability of MAX3SAT. This positive dimension hypothesis is apparently much weaker than Lutz's often-investigated $\mu_{\mathrm{p}}(\mathrm{NP}) \neq 0$ hypothesis, but is a stronger assumption than $\mathrm{P} \neq \mathrm{NP}$:

$$\mu_{\mathrm{p}}(\mathrm{NP}) \neq 0 \Rightarrow \dim_{\mathrm{p}}(\mathrm{NP}) = 1 \Rightarrow \dim_{\mathrm{p}}(\mathrm{NP}) > 0 \Rightarrow \mathrm{P} \neq \mathrm{NP}.$$

The measure hypothesis $\mu_{\mathrm{p}}(\mathrm{NP}) \neq 0$ has many plausible consequences that are not known to follow from $\mathrm{P} \neq \mathrm{NP}$ (see, e.g., [22]). So far, few consequences of $\dim_{\mathrm{p}}(\mathrm{NP}) > 0$ are known. The following corollary of our results begins to remedy this.

THEOREM 6.1. *If* $\dim_{\mathrm{p}}(\mathrm{NP}) > 0$*, then every set that is hard for* NP *under* $\leq^{\mathrm{p}}_{\mathrm{d}}$ *reductions,* $\leq^{\mathrm{p}}_{\mathrm{c}}$ *reductions, or* $\leq^{\mathrm{p}}_{n^\alpha-\mathrm{T}}$ *reductions* ($\alpha < 1$) *is dense, and every set that is hard under* $\leq^{\mathrm{p}}_{o(n/\log n)-\mathrm{T}}$ *reductions is not sparse.*

The consequences in Theorem 6.1 are much stronger than what is known to follow from $\mathrm{P} \neq \mathrm{NP}$. If $\mathrm{P} \neq \mathrm{NP}$, then no $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard or $\leq^{\mathrm{p}}_{\mathrm{c}}$-hard set is sparse [26, 3], but it is not known whether hard sets under disjunctive reductions or unbounded Turing reductions can be sparse.

Another result is that if $\mathrm{NP} \neq \mathrm{RP}$, then no $\leq^{\mathrm{p}}_{\mathrm{d}}$-hard set for NP is sparse [7, 6]. It is interesting to see that while the hypotheses $\dim_{\mathrm{p}}(\mathrm{NP}) > 0$ and $\mathrm{NP} \neq \mathrm{RP}$ are apparently incomparable, they both have implications for the density of the disjunctively hard sets for NP.

7. Conclusion. Our connection between online learning and resource-bounded dimension appears to be a powerful tool for computational complexity. We have used it to give relatively simple proofs and improvements of several previous results.

An interesting observation is that for all reductions $\leq^{\mathrm{p}}_\tau$ for which we know how to prove "every $\leq^{\mathrm{p}}_\tau$-hard set for E is dense," by the results presented here we can actually prove "$\mathrm{P}_\tau(\mathrm{DENSE}^c)$ has p-dimension 0." Indeed, we have proven the strongest results for Turing reductions in this way.

Acknowledgment. I thank the anonymous referees for helpful comments and corrections.

REFERENCES

[1] E. ALLENDER, L. A. HEMACHANDRA, M. OGIWARA, AND O. WATANABE, *Relating equivalence and reducibility to sparse sets*, SIAM J. Comput., 21 (1992), pp. 521–539.

[2] D. Angluin, *Queries and concept learning*, Machine Learning, 2 (1988), pp. 319–342.
[3] V. Arvind, Y. Han, L. Hemachandra, J. Köbler, A. Lozano, M. Mundhenk, A. Ogiwara, U. Schöning, R. Silvestri, and T. Thierauf, *Reductions to sets of low information content*, in Complexity Theory: Current Research, K. Ambos-Spies, S. Homer, and U. Schöning, eds., Cambridge University Press, Cambridge, UK, 1993, pp. 1–45.
[4] K. B. Athreya, J. M. Hitchcock, J. H. Lutz, and E. Mayordomo, *Effective strong dimension in algorithmic information and computational complexity*, in Proceedings of the 21st Annual Symposium on Theoretical Aspects of Computer Science, Springer, Berlin, 2004, pp. 632–643.
[5] L. Berman and J. Hartmanis, *On isomorphism and density of NP and other complete sets*, SIAM J. Comput., 6 (1977), pp. 305–322.
[6] H. Buhrman, L. Fortnow, and L. Torenvliet, *Six hypotheses in search of a theorem*, in Proceedings of the 12th Annual IEEE Conference on Computational Complexity, IEEE Computer Society, Piscataway, NJ, 1997, pp. 2–12.
[7] J. Cai, A. V. Naik, and D. Sivakumar, *On the existence of hard sparse sets under weak reductions*, in Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science, Springer, Berlin, 1996, pp. 307–318.
[8] B. Fu, *With quasilinear queries EXP is not polynomial time Turing reducible to sparse sets*, SIAM J. Comput., 24 (1995), pp. 1082–1090.
[9] X. Gu, *A note on dimensions of polynomial size circuits*, Theoret. Comput. Sci., 359 (2006), pp. 176–187.
[10] L. A. Hemachandra, M. Ogiwara, and O. Watanabe, *How hard are sparse sets?*, in Proceedings of the Seventh Annual Structure in Complexity Theory Conference, IEEE Computer Society Press, Piscataway, NJ, 1992, pp. 222–238.
[11] J. M. Hitchcock, *MAX3SAT is exponentially hard to approximate if NP has positive dimension*, Theoret. Comput. Sci., 289 (2002), pp. 861–869.
[12] J. M. Hitchcock, J. H. Lutz, and E. Mayordomo, *Scaled dimension and nonuniform complexity*, J. Comput. System Sci., 69 (2004), pp. 97–122.
[13] J. M. Hitchcock, J. H. Lutz, and E. Mayordomo, *The fractal geometry of complexity classes*, SIGACT News, 36 (2005), pp. 24–38.
[14] K. Ko, *Distinguishing conjunctive and disjunctive reducibilities by sparse sets*, Inform. Comput., 81 (1989), pp. 62–87.
[15] W. Lindner, R. Schuler, and O. Watanabe, *Resource-bounded measure and learnability*, Theory Comput. Syst., 33 (2000), pp. 151–170.
[16] N. Littlestone, *Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm*, Machine Learning, 2 (1987), pp. 285–318.
[17] J. H. Lutz, *Almost everywhere high nonuniform complexity*, J. Comput. System Sci., 44 (1992), pp. 220–258.
[18] J. H. Lutz, *The quantitative structure of exponential time*, in Complexity Theory Retrospective II, L. A. Hemaspaandra and A. L. Selman, eds., Springer, New York, 1997, pp. 225–254.
[19] J. H. Lutz, *Dimension in complexity classes*, SIAM J. Comput., 32 (2003), pp. 1236–1259.
[20] J. H. Lutz, *Effective fractal dimensions*, Math. Logic Quart., 51 (2005), pp. 62–72.
[21] J. H. Lutz and E. Mayordomo, *Measure, stochasticity, and the density of hard languages*, SIAM J. Comput., 23 (1994), pp. 762–779.
[22] J. H. Lutz and E. Mayordomo, *Twelve problems in resource-bounded measure*, Bull. Eur. Assoc. Theoret. Comput. Sci. EATCS, 68 (1999), pp. 64–80. Also in *Current Trends in Theoretical Computer Science: Entering the 21st Century*, World Scientific Publishing, River Edge, NJ, 2001, pp. 83–101.
[23] J. H. Lutz and Y. Zhao, *The density of weakly complete problems under adaptive reductions*, SIAM J. Comput., 30 (2000), pp. 1197–1210.
[24] S. R. Mahaney, *Sparse complete sets for NP: Solution of a conjecture of Berman and Hartmanis*, J. Comput. System Sci., 25 (1982), pp. 130–143.
[25] E. Mayordomo. *Personal communication*, 2002.
[26] M. Ogiwara and O. Watanabe, *On polynomial-time bounded truth-table reducibility of NP sets to sparse sets*, SIAM J. Comput., 20 (1991), pp. 471–483.
[27] O. Watanabe, *Polynomial time reducibility to a set of small density*, in Proceedings of the Second Structure in Complexity Theory Conference, IEEE Computer Society, Piscataway, NJ, 1987, pp. 138–146.
[28] C. B. Wilson, *Relativized circuit complexity*, J. Comput. System Sci., 31 (1985), pp. 169–181.

SIAM J. COMPUT.
Vol. 36, No. 6, pp. 1709–1728

ONLINE SCHEDULING OF EQUAL-LENGTH JOBS: RANDOMIZATION AND RESTARTS HELP*

MAREK CHROBAK†, WOJCIECH JAWOR†, JIŘÍ SGALL‡, AND TOMÁŠ TICHÝ‡

Abstract. We consider the following scheduling problem. The input is a set of jobs with equal processing times, where each job is specified by its release time and deadline. The goal is to determine a single-processor nonpreemptive schedule that maximizes the number of completed jobs. In the online version, each job arrives at its release time. We give two online algorithms with competitive ratios below 2 and show several lower bounds on the competitive ratios. First, we give a barely random 5/3-competitive algorithm that uses only one random bit. We also show a lower bound of 3/2 on the competitive ratio of barely random algorithms that randomly choose one of two deterministic algorithms. If the two algorithms are selected with equal probability, we can further improve the bound to 8/5. Second, we give a deterministic 3/2-competitive algorithm in the model that allows restarts, and we show that in this model the ratio 3/2 is optimal. For randomized algorithms with restarts we show a lower bound of 6/5.

Key words. job scheduling, online algorithms, competitive analysis, randomization

AMS subject classifications. 68W10, 68W20, 68W25, 68W40, 90B99

DOI. 10.1137/S0097539704446608

1. Introduction.

We consider the following fundamental problem in the area of real-time scheduling. The input is a collection of jobs with equal processing times p, where each job j is specified by its release time r_j and deadline d_j. (All numbers are assumed to be positive integers.) The desired output is a single-processor nonpreemptive schedule. Naturally, each scheduled job must be executed between its release time and deadline, and different jobs cannot overlap. The term "nonpreemptive" means that each job must be executed without interruptions, in a contiguous interval of length p. The objective is to maximize the number of completed jobs.

In the *online version*, each job j arrives at time r_j, and its deadline d_j is revealed at this time. The number of jobs and future release times are unknown. At each time step when no job is running, we have to decide whether to start a job and, if so, to choose which one, based only on the information about the jobs released so far. An online algorithm is called *c-competitive* if on every input instance it schedules at least $1/c$ as many jobs as the optimum schedule.

Our results. It is known that a simple greedy algorithm is 2-competitive for this problem and that this ratio is optimal for deterministic algorithms. We present two ways to improve the competitive ratio of 2.

First, addressing an open question in [13, 14], we give a 5/3-competitive randomized algorithm. Interestingly, our algorithm is *barely random*; it chooses with probability 1/2 one of two deterministic algorithms, i.e., it uses only one random bit.

*Received by the editors December 1, 2004; accepted for publication (in revised form) June 12, 2006; published electronically March 19, 2007.
http://www.siam.org/journals/sicomp/36-6/44660.html

†Department of Computer Science, University of California, Riverside, CA 92521 (marek@cs.ucr.edu, wojtek@cs.ucr.edu). The work of these authors was supported by NSF grants CCR-9988360, CCR-0208856, and OISE-0340752.

‡Mathematical Institute, AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic (sgall@math.cas.cz, tichy@math.cas.cz). The work of these authors was partially supported by Institutional Research Plan AV0Z10190503, by Institute for Theoretical Computer Science, Prague (project 1M0545 of MŠMT ČR), grant 201/05/0124 of GA ČR, and grant IAA1019401 of GA AV ČR.

These two algorithms are two *identical* copies of the same deterministic algorithm that are run concurrently and use a shared lock to break the symmetry and coordinate their behaviors. We are not aware of previous work in the design of randomized online algorithms that uses such a mechanism to coordinate identical algorithms; thus, this technique may be of its own independent interest.

We then show a lower bound of 3/2 on the competitive ratio of barely random algorithms that choose one of two deterministic algorithms with any probability. If these algorithms are each chosen with probability 1/2, we improve the lower bound to 8/5.

Second, we give a deterministic 3/2-competitive algorithm in the *preemption–restart* model. In this model, an online algorithm is allowed to abort a job during execution in order to start another job. The algorithm gets credit only for jobs that are executed contiguously from beginning to end. Aborted jobs can be restarted (from scratch) and completed later. Note that the final schedule produced by such an algorithm is *not* preemptive. Thus the distinction between nonpreemptive and preemption–restart models makes sense only in the online case. (The optimal solutions are always the same.) In addition to the algorithm, we give a matching lower bound, by showing that no deterministic online algorithm with restarts can be better than 3/2-competitive. We also show a lower bound of 6/5 for randomized algorithms with restarts.

We remark that both our algorithms are natural and easy to state and implement. The competitive analysis is, however, fairly involved, and it relies on some structural lemmas about schedules of equal-length jobs.

An extended abstract of this paper appeared as [9].

Previous work. The problem of scheduling equal-length jobs to maximize the number of completed jobs has been well studied in the literature. In the offline case, an $O(n \log n)$-time algorithm for the feasibility problem (checking if *all* jobs can be completed) was given by Garey *et al.* [12] (see also [23, 7]). The maximization version can also be solved in polynomial time [8, 2], although the known algorithms are rather slow. (Carlier [7] claimed an $O(n^3 \log n)$ algorithm but, as pointed out in [8], his algorithm is not correct.)

As the first positive result on the online version, Baruah, Haritsa, and Sharma [4, 5] show that a deterministic greedy algorithm is 2-competitive; in fact, they show that any nonpreemptive deterministic algorithm that is never idle at times when jobs are available for execution is also 2-competitive.

Goldman, Parwatikar, and Suri [13] gave a lower bound of 4/3 on the competitive ratio of randomized algorithms and the tight bound of 2 for deterministic algorithms. We briefly sketch these lower bounds, as they illustrate well what situations an online algorithm needs to avoid in order to achieve a small competitive ratio. Let $p \geq 2$. The jobs, written in the form $j = (r_j, d_j)$, are $1 = (0, 2p + 1)$, $2 = (1, p + 1)$, and $3 = (p, 2p)$. The instance consists of jobs 1,2 or jobs 1,3; in both cases the optimum is 2. Figure 1.1 illustrates the input instance and the adversary strategy. (In this figure, and later throughout the paper, the horizontal dimension corresponds to the time axis, each job j in the input instance is drawn as a line segment spanning the interval $[r_j, d_j]$, and jobs that appear in the schedules are represented by rectangles of length p positioned at the actual time of execution.) In the deterministic case, release job 1. If at time 0 the online algorithm starts job 1, then release job 2; otherwise, release job 3. The online algorithm completes only one job and the competitive ratio is no better than 2. In the randomized case, using Yao's principle [24, 6], we choose

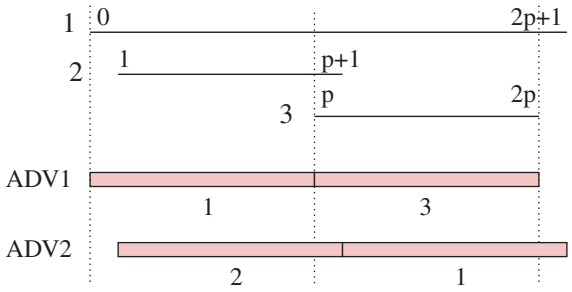


FIG. 1.1. *Jobs used in the lower bound proof.*

each of the two instances with probability 1/2. The expected number of completed jobs of any deterministic online algorithm is at most 1.5, as on one of the instances it completes only one job. Thus the competitive ratio is no better than $2/1.5 = 4/3$.

Goldman, Parwatikar, and Suri [13] show that the lower bound of 2 can be beaten if the jobs on input have sufficiently large "slack": more specifically, they prove that a greedy algorithm is 3/2-competitive for instances where $d_j - r_j \geq 2p$ for all jobs j. This is closely related to our algorithm with restarts: On such instances, our algorithm never uses restarts and becomes identical to the greedy algorithm. Thus in this special case our result constitutes an alternative proof of the result from [13]. Exploring further this direction, Goldwasser [14] obtained a parameterized extension of this result: if $d_j - r_j \geq \lambda p$ for all jobs j, where $\lambda \geq 1$ is an integer, then the competitive ratio is $1 + 1/\lambda$.

In our brief overview of the literature given above, we focused on the case when jobs are of equal length and the objective function is the number of completed jobs. We need to stress though that, in addition to the work cited above, there is a vast literature on real-time scheduling problems where a variety of other models is considered: other or no restrictions can be placed on processing times, jobs may have different weights (benefits), we can have multiple processors, and preemptions may be allowed. For example, once arbitrary processing times and/or weights are introduced, no constant-competitive nonpreemptive algorithms exist. Therefore it is common in the literature to allow preemption with resumption, where a job can be preempted and later started from where it was stopped.

The model with restarts was studied by Hoogeveen, Potts, and Woeginger [17]. They present a 2-competitive deterministic algorithm with restarts for jobs with arbitrary processing times and the objective to maximize the number of completed jobs. They also give a matching lower bound. Their algorithm does not use restarts on the instances with equal processing times, and thus it is no better than 2-competitive for our problem.

Real-time scheduling is an area where randomized algorithms have been found quite effective. Most randomized algorithms in the general scenarios use the classify-and-randomly-select technique by Lipton and Tomkins [20]. Typically, this method decreases the dependence of the competitive ratio from linear to logarithmic in certain parameters (e.g., the maximum ratio between job weights), but it does not apply to the case of jobs with equal lengths and weights. Our randomized algorithm is based on entirely different ideas.

Barely random algorithms have been successfully applied in the past to a variety of online problems, including the list update problem [21], the k-server problem [3], and makespan scheduling [1, 11, 22]. In particular, the algorithm of Albers [1] involves two

deterministic processes in which the second one keeps track of the first and corrects its potential "mistakes"—a coordination idea somewhat similar to ours, although in [1] the two processes are not symmetric. Closer to the topic of this paper, for the general throughput maximization problem with arbitrary processing times and with preemption, Kalyanasundaram and Pruhs [19] showed that a constant competitive ratio can be achieved with a barely random algorithm, even though no constant-competitive deterministic algorithms are possible in that model.

The area of real-time scheduling is of course well motivated by multitudes of applied scenarios. In particular, the model of equal-length jobs—without or with limited preemption—is related to applications in packet switched networks. When different weights are considered, the problem has further connections to the "quality of service" issues (recently a fashionable phrase). Nevertheless, we shamelessly admit that this work has been partially driven by plain curiosity. It is quite intriguing, after all, that so little is known about the competitiveness of such a fundamental scheduling problem.

2. Preliminaries. The input consists of a set of jobs $J = \{1, 2, \ldots\}$, where each job j is given by its release time r_j and deadline d_j. All jobs have processing time p. (We assume that all numbers are positive integers and that $d_j \geq r_j + p$ for all j.) The *expiration time* of a job j is $x_j = d_j - p$, i.e., the last time when it can be started. A job j is called *admissible* at time t if $r_j \leq t \leq x_j$. A job j is called *tight* if $x_j - r_j < p$.

A *nonpreemptive schedule* $\mathcal{A}$ assigns to each completed job j an interval $[S_j^{\mathcal{A}}, C_j^{\mathcal{A}})$, with $r_j \leq S_j^{\mathcal{A}} \leq x_j$ and $C_j^{\mathcal{A}} = S_j^{\mathcal{A}} + p$, during which j is executed. These intervals are disjoint for distinct jobs. $S_j^{\mathcal{A}}$ and $C_j^{\mathcal{A}}$ are called the *start time* and *completion time* of job j. Without loss of generality, both are assumed to be an integer. The number of jobs completed in $\mathcal{A}$ is denoted $|\mathcal{A}|$. We adopt a convention in which "job running (a schedule being idle, etc.) at time t" is an equivalent shorthand for "job running (a schedule being idle, etc.) in the interval $[t, t+1)$." Given a schedule $\mathcal{A}$, a job is *pending* at time t in $\mathcal{A}$ if it is admissible at t (i.e., $r_j \leq t \leq x_j$) but not yet completed in $\mathcal{A}$. Note that according to this definition a job that is being executed at t may also be considered pending. When $\mathcal{A}$ is understood from context, we will typically use notation P_t to denote the set of jobs pending at time t.

For any set of jobs Q, we say that Q is *feasible* at time t if there exists a schedule which completes all jobs in Q such that no job is started before t. Q is *flexible* at time t if it is feasible at time $t + p$.

Applying the Jackson rule [18], it is quite easy to determine whether a set P of pending jobs is feasible at t: Order the jobs in P in order of increasing deadlines, and schedule them at times t, $t + p$, $t + 2p$, etc. Then P is feasible if and only if all jobs in P meet their deadlines. Furthermore, if we want to compute the maximum-size feasible subset $P' \subseteq P$, we can start with $P' = \emptyset$, and then add jobs $j \in P - P'$ to P', one by one and in arbitrary order, as long as P' remains feasible. This means, in particular, that P' is a maximum-size feasible subset of P if and only if P' is a $\subseteq$-maximal feasible subset of P. All those properties can be proven by elementary exchange arguments, and the proofs are left to the reader.

We say that a job started by a schedule $\mathcal{A}$ at time t is *flexible in* $\mathcal{A}$ if the set of all jobs pending in $\mathcal{A}$ at t is flexible; otherwise, the job is called *urgent*. Intuitively, a job is flexible if we could possibly postpone it and stay idle for time p without losing any of the currently pending jobs; this could improve the schedule if a tight job arrives. On the other hand, postponing an urgent job can bring no advantage to the algorithm.

An *online algorithm* constructs a schedule incrementally: at each step t making decisions based only on the jobs released at or before t. The information about each

job j, including its deadline, is revealed to the algorithm at its release time r_j. A *nonpreemptive online algorithm* can start a job only when no job is running; thus, if a job is started at time t, the algorithm has no choice but to let it run to completion at time $t + p$. An *online algorithm with restarts* can start a job at any time. If we start a job j when another job, say k, is running, then k is aborted and started from scratch when (and if) it is started again later. The unfinished portion of k is removed from the final schedule, which is considered to be idle during this time interval. Thus the final schedule generated by an online algorithm with restarts is nonpreemptive.

An online algorithm is called *c-competitive* if, for any set of jobs J and any schedule $\mathcal{Z}$ for J, the schedule $\mathcal{A}$ generated by the algorithm on J satisfies $|\mathcal{Z}| \leq c|\mathcal{A}|$. If the algorithm is randomized, the expression $|\mathcal{A}|$ is replaced by the expected (average) number of jobs completed on the given instance.

The definitions above assume the model—standard in the scheduling literature—with integer release times and deadlines, which implicitly makes the time discrete. Some papers on real-time scheduling work with continuous time. Both our algorithms can be modified to the continuous time model and unit processing time jobs without any change in performance, at the cost of a somewhat more technical presentation.

Properties of schedules. For every instance J, we fix a *canonical linear ordering* $\prec$ of J such that $j \prec k$ implies $d_j \leq d_k$. In other words, we order the jobs by their deadlines, breaking the ties arbitrarily but consistently for all applications of the deadline ordering. The term "earliest-deadline" (ED) now refers to the $\prec$-minimal job.

A schedule $\mathcal{A}$ is called "earliest-deadline-first" (EDF) if, whenever it starts a job, it chooses the ED job of all the pending jobs that are later completed in $\mathcal{A}$. (Note that this may not be the overall ED pending job.)

A schedule $\mathcal{A}$ is *normal* if it satisfies the following two properties:

(n1) when $\mathcal{A}$ starts a job, it chooses the ED job from the set of all pending jobs;

(n2) if the set of all pending jobs in $\mathcal{A}$ at some time t is not flexible, then some job is running at t.

Obviously, any normal schedule is EDF, but the reverse is not true. All algorithms presented in this paper generate normal schedules. The properties (n1) and (n2) are reasonable, as the online algorithm cannot make a mistake by enforcing them. Formally, any online algorithm can be modified, using a standard exchange argument, to produce normal schedules without reducing the number of scheduled jobs. (We omit the proof as we do not need this fact in the paper.)

The following property will be crucial in our proofs.

LEMMA 2.1. *Suppose that a job j is urgent in a normal schedule $\mathcal{A}$. Then at any time t, $S_j^{\mathcal{A}} \leq t \leq x_j$, an urgent job is running in $\mathcal{A}$.*

Proof. Denote by P the set of jobs pending at time $S_j^{\mathcal{A}}$ (including j). By the assumption about j, P is not flexible at $S_j^{\mathcal{A}}$. Towards contradiction, suppose that $\mathcal{A}$ is idle or starts a flexible job at time t, where $C_j^{\mathcal{A}} \leq t \leq x_j$. Then the set Q of jobs pending at time t is flexible at t. Since j is the ED job from P (by the normality of $\mathcal{A}$) and $t \leq x_j$, all other jobs in P have not expired until t, and thus Q contains all the jobs from P that are not completed in $\mathcal{A}$ until time t.

Using the above properties, we can rearrange the schedule as follows. Since Q is flexible at t, we can schedule all jobs of Q at time $t + p$ or later, start j at t, and schedule all jobs in $P - Q - \{j\}$ as in $\mathcal{A}$. But this shows that P is flexible at time $S_j^{\mathcal{A}}$—a contradiction. □

Two schedules $\mathcal{D}$ and $\mathcal{D}'$ for an instance J are called *equivalent* if they satisfy the following conditions for each time t:

(eq1) $\mathcal{D}$ starts a job at t if and only if $\mathcal{D}'$ starts a job at t.

(eq2) Suppose that $\mathcal{D}$ starts a job j at time t and $\mathcal{D}'$ starts a job j' at time t. Then j is flexible in $\mathcal{D}$ if and only if j' is flexible in $\mathcal{D}'$. Furthermore, if they are both flexible then $j = j'$.

Obviously, if $\mathcal{D}$, $\mathcal{D}'$ are equivalent, then $|\mathcal{D}| = |\mathcal{D}'|$.

To facilitate competitive analysis, we modify normal schedules into equivalent EDF schedules with better structural properties. In particular, the next lemma gives us more control over the choice of jobs that we can include in the schedule, in situations where there are several choices. The modified schedules are no longer normal (only EDF, which is a weaker requirement); nevertheless, as they are equivalent to normal schedules, they inherit some of their important properties, including Lemma 2.1.

LEMMA 2.2. *Let $\mathcal{X}$ be a normal schedule for a set of jobs J. Let $f : J \to J$ be a partial function such that if $f(k)$ is defined then k is scheduled as flexible in $\mathcal{X}$ and $r_{f(k)} \le C_k^{\mathcal{X}} \le x_{f(k)}$. Then there exists an EDF schedule $\mathcal{A}$ equivalent to $\mathcal{X}$ such that the following hold:*

(1) *All jobs $f(k)$ are completed in $\mathcal{A}$.*

(2) *Consider a time t when either $\mathcal{A}$ is idle or it starts a job and the set of all its pending jobs is feasible at t. Then all jobs pending at t are completed in $\mathcal{A}$. In particular, each job that is pending when $\mathcal{A}$ starts a flexible job is completed in $\mathcal{A}$.*

Furthermore, if $\mathcal{X}$ is constructed by an online algorithm and $f(k)$ can be determined online at time $C_k^{\mathcal{X}}$ for each flexible job k in $\mathcal{X}$, then $\mathcal{A}$ can be produced by an online algorithm.

Remark. Property (1) is useful in our proofs, since it allows us to modify the schedule computed by the algorithm to resemble more the optimal schedule. Property (2) guarantees that any job planned to be scheduled is indeed scheduled in the future.

Since $\mathcal{A}$ and $\mathcal{X}$ are equivalent, flexible jobs are the same and scheduled at the same times in $\mathcal{A}$ and $\mathcal{X}$. In particular, all the jobs k on which $f(k)$ is defined are scheduled at the same time in both $\mathcal{A}$ and $\mathcal{X}$—a property that will play an important role in our later arguments.

The basic idea of the construction of $\mathcal{A}$ is quite straightforward: Maintain a set Q_t of jobs that we plan to schedule. If the set of all pending jobs is feasible, we always plan to schedule them all. In addition, if we start a flexible job k at time t, the flexibility of k allows us to add to Q_{t+p} an extra job released during the execution of k; so if $f(k)$ is defined, we add $f(k)$.

Proof. We construct $\mathcal{A}$ iteratively. Throughout the proof, t ranges over times when $\mathcal{X}$ is idle or starts a job. For each such t, let P_t and P'_t denote the set of jobs pending in $\mathcal{X}$ and $\mathcal{A}$, respectively.

We will maintain an auxiliary set of jobs Q_t that are pending at t in $\mathcal{X}$ and $\mathcal{A}$, i.e., $Q_t \subseteq P_t \cap P'_t$. Simultaneously with the construction, we prove inductively that, for all t, the following invariant holds:

$(*)$ Q_t is a $\subseteq$-maximal feasible subset of each of P_t and P'_t.

Before describing the construction, we make two observations. First, recall that condition $(*)$ implies that Q_t is also maximum with respect to size. Second, if any of sets P_t, P'_t, Q_t is flexible, then $Q_t = P_t = P'_t$ by the maximality of Q_t.

We now describe the construction. Initially, choose Q_0 as an arbitrary maximal feasible set of the jobs released at time 0.

Assume we have already defined Q_t. If $\mathcal{X}$ is idle at t, we let $\mathcal{A}$ idle and choose an arbitrary $Q_{t+1} \supseteq Q_t$ by adding to Q_t the jobs released at $t+1$, as long as the set

remains feasible. Since $\mathcal{X}$ is idle, P_t is flexible at t, and thus $Q_t = P_t = P'_t$. Therefore Q_t is feasible at $t+1$, $P_{t+1} = P'_{t+1}$, and we can conclude that $(*)$ holds at time $t+1$.

Now suppose that $\mathcal{X}$ starts a job k at time t. We consider two subcases, depending on whether k is flexible or urgent.

Case 1. Job k is flexible in $\mathcal{X}$. Then $\mathcal{A}$ starts k, too. This is possible since in this case we have $Q_t = P_t = P'_t$, and thus k is pending in $\mathcal{A}$ at time t. Note that k is executed as flexible in $\mathcal{A}$. Further, we also have $P'_{t+p} = P_{t+p}$.

Since Q_t is flexible, it is feasible at $t+p$. To construct Q_{t+p}, we start with $Q_{t+p} = Q_t - \{k\}$ and expand it by processing newly released jobs, one by one, adding each processed job into Q_{t+p} if Q_{t+p} remains feasible. We process first the job $f(k)$, if it is defined, pending, and not yet in Q_{t+p}. Then we process the remaining jobs h with $t < r_h \le t+p$ in an arbitrary order. Since Q_t is feasible at $t+p$ and k is the ED job in Q_t, $Q_t - \{k\} \cup \{f(k)\}$ is feasible at $t+p$ as well, so $f(k)$ can always be added to Q_{t+p} without violating feasibility. By the construction, $(*)$ is satisfied at time $t+p$.

Case 2. Job k is urgent in $\mathcal{X}$. $\mathcal{A}$ starts the ED (more precisely, $\prec$-minimal) job k' from Q_t. Since Q_t is a maximal feasible set both for $\mathcal{X}$ and $\mathcal{A}$, it is nonempty whenever $\mathcal{X}$ starts a job. Furthermore, we know that Q_t is not flexible at t and thus k' is urgent.

Let $T = Q_t - \{k'\}$. We claim that

(t1) T is a maximal subset of P_t (resp., P'_t) that is feasible at time $t+p$, and

(t2) $T \subseteq P_{t+p} \cap P'_{t+p}$.

That T is feasible at $t+p$ follows directly from the definition of T and the fact that k' is the ED job in Q_t. For the same reason, all jobs in T are pending in $\mathcal{A}$ at time $t+p$. Since k' is pending in $\mathcal{X}$ at t, and $\mathcal{X}$ schedules the ED pending job (as $\mathcal{X}$ is normal), we have $k \prec k'$. Therefore all jobs in T are pending at time $t+p$ in $\mathcal{X}$ as well. We conclude that (t2) holds.

No job in $P'_t - Q_t$ can be feasibly added to T at time $t+p$, as otherwise it could be feasibly added to Q_t at time t, contradicting the maximality of Q_t for $\mathcal{A}$. The same argument applies to $\mathcal{X}$. Thus, T satisfies condition (t1) for both P_t and P'_t.

We construct Q_{t+p} as in the previous case. We start with $Q_{t+p} = T$ and process newly released jobs in an arbitrary order, one by one, adding each processed job into Q_{t+p} if Q_{t+p} remains feasible. Again, the maximality of T and the construction implies that Q_{t+p} satisfies $(*)$ at time $t+p$.

This completes the construction. Obviously, $\mathcal{X}$ and $\mathcal{A}$ are equivalent. Also, $\mathcal{A}$ is EDF since, whenever it schedules a job, it chooses the ED job of Q_t, and jobs in $P'_t - Q_t$ are never added to Q_s for $s > t$, so they will not be scheduled in $\mathcal{A}$.

By the construction, $\mathcal{A}$ schedules all the jobs that are in some Q_t. At any time t when $\mathcal{A}$ is idle or starts a flexible job, Q_t is flexible and thus $Q_t = P'_t$. This proves Lemma 2.2(2). This also implies Lemma 2.2(1) since, for $t = S_k^{\mathcal{X}}$, $f(k)$ is either completed by time $t+p$ or $f(k) \in Q_{t+p}$. □

Lemma 2.2 gives an easy proof that any normal schedule $\mathcal{X}$ schedules at least half as many jobs as the optimum. Take the modified schedule $\mathcal{A}$ from Lemma 2.2 (with f undefined). Charge any job j completed in an optimal schedule $\mathcal{Z}$ to a job completed in $\mathcal{A}$ as follows: (a) If $\mathcal{A}$ is running a job k at time $S_j^{\mathcal{Z}}$, charge j to k; (b) otherwise, charge j to j. This is well defined since if j is admissible and $\mathcal{A}$ is idle at time $S_j^{\mathcal{Z}}$, then $\mathcal{A}$ completes j by Lemma 2.2(2). Furthermore, only one job can be charged to k using rule (a), as all jobs have the same processing time and only one job can be started in $\mathcal{Z}$ during the interval when k is running in $\mathcal{A}$. Thus overall at most two jobs in $\mathcal{Z}$ are charged to each job in $\mathcal{A}$, and $|\mathcal{Z}| \le 2|\mathcal{A}| = 2|\mathcal{X}|$, as claimed.

This shows that any online algorithm that generates a normal schedule is 2-competitive. In particular, this includes the known result that the greedy algorithm which always schedules the ED pending job when there are any pending jobs is 2-competitive. We use similar but more refined charging schemes to analyze our improved algorithms.

Goldwasser and Kerbikov [15] introduced a concept of online algorithms that upon release of a job immediately commit whether it will be completed or not. We do not formulate our algorithms in this form, but Lemma 2.2 can be applied to normal schedules generated by our algorithms (with f undefined) to obtain equivalent online algorithms with immediate notification. (For the model with restarts, this implies that any preempted job is completed later.) Since the construction produces equivalent schedules, the performance is also the same.

3. Randomized algorithms. In this section we present our 5/3-competitive barely random algorithm. This algorithm uses only one random bit, namely, at the beginning of computation it chooses with probability 1/2 between two deterministic algorithms. We also show two lower bounds for barely random algorithms: Any randomized algorithm that randomly chooses between two schedules has a ratio of at least 3/2. Furthermore, if the two algorithms are selected with equal probability, the competitive ratio is at least 8/5.

Algorithm RandLock. We describe our algorithm in terms of two identical processes that are denoted by $\mathcal{X}$ and $\mathcal{Y}$. Each process is, in essence, a scheduling algorithm that receives its own copy of the input instance J and computes its own schedule for J. (This means that a given job can be executed by both processes, at the same or different times.) We chose to use the term "process" rather than "algorithm" since $\mathcal{X}$ and $\mathcal{Y}$ are not fully independent: they both have access to a shared lock mechanism used to coordinate their behavior.

Each process $\mathcal{X}$ and $\mathcal{Y}$ is defined as follows:

(RL1) If there are no pending jobs, wait until some job is released.

(RL2) If the set of pending jobs is not flexible, execute the ED pending job.

(RL3) If the set of pending jobs is flexible and the lock is available, acquire the lock (ties broken arbitrarily), execute the ED pending job, and release the lock upon its completion.

(RL4) Otherwise, wait until the lock becomes available or the set of pending jobs becomes nonflexible (due to progress of time or new jobs being released).

Algorithm RandLock selects initially one of the two processes $\mathcal{X}$ or $\mathcal{Y}$, each with probability 1/2. Then it simulates the two processes on a given instance, outputting the schedule generated by the selected process.

By the description of the algorithm, at each step only one process, namely, the one that possesses the lock, can be executing a flexible job.

Before we analyze the algorithm, we illustrate its behavior on the instance in Figure 3.1. Both processes schedule only three jobs, while the optimal schedule has five jobs. Thus RandLock is not better than 5/3-competitive.

Theorem 3.1. RandLock *is a* 5/3*-competitive nonpreemptive randomized algorithm for scheduling equal-length jobs.*

Proof. Overloading the notation, let $\mathcal{X}$ and $\mathcal{Y}$ denote the schedules generated by the corresponding processes on a given instance J. By rules (RL2) and (RL3), both schedules are normal. Fix an arbitrary schedule $\mathcal{Z}$ for the given instance J.

We start by modifying the schedules $\mathcal{X}$ and $\mathcal{Y}$ according to Lemma 2.2. Define a partial function $f^{\mathcal{A}} : J \to J$ as follows. Let $f^{\mathcal{A}}(k) = h$ if k is a flexible job completed

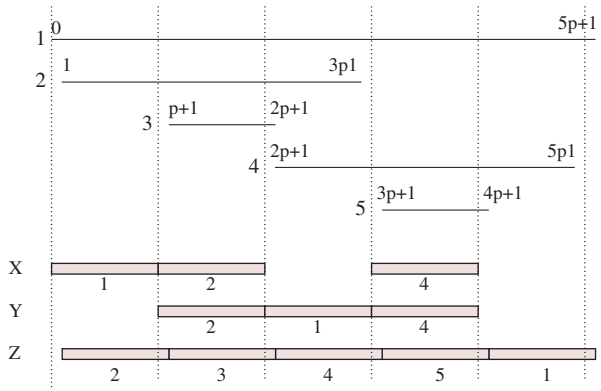


FIG. 3.1. *An instance on which* RANDLOCK *schedules three jobs out of five. At time* 0*, process* $\mathcal{X}$ *acquires the lock and executes job* 1*. Process* $\mathcal{Y}$ *must then wait to execute job* 2 *until it becomes urgent at time* p*. At time* $2p$*,* $\mathcal{Y}$ *acquires the lock and executes* 1*, while* $\mathcal{X}$ *waits with job* 4 *until it becomes urgent at time* $3p$*. Overall, job* 1 *is executed as flexible by both processes; the other jobs are executed as urgent.*

in $\mathcal{X}$ and h is a job started in $\mathcal{Z}$ during the execution of k in $\mathcal{X}$ and admissible at the completion of k in $\mathcal{X}$, i.e., $S_k^{\mathcal{X}} \le S_h^{\mathcal{Z}} < C_k^{\mathcal{X}} \le x_h$. Otherwise (if k is urgent or no such h exists), $f^{\mathcal{A}}(k)$ is undefined. Note that if h exists, it is unique for a given k. Then we define $\mathcal{A}$ to be the schedule constructed from $\mathcal{X}$ in Lemma 2.2 using function $f^{\mathcal{A}}(\cdot)$. Analogously we define function $f^{\mathcal{B}}(\cdot)$, and we modify schedule $\mathcal{Y}$ to obtain schedule $\mathcal{B}$. We stress that these new schedules $\mathcal{A}$ and $\mathcal{B}$ cannot be constructed online as their definition depends on $\mathcal{Z}$; they only serve as tools for the analysis of RANDLOCK.

Since $\mathcal{A}$ (resp., $\mathcal{B}$) is equivalent to a normal schedule $\mathcal{X}$ (resp., $\mathcal{Y}$), Lemma 2.1 still applies to $\mathcal{A}$ (resp., $\mathcal{B}$) and the number of completed jobs remains the same as well.

Throughout the proof we use the convention that whenever $\mathcal{D}$ denotes one of the schedules $\mathcal{A}$ and $\mathcal{B}$, then $\bar{\mathcal{D}}$ denotes the other one. □

LEMMA 3.2. *Let* $\mathcal{D} \in \{\mathcal{A}, \mathcal{B}\}$*, and let* $\bar{\mathcal{D}}$ *be the other process of* RANDLOCK*. Suppose that at time* t $\mathcal{D}$ *is idle or is executing an urgent job and* $\bar{\mathcal{D}}$ *is idle. Then each job admissible at time* t *is completed in* $\bar{\mathcal{D}}$ *as a flexible job by time* t*.*

Proof. The lemma is a direct consequence of the lock mechanism. By the assumption, the lock is available at time t, yet the process corresponding to $\bar{\mathcal{D}}$ does not schedule any job. This is possible only if no job is pending. Consequently, any job k admissible at time t must have been completed in $\bar{\mathcal{D}}$ by time t. Furthermore, if k would be executed as urgent in $\mathcal{D}$ before time t then, since $S_k^{\mathcal{D}} \le t \le x_k$, Lemma 2.1 implies that $\mathcal{D}$ could not be idle at time t. This shows that k is completed as a flexible job. □

The charging scheme. Our proof is based on a charging scheme. The fundamental principle of this scheme is the same as in the proof for the greedy algorithm in section 2. Each adversary job will generate a charge of 1. This charge will be distributed among the jobs in schedules $\mathcal{A}$ and $\mathcal{B}$ in such a way that each job in these schedules will receive a charge of at most 5/6. This will imply the 5/3 bound on the competitive ratio of RANDLOCK.

Let j be a job started in $\mathcal{Z}$ at time $t = S_j^{\mathcal{Z}}$. This job generates several charges of different weights to (the occurrences of) the jobs in schedules $\mathcal{A}$ and $\mathcal{B}$. Each charge

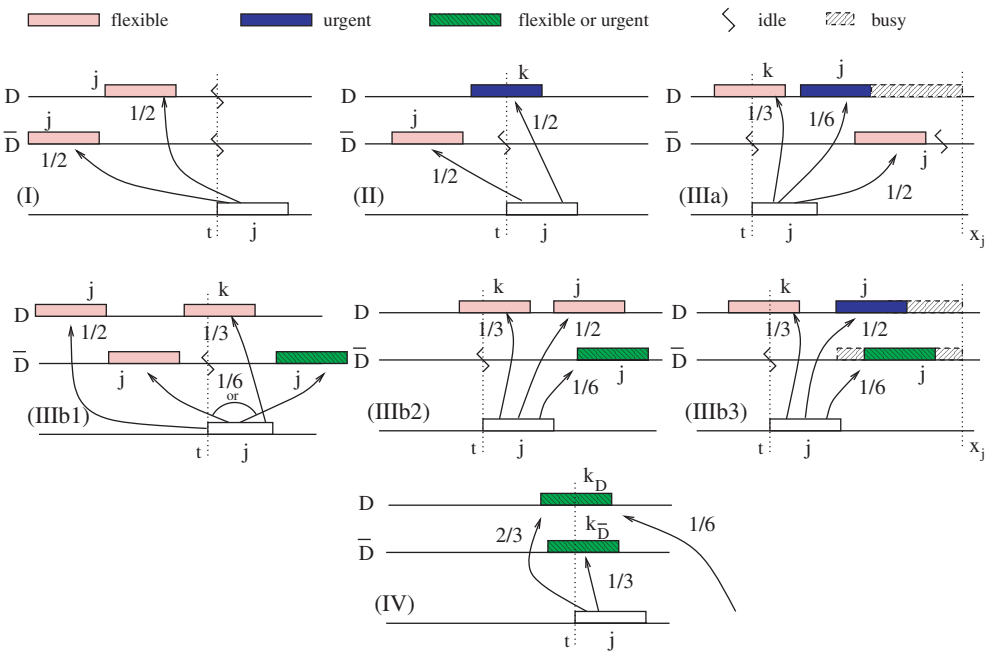


FIG. 3.2. *Illustration of the charging scheme in the analysis of Algorithm* RANDLOCK. *The figure gives examples of different types of charges. In case* (IIIb)*, there are several illustrations that cover possibilities playing a different role in the proof. (To reduce the number of cases, in the figures for case* (III) *we assume that* $j \neq k$ *and* j *is executed in* $\mathcal{D}$ *before it is executed in* $\bar{\mathcal{D}}$*.)*

is uniquely labeled as a *self-charge* or an *up-charge*. Self-charges from j go to the occurrences of j in $\mathcal{A}$ or $\mathcal{B}$, and up-charges from j go to the jobs running at time t in $\mathcal{A}$ and $\mathcal{B}$. If one of the processes runs j at time t, then the charge to this job may be designated as either an up-charge or a self-charge; in case (III) below such a j can even receive both a self-charge and an up-charge from j. The total of charges generated by j is always 1. The charges depend on the status of $\mathcal{A}$ and $\mathcal{B}$ at time t. (See Figure 3.2.)

(I) Both schedules $\mathcal{A}$ and $\mathcal{B}$ are idle. By Lemma 3.2, in both $\mathcal{A}$ and $\mathcal{B}$, j is completed as flexible by time t. We generate two self-charges of $1/2$ to the two occurrences of j in $\mathcal{A}$ and $\mathcal{B}$.

(II) One schedule $\mathcal{D} \in \{\mathcal{A}, \mathcal{B}\}$ is running an urgent job k and the other schedule $\bar{\mathcal{D}}$ is idle. By Lemma 3.2, in $\bar{\mathcal{D}}$, j is completed as flexible by time t. We generate a self-charge of $1/2$ to the occurrence of j in $\bar{\mathcal{D}}$ and an up-charge of $1/2$ to k in $\mathcal{D}$.

(III) One schedule $\mathcal{D} \in \{\mathcal{A}, \mathcal{B}\}$ is running a flexible job k and the other schedule $\bar{\mathcal{D}}$ is idle. We claim that j is completed in both $\mathcal{A}$ and $\mathcal{B}$. For $\bar{\mathcal{D}}$, this follows directly from Lemma 2.2(2). We now prove it for $\mathcal{D}$. If $r_j \leq S_k^{\mathcal{D}}$, then Lemma 2.2(2) applied to time $t' = S_k^{\mathcal{D}}$ implies that $\mathcal{D}$ completes j. If $x_j \geq C_k^{\mathcal{D}}$, then $f^{\mathcal{D}}(k) = j$, so $\mathcal{D}$ completes j by Lemma 2.2(1). The remaining case, namely, $S_k^{\mathcal{D}} < r_j \leq t \leq x_j < C_k^{\mathcal{D}}$, cannot happen, since this condition implies that j is tight and thus the set of jobs pending at time t for $\bar{\mathcal{D}}$ is not flexible. So $\bar{\mathcal{D}}$ would not be idle at t, contradicting the case condition.

In this case we generate one up-charge of $1/3$ to k in $\mathcal{D}$ and two self-charges of $1/2$ and $1/6$ to the occurrences of j according to the two subcases below. Let $\mathcal{E} \in \{\mathcal{A}, \mathcal{B}\}$ be the schedule which starts j first (breaking ties arbitrarily).

(IIIa) If $\mathcal{E}$ schedules j as an urgent job and the other schedule $\bar{\mathcal{E}}$ is idle at some time t' satisfying $S_j^{\mathcal{E}} \le t' \le x_j$, then charge $1/6$ to the occurrence of j in $\mathcal{E}$ and $1/2$ to the occurrence of j in $\bar{\mathcal{E}}$.

We make here a few observations that will be useful later in the proof. Since in this case j is urgent in $\mathcal{E}$ and $\mathcal{E}$ is either idle or executes a flexible job at time t, Lemma 2.1 implies that j is executed in $\mathcal{E}$ after time t. It also implies that $\mathcal{E}$ runs urgent jobs between $S_j^{\mathcal{E}}$ and x_j. This means that $\mathcal{E}$ runs an urgent job at t'. Since $\bar{\mathcal{E}}$ is idle at time t' by the case condition, Lemma 3.2 implies that $\bar{\mathcal{E}}$ schedules j as flexible before time t'.

(IIIb) Otherwise, charge $1/2$ to the occurrence of j in $\mathcal{E}$ and $1/6$ to the occurrence of j in $\bar{\mathcal{E}}$.

(IV) Both processes $\mathcal{A}$ and $\mathcal{B}$ are running jobs $k_{\mathcal{A}}$ and $k_{\mathcal{B}}$, respectively, at time t. We show below in Lemma 3.4 that in the previous cases either $k_{\mathcal{A}}$ or $k_{\mathcal{B}}$ receives a self-charge of at most $1/6$ from its occurrence in $\mathcal{Z}$. We generate an up-charge of $2/3$ from j to this job and an up-charge of $1/3$ to the other one. No self-charge is generated in this case.

This completes the description of the charging scheme. Before we resume the proof of the theorem, we prove two lemmas, the purpose of which is to justify the correctness of the charges in case (IV).

LEMMA 3.3. *Assume that case* (IV) *applies to j. Suppose also that $k_{\mathcal{F}}$ for some $\mathcal{F} \in \{\mathcal{A}, \mathcal{B}\}$ is scheduled before j in $\mathcal{Z}$ (i.e., $S_{k_{\mathcal{F}}}^{\mathcal{Z}} \le t - p$), and that $k_{\mathcal{F}}$ in $\mathcal{F}$ receives a self-charge of* $1/2$ *generated in case* (IIIb) *applied to $k_{\mathcal{F}}$. Then $k_{\bar{\mathcal{F}}} \prec k_{\mathcal{F}}$ or $k_{\bar{\mathcal{F}}} = k_{\mathcal{F}}$.*

Proof. Since $k_{\mathcal{F}}$ receives a charge of $1/2$ in (IIIb), the choice of $\mathcal{E}$ in case (III) implies that $k_{\mathcal{F}}$ is executed in $\bar{\mathcal{F}}$ later than in $\mathcal{F}$, i.e., $S_{k_{\mathcal{F}}}^{\bar{\mathcal{F}}} \ge S_{k_{\mathcal{F}}}^{\mathcal{F}} > t - p$. On the other hand, $S_{k_{\bar{\mathcal{F}}}}^{\bar{\mathcal{F}}} \le t$, so $k_{\mathcal{F}}$ must be executed in $\bar{\mathcal{F}}$ after $k_{\bar{\mathcal{F}}}$. Furthermore, $S_{k_{\bar{\mathcal{F}}}}^{\bar{\mathcal{F}}} > t - p \ge S_{k_{\mathcal{F}}}^{\mathcal{Z}} \ge r_{k_{\mathcal{F}}}$, and thus $k_{\mathcal{F}}$ is pending in $\bar{\mathcal{F}}$ when $k_{\bar{\mathcal{F}}}$ is started. Since $\bar{\mathcal{F}}$ is EDF, we have $k_{\bar{\mathcal{F}}} \prec k_{\mathcal{F}}$ or $k_{\bar{\mathcal{F}}} = k_{\mathcal{F}}$, completing the proof. □

LEMMA 3.4. *Assume that case* (IV) *applies to j. Then for some $\mathcal{D} \in \{\mathcal{A}, \mathcal{B}\}$ the self-charge to $k_{\mathcal{D}}$ in $\mathcal{D}$ does not exceed* $1/6$.

Proof. Note that self-charges are generated only in cases (I)–(III) and any self-charge has weight $1/2$ or $1/6$. Assume, towards contradiction, that both $k_{\mathcal{A}}$ and $k_{\mathcal{B}}$ receive a self-charge of $1/2$. At least one of $k_{\mathcal{A}}$ and $k_{\mathcal{B}}$ is scheduled as urgent in the corresponding schedule, due to the lock mechanism. Thus $k_{\mathcal{A}} \ne k_{\mathcal{B}}$, as (I) is the only case when two self-charges $1/2$ to the same job are generated and then both occurrences are flexible. Furthermore, if $j = k_{\mathcal{G}}$ for some $\mathcal{G} \in \{\mathcal{A}, \mathcal{B}\}$, then $k_{\mathcal{G}}$ would not receive any self-charge. Thus $k_{\mathcal{A}}$, $k_{\mathcal{B}}$, and j are three distinct jobs.

Choose $\mathcal{D}$ such that $k_{\mathcal{D}}$ is urgent in $\mathcal{D}$ (as noted above, such $\mathcal{D}$ exists). The only case when an urgent job receives a self-charge of $1/2$ is (IIIb). By Lemma 2.1, $\mathcal{D}$ executes urgent jobs at all times t', $t \le t' \le x_{k_{\mathcal{D}}}$, which, together with the condition for case (III) applied to $k_{\mathcal{D}}$ (namely, that $\mathcal{D}$ is either idle or executes a flexible job at $S_{k_{\mathcal{D}}}^{\mathcal{Z}}$), implies that $S_{k_{\mathcal{D}}}^{\mathcal{Z}} \le t$. As $j \ne k_{\mathcal{D}}$, it follows that $S_{k_{\mathcal{D}}}^{\mathcal{Z}} \le t - p$. By Lemma 3.3, $k_{\bar{\mathcal{D}}} \prec k_{\mathcal{D}}$ and $x_{k_{\bar{\mathcal{D}}}} \le x_{k_{\mathcal{D}}}$. Furthermore, since (IIIa) does not apply to $k_{\mathcal{D}}$, $\bar{\mathcal{D}}$ is also not idle at any time t', $t \le t' \le x_{k_{\mathcal{D}}}$.

We now show that the assumption of a self-charge of 1/2 to $k_{\bar{\mathcal{D}}}$ in $\bar{\mathcal{D}}$ leads to a contradiction. The proof is accomplished by considering several cases. In most cases the contradiction is with the fact that, as shown in the previous paragraph, both processes are busy at all times between t and $x_{k_{\mathcal{D}}}$ (keeping in mind that $x_{k_{\bar{\mathcal{D}}}} \le x_{k_{\mathcal{D}}}$).

If this charge is generated in case (I) or (II), then by the case conditions, $\bar{\mathcal{D}}$ would be idle at time $S^{\mathcal{Z}}_{k_{\bar{\mathcal{D}}}}$, and we would have $S^{\mathcal{Z}}_{k_{\bar{\mathcal{D}}}} \ge S^{\bar{\mathcal{D}}}_{k_{\bar{\mathcal{D}}}}$ and thus $t \le S^{\mathcal{Z}}_{k_{\bar{\mathcal{D}}}} \le x_{k_{\bar{\mathcal{D}}}}$, which is a contradiction.

Suppose that this self-charge is generated in case (III). Similarly as before, the condition of this case implies that one process is idle at time $S^{\mathcal{Z}}_{k_{\bar{\mathcal{D}}}}$, so we must have $S^{\mathcal{Z}}_{k_{\bar{\mathcal{D}}}} \le t$, for otherwise we would again have an idle time between t and $x_{k_{\bar{\mathcal{D}}}}$.

We have now two subcases. If the self-charge originated from case (IIIa), the condition of this case implies that there is an idle time t' between $S^{\mathcal{Z}}_{k_{\bar{\mathcal{D}}}}$ and $x_{k_{\bar{\mathcal{D}}}}$. As $t \le t' \le x_{k_{\bar{\mathcal{D}}}}$, this is again a contradiction.

The last possibility is that this self-charge originated from case (IIIb). But then $S^{\mathcal{Z}}_{k_{\bar{\mathcal{D}}}} \le t - p$ as $j \ne k_{\bar{\mathcal{D}}}$, and Lemma 3.3 above applies to $k_{\bar{\mathcal{D}}}$. However, the conclusion that $k_{\mathcal{D}} \prec k_{\bar{\mathcal{D}}}$ contradicts the linearity of $\prec$ as $k_{\mathcal{D}} \ne k_{\bar{\mathcal{D}}}$, and we have already shown that $k_{\bar{\mathcal{D}}} \prec k_{\mathcal{D}}$.

Summarizing, we get a contradiction in all the cases, completing the proof of the lemma. □

Continuing the proof of the theorem, we now show that the total charge to each occurrence of a job in $\mathcal{A}$ or $\mathcal{B}$ is at most 5/6. Suppose that k is executed in $\mathcal{D} \in \{\mathcal{A}, \mathcal{B}\}$. During the execution of k at most one job is started in $\mathcal{Z}$; thus, k gets at most one up-charge in addition to a possible self-charge. If k does not receive any up-charge, it is self-charged 1/2 or 1/6, i.e., less than 5/6.

If k receives an up-charge in (II), then k is an urgent job and, since $\bar{\mathcal{D}}$ is idle, it is already completed in $\bar{\mathcal{D}}$, so $S^{\bar{\mathcal{D}}}_k < S^{\mathcal{D}}_k$. The only case where the occurrence of k that is later in time is urgent and receives a self-charge is case (IIIb), and in this case this self-charge is 1/6. So the total charge would be at most $1/6 + 1/2 < 5/6$.

If a job receives an up-charge in (III), the up-charge is only 1/3, and thus the total is at most $1/3 + 1/2 = 5/6$.

If a job receives an up-charge in (IV), Lemma 3.4 implies that the up-charges can be defined as claimed in the case description. The total charge is then bounded by $1/6 + 2/3 = 5/6$ and $1/2 + 1/3 = 5/6$, respectively.

The expected number of jobs completed by RandLock is $(|\mathcal{A}| + |\mathcal{B}|)/2$. Since each job in $\mathcal{A}$ and $\mathcal{B}$ receives a charge of at most 5/6 and all jobs in $\mathcal{Z}$ generate a charge of 1, we have $(5/3) \cdot (|\mathcal{A}| + |\mathcal{B}|)/2 = (5/6) \cdot (|\mathcal{A}| + |\mathcal{B}|) \ge |\mathcal{Z}|$. This implies that RandLock is 5/3-competitive. □

As discussed in the introduction, a lower bound of 4/3 is known for randomized algorithms [13]. For barely random algorithms that choose between two deterministic algorithms, we can improve this bound to 3/2. Assuming also that the two algorithms are selected with equal probability, we can further improve the bound to 8/5.

THEOREM 3.5. *Suppose that $\mathcal{R}$ is a barely random nonpreemptive algorithm for scheduling equal-length jobs that chooses one of two deterministic algorithms. Then $\mathcal{R}$ is not better than 3/2-competitive.*

Proof. Assume that $\mathcal{R}$ chooses randomly one of two deterministic algorithms, $\mathcal{A}$ or $\mathcal{B}$, with some arbitrary probabilities. Let $p \ge 3$, and write the jobs as $j = (r_j, d_j)$. We start with job $1 = (0, 4p)$. Let t be the first time when one of the algorithms, say $\mathcal{A}$, schedules job 1. If $\mathcal{B}$ schedules it at t as well, release a job $1' = (t+1, t+p+1)$; the optimum schedules both jobs while both $\mathcal{A}$ and $\mathcal{B}$ schedule only one, so the competitive ratio is at least 2.

So we may assume that $\mathcal{B}$ is idle at t. Release job $2 = (t+1, t+2p+2)$. If $\mathcal{B}$ starts any job (1 or 2) at $t+1$, release job $3 = (t+2, t+p+2)$; otherwise, release job $4 = (t+p+1, t+2p+1)$. $\mathcal{B}$ completes only one of the jobs 2, 3, or 4. Since $\mathcal{A}$ is busy with job 1 until time $t+p$, it also completes only one of the jobs 2, 3, or 4 as their deadlines are smaller than $t+3p$. So each of $\mathcal{A}$ and $\mathcal{B}$ completes at most two jobs.

The optimal schedule completes three jobs: If 3 is issued, schedule 3 and 2, back to back, starting at time $t+2$. If 4 is issued, schedule 2 and 4, back to back, starting at time $t+1$. In either case, two of jobs 2, 3, and 4 fit in the interval $[t+1, t+2p+2)$. If $t \geq p-1$, schedule job 1 at time 0; otherwise, schedule job 1 at time $3p \geq t+2p+2$. Thus the competitive ratio of $\mathcal{R}$ is at least $3/2$. □

THEOREM 3.6. *Suppose that $\mathcal{R}$ is a barely random nonpreemptive algorithm for scheduling equal-length jobs that chooses one of two deterministic algorithms, each with probability $1/2$. Then $\mathcal{R}$ is not better than $8/5$-competitive.*

Proof. Assume that $\mathcal{R}$ chooses one of two deterministic algorithms, $\mathcal{A}$ or $\mathcal{B}$, each with probability $1/2$. Let $p \geq 3$, and write the jobs in the format $j = (r_j, d_j)$. We start with job $1 = (0, 6p)$. Let t be the first time when one of the algorithms, say $\mathcal{A}$, schedules job 1.

At time $t+1$ release job $2 = (t+1, t+p+1)$. If $\mathcal{B}$ does not start 2 at time $t+1$, then no more jobs will be released and the ratio is at least 2.

We may thus assume that $\mathcal{B}$ starts 2 at time $t+1$ and then starts 1 at some time $t' \geq t+p+1$. Release job $3 = (t'+1, t'+2p+2)$. If $\mathcal{A}$ starts job 3 at $t'+1$, release job $4 = (t'+2, t'+p+2)$; otherwise, release $5 = (t'+p+1, t'+2p+1)$. By the choice of the last job, $\mathcal{A}$ can complete only one of the jobs 3, 4, or 5. Since $\mathcal{B}$ is busy with job 1 until time $t'+p \geq t'+3$, it also can complete only one of the jobs 3, 4, or 5 as their deadlines are strictly smaller than $t'+3p$. So $\mathcal{A}$ can complete 2 jobs only, and $\mathcal{B}$ can complete 3 jobs.

The optimal schedule can complete all four released jobs. If 4 is issued, schedule 4, and 3, back to back, starting at time $t'+2$. If 5 is issued, schedule 3, and 5, back to back, starting at time $t'+1$. In either case, both jobs fit in the interval $[t'+1, t'+2p+2)$. This interval is disjoint with the interval $[t+1, t+p+1)$ where 2 is scheduled. Finally, these two intervals occupy length $3p+1$ of the interval $[0, 6p)$ and divide it into at most 3 contiguous pieces; thus, one of the remaining pieces has a length of at least p, and job 1 can be scheduled.

Summarizing, $\mathcal{R}$ completes at most $(2+3)/2 = 2.5$ jobs on average, while the optimal schedule completes 4 jobs. Therefore the competitive ratio is at least $4/2.5 = 8/5$, as claimed. □

4. Scheduling with restarts. Our algorithm with restarts is very natural. At any time, it greedily schedules the ED job. However, if a tight job arrives that would expire before the running job is completed, we consider a preemption. A preemption occurs only if it guarantees to increase the number of completed jobs among those that are known to the algorithm, which includes the currently executed job and all pending jobs.

To formalize this idea, we need an auxiliary definition. Suppose that a job k is started at time s by the online algorithm. We call a job h a *preemption candidate* for k if $s < r_h \leq x_h < s+p$.

The exact statement of the algorithm is somewhat technical, as it needs to properly handle the case when two preemption candidates arrive at the same time and also the case when some other jobs arrive between the start of a job and the arrival of the first preemption candidate.

ALGORITHM TIGHTRESTART. At time t, do the following.

(TR1) If no job is running, start the ED pending job, providing there is at least one pending job; otherwise, stay idle until some job is released.

(TR2) Otherwise, let k be the running job. If k was started as urgent or if no preemption candidate is released at t, continue running k.

(TR3) Otherwise, the running job k was started as flexible. Let P_t^* be the set of all jobs pending at time t, including k but excluding any preemption candidates. If P_t^* is flexible at t, preempt k and start (at time t) a preemption candidate; choose the ED preemption candidate, if more are admissible at time t. Otherwise, continue running k.

Note that in case (TR3) job k is indeed still pending at time t, for its flexibility at its start time implies that k is still admissible at t. (Recall that only admissible jobs are considered pending, and a job that is partially executed is pending as well, as long as it is still admissible.)

Let $\mathcal{X}$ be the final schedule generated by TIGHTRESTART, after removing the preempted parts of jobs. For any time t, as before, we denote as P_t the set of jobs that are pending in $\mathcal{X}$ at time t. We stress that we distinguish between $\mathcal{X}$ being idle and TIGHTRESTART being idle: At some time steps TIGHTRESTART can process a job that will be preempted later, in which case $\mathcal{X}$ is considered idle at these steps but TIGHTRESTART is not.

LEMMA 4.1. *Schedule $\mathcal{X}$ is normal.*

Proof. By rules (TR1) and (TR3), TIGHTRESTART always starts the ED pending job; in (TR3) note that, by definition, any preemption candidate is tight and thus it has an earlier deadline than any job in the flexible set P_t^* of the remaining pending jobs. The property (n1) of normal schedules follows as, obviously, at each time step, the pending jobs in TIGHTRESTART and $\mathcal{X}$ are the same.

If TIGHTRESTART is idle then there is no pending job. Thus, to show the property (n2), it remains to verify that $P_{t'}$ is flexible at any time t' when $\mathcal{X}$ is idle but TIGHTRESTART is not. This means that TIGHTRESTART is running a job which is later preempted.

Suppose TIGHTRESTART starts a job k at time s and preempts it at time t. By (TR2), k is started as flexible. Let t' be any time such that $s < t' < t$. Since k is flexible at s and it is the ED job in P_s, no job in P_s expires before $s + p > t$. Thus we have $P_s \subseteq P_{t'}^* \subseteq P_t^*$ by the definition of $P_{t'}^*$ and P_t^* in (TR3). As TIGHTRESTART preempts at time t, P_t^* is flexible at t. Consequently, $P_{t'}^* \subseteq P_t^*$ is flexible at t and also at $t' < t$. Using this for all t', we conclude that the first preemption candidate for k is released at t, as otherwise k would be preempted earlier. Thus no preemption candidate is admissible at any t', $s < t' < t$, and $P_{t'} = P_{t'}^*$ which we have shown is flexible at t'. Thus (n2) holds, and $\mathcal{X}$ is normal. □

THEOREM 4.2. TIGHTRESTART *is a 3/2-competitive algorithm with restarts for scheduling equal-length jobs.*

Proof. As usual, by $\mathcal{Z}$ we denote an optimal schedule. The proof is based on a charging scheme, where each job in $\mathcal{Z}$ generates a charge of 1 and each job in TIGHTRESTART's schedule receives a charge of at most 3/2.

Let us start by giving some intuition behind the charging scheme. Suppose that a job j is started at time t in $\mathcal{Z}$. If TIGHTRESTART is running a job k at t and k is not preempted later, we want to charge j to k. If TIGHTRESTART is running a job k which is later preempted by a job h, we charge 1/2 to h and 1/2 to j (using Lemma 2.2 to guarantee that the modified schedule completes j). The main problem

is to handle the case when TightRestart is idle when j starts in $\mathcal{Z}$; we call such a j a *free* job. In this case, TightRestart was "tricked" into scheduling j too early. We would like to charge j to itself. However, it may happen that then j would be charged twice, so we need to split this charge and find another job that we can charge $1/2$. The definition of $f(j)$ below chooses such a job and Lemma 2.2 again guarantees that the modified schedule completes $f(j)$. The rule (f2) in the definition of f below chooses a value of f to be a job not scheduled in $\mathcal{Z}$, which is opposite to the general intuition that the modified schedule is more similar to $\mathcal{Z}$; however, it guarantees that this job may be charged that additional $1/2$. Another difficulty that arises in the above scheme is that, due to preemptions in TightRestart's schedule and to idle times in $\mathcal{Z}$, the jobs can become misaligned. To deal with this problem, we define a matching M between the jobs in $\mathcal{X}$ and $\mathcal{Z}$. Typically, a job k in $\mathcal{X}$ is matched to the first unmatched job in $\mathcal{Z}$ that starts later than k. In some situations we match a free job k to itself.

We now proceed with the formal proof. First, we define a partial function $f : J \to J$. For any job k scheduled as flexible in $\mathcal{X}$, we define $f(k)$ as follows.

(f1) If at some time t, $S_k^{\mathcal{X}} \le t < C_k^{\mathcal{X}}$, $\mathcal{Z}$ starts a job h which is not a preemption candidate, then let $f(k) = h$.

(f2) Otherwise, if there exists a job h with $S_k^{\mathcal{X}} < r_h \le C_k^{\mathcal{X}} \le x_h$ such that $\mathcal{Z}$ does not complete h, then let $f(k) = h$ (choose arbitrarily if there are more such h's).

(f3) Otherwise, $f(k)$ is undefined.

Notice that $f(\cdot)$ is one-to-one, for the first two cases are disjoint, and in each case k is uniquely determined by $h = f(k)$: If $h = f(k)$ and (f1) applied to k, then k is the job that is being executed by $\mathcal{X}$ when $\mathcal{Z}$ starts h. If (f2) applied to k, then k is the job being executed by $\mathcal{X}$ at time $r_h - 1$. □

According to Lemma 4.1, $\mathcal{X}$ is a normal schedule. Let $\mathcal{A}$ be the schedule constructed in Lemma 2.2 from $\mathcal{X}$ and function $f(\cdot)$. Since $\mathcal{A}$ is equivalent to $\mathcal{X}$, it also satisfies Lemma 2.1.

Call a job j scheduled in $\mathcal{Z}$ a *free* job if TightRestart is idle at time $S_j^{\mathcal{Z}}$. This condition implies that at time $S_j^{\mathcal{Z}}$ no job is pending in $\mathcal{A}$; in particular, by Lemma 2.1, j is completed as a flexible job by time $S_j^{\mathcal{Z}}$ in $\mathcal{A}$.

Now define a partial function $M : J \to J$ which is a matching of (some) occurrences of jobs in $\mathcal{A}$ to those in $\mathcal{Z}$. Process the jobs k scheduled in $\mathcal{A}$ in the order of increasing $S_k^{\mathcal{A}}$. For a given k, let j be the first unmatched job in $\mathcal{Z}$ started at or after $S_k^{\mathcal{A}}$ or, more formally, the job with smallest $S_j^{\mathcal{Z}}$ among those with $S_j^{\mathcal{Z}} \ge S_k^{\mathcal{A}}$ and such that $j \ne M(k')$ for all k' in $\mathcal{A}$ with $S_{k'}^{\mathcal{A}} < S_k^{\mathcal{A}}$. If no such j exists, $M(k)$ is undefined. Else, do the following.

(m1) If $S_j^{\mathcal{Z}} \ge C_k^{\mathcal{A}}$ and k is a free job which is not in the current range of M, then let $M(k) = k$.

(m2) Otherwise, let $M(k) = j$.

The definition implies that M is one-to-one. See Figure 4.1 for an example.

Lemma 4.3. *Let j be a job executed in $\mathcal{Z}$.*

(1) *If $\mathcal{A}$ executes some job when j starts in $\mathcal{Z}$, i.e., $S_k^{\mathcal{A}} \le S_j^{\mathcal{Z}} < C_k^{\mathcal{A}}$ for some k, then j is in the range of M.*

(2) *If j is free and $f(j)$ is undefined, then j is in the range of M.*

Proof. Part (2) is simple: Suppose that $\mathcal{A}$ is executing some job k at $S_j^{\mathcal{Z}}$, and consider the step in the construction of M when we are about to define $M(k)$. If j is not in the range of M at this time, then we would define $M(k)$ as j.

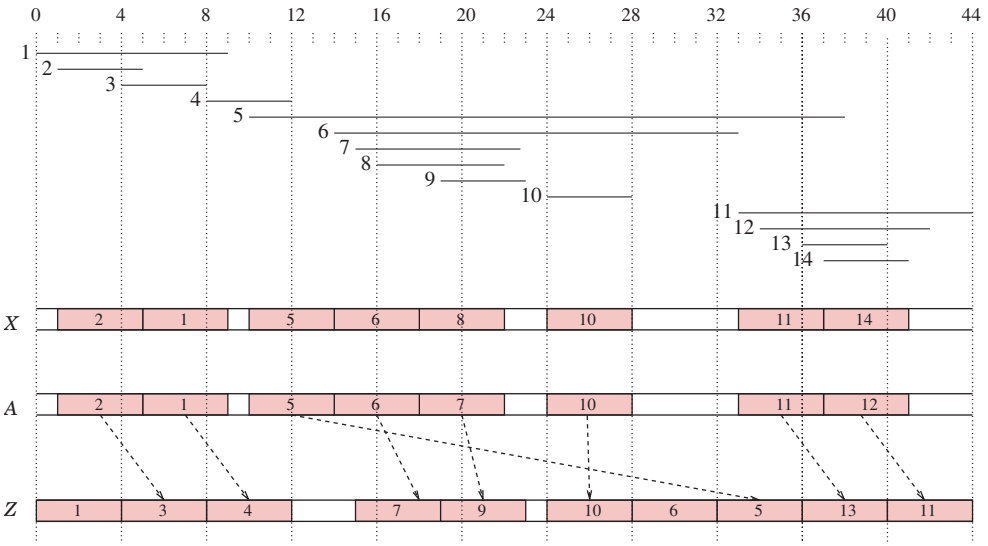


FIG. 4.1. *An example of an instance, the schedule $\mathcal{X}$ produced by* TIGHTRESTART*, the modified schedule $\mathcal{A}$, and the construction of M (represented by directed edges). The processing time is $p = 4$. Jobs are identified by positive integers. Preempted pieces of jobs are not shown. In $\mathcal{X}$, jobs* 5, 6, *and* 11 *are flexible, and the other jobs are urgent. Note that $f(5)$ is undefined, $f(6) = 7$ (since* 7 *is not a preemption candidate), and $f(11) = 12$ (since* 13 *is a preemption candidate and* 12 *is not, and* 12 *is not executed in X).*

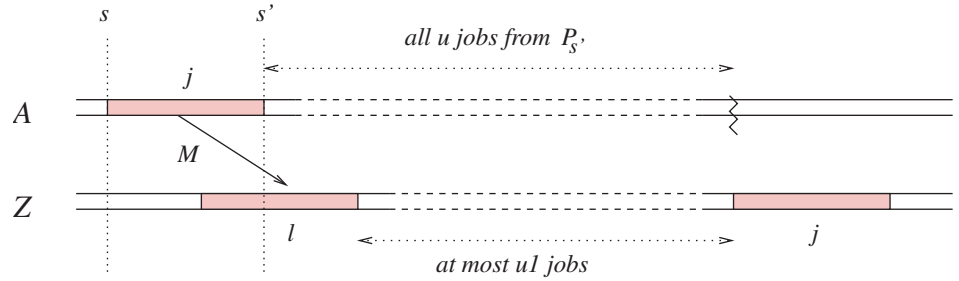


FIG. 4.2. *Illustration to the proof of Lemma* 4.3(2).

We now prove (2). Let $s = S_j^{\mathcal{A}}$ be the start time of j in $\mathcal{A}$ and $s' = C_j^{\mathcal{A}} = s + p$ its completion time. Since j is free, it is completed in $\mathcal{A}$ before it is started in $\mathcal{Z}$, i.e., $S_j^{\mathcal{Z}} \geq s'$ and j is flexible in $\mathcal{A}$.

Suppose for a contradiction that j is not in the range of M. By the definition of M, this implies that $M(j) = l$ for some job l with $s \leq S_l^{\mathcal{Z}} < s'$. Otherwise, during the construction of M when we are about to define $M(j)$, we would set $M(j) = j$.

Since $f(j)$ is undefined, by condition (f1), l must be a preemption candidate for j, i.e., $s < r_l \leq x_l < s'$. Furthermore, as TIGHTRESTART does not preempt j when l is released, the set $P^*_{r_l}$ is not flexible.

Figure 4.2 illustrates the argument that follows. The idea is this: Since j is not preempted even though a preemption candidate l arrives, $\mathcal{A}$ must be nearly full between s' and d_j. So, intuitively, one of the jobs scheduled in this interval should overlap in time with the occurrence of j in $\mathcal{Z}$, and this job would end up being

matched to j. The rigorous argument gets a bit technical because of possible gaps in the schedules.

Let $H = \{h \mid s < r_h \le s' \le x_h\}$ be the set of all jobs released during the execution of j in $\mathcal{A}$ or exactly at $C_j^{\mathcal{A}}$, excluding preemption candidates. Since $f(j)$ is undefined, by condition (f2), all these jobs are completed in $\mathcal{Z}$, and obviously they cannot be completed before $S_l^{\mathcal{Z}}$. Also, $l \notin H$. Thus H is feasible at $C_l^{\mathcal{Z}}$ and also at $s' \le C_l^{\mathcal{Z}}$.

Since $\mathcal{A}$ is an EDF schedule and j is flexible in $\mathcal{A}$, all jobs $h \in P_s - \{j\}$ have $x_h \ge x_j \ge s'$, so they are still pending at s'. Therefore $P_{s'} = P_s \cup H - \{j\} = P^*_{r_l} \cup H - \{j\}$. (Job j is not pending at s' since it is already completed.)

We claim that $P_{s'}$ is feasible at s'. Suppose, towards contradiction, that it is not. Let d be smallest time such that $R = \{h \in P_{s'} \mid d_h \le d\}$ is not feasible; i.e., R is the smallest infeasible initial segment of $P_{s'}$ ordered by $\prec$. Then TightRestart would execute urgent jobs from s' until time $d - p + 1$, as always the ED job is started as urgent and then the set of pending jobs cannot become feasible before or at time $d-p$. Since $\mathcal{X}$ is idle at time $S_j^{\mathcal{Z}}$, this implies that $d < C_j^{\mathcal{Z}}$. Since all jobs $h \in P_{s'}$ with $d_h < d_j$ are in H, this implies that $R \subseteq H$, which is a contradiction with feasibility of H. We conclude that $P_{s'}$ is feasible at s', as claimed.

Since $P_{s'}$ is feasible at s', Lemma 2.2(2) implies that $\mathcal{A}$ completes all jobs in $P_{s'}$. Furthermore, all jobs in $P_{s'}$ are scheduled between s' and $S_j^{\mathcal{Z}}$, as TightRestart is idle at $S_j^{\mathcal{Z}}$.

Let $u = |P_{s'}|$. Next we claim that

(i) $S_j^{\mathcal{Z}} - C_l^{\mathcal{Z}} < up$, and

(ii) $\mathcal{Z}$ does not schedule any of the jobs in $P_{s'}$ after j.

If either of (i) or (ii) were violated, $P_{s'} \cup \{j\}$ would be feasible at $C_l^{\mathcal{Z}}$, for we can first schedule H, which is feasible at $C_l^{\mathcal{Z}}$, and then the remaining jobs from $P_{s'}$: If (i) is violated, we can complete all jobs in $P_{s'}$ by the time $S_j^{\mathcal{Z}}$, which is smaller than the deadlines in $P^*_{r_l} - H$, and start j at $S_j^{\mathcal{Z}}$. If (ii) is violated, let j' be the job in $P_{s'}$ scheduled after j in $\mathcal{Z}$. We know that $S_j^{\mathcal{Z}} - C_l^{\mathcal{Z}} > S_j^{\mathcal{Z}} - s' - p \ge (u-1)p$; thus, we can complete all jobs in $P_{s'}$ by the time $S_j^{\mathcal{Z}}$ and schedule j and j' as in $\mathcal{Z}$.

By the previous paragraph, if either (i) or (ii) does not hold, then $P^*_{r_l} \cup \{j\} \subseteq P_{s'} \cup \{j\}$ is feasible at $r_l + p \le C_l^{\mathcal{Z}}$ and thus flexible at r_l, contradicting the assumption that l (which is a preemption candidate) did not cause preemption. We thus obtain that (i) and (ii) are true, as claimed.

Summarizing, $\mathcal{A}$ completes the u jobs in $P_{s'}$ between s' and $S_j^{\mathcal{Z}}$, and by (ii), these jobs are not executed after $S_j^{\mathcal{Z}}$ in $\mathcal{Z}$. Therefore, if j were not in the range of M, the jobs in $P_{s'}$ would have to be matched to the jobs in $\mathcal{Z}$ between $C_l^{\mathcal{Z}}$ and $S_j^{\mathcal{Z}}$, which is not possible, because there are at most $u-1$ such jobs by (i). We can thus conclude that j is indeed in the range of M. □

Charging scheme. Let j be a job started at time $t = S_j^{\mathcal{Z}}$ in $\mathcal{Z}$. We charge j to jobs in $\mathcal{A}$ according to the following cases.

(I) $j = M(k)$ for some k. Charge j to k. By Lemma 4.3(1), this case always applies when $\mathcal{A}$ is not idle at t, so in the remaining cases $\mathcal{A}$ is idle at t.

(II) Otherwise, if j is free, then charge 1/2 of j to the occurrence of j in $\mathcal{A}$ and 1/2 of j to the occurrence of $f(j)$ in $\mathcal{A}$. Note that, since (I) does not apply, Lemma 4.3(2) implies that $f(j)$ is defined, and then Lemma 2.2 implies that both j and $f(j)$ are completed in $\mathcal{A}$.

(III) Otherwise, $\mathcal{A}$ is idle at t, but TightRestart is running some job k at t which is later preempted by another job h. Charge 1/2 of j to j and 1/2 to

h. By Lemma 2.2(2), j is completed in $\mathcal{A}$. Job h is urgent, and thus it is completed as well.

Analysis. We prove that each job scheduled in $\mathcal{A}$ is charged at most 3/2. Each job is charged at most 1 in case (I), as M defines a matching.

We claim that each job receives at most one charge of 1/2. For the rest of the proof, we will distinguish two types of charges of 1/2: *self-charges*, when j is charged to itself, and *non-self-charges*, when j is charged to a different job.

Suppose first that k receives a self-charge. (Obviously, it can receive only one.) Then $\mathcal{A}$ is idle at time $S_k^{\mathcal{Z}}$, for otherwise case (I) would apply to k in $\mathcal{Z}$. This implies two things. First, k is not tight, so it cannot receive a non-self-charge in case (III). Second, k cannot be in the range of $f(\cdot)$, since each job $f(j)$ is either not in $\mathcal{Z}$ or, if it is, $\mathcal{A}$ is executing some job at time $S_{f(j)}^{\mathcal{Z}}$. Therefore k cannot receive a non-self-charge in case (II).

Next, suppose that k does not receive a self-charge. Since $f(\cdot)$ is one-to-one, k can receive at most one non-self-charge in case (II). If k receives a non-self-charge in case (III) from a job j, then k is started in $\mathcal{A}$ while $\mathcal{Z}$ is executing j, so k can receive only one such charge. Finally, if k receives a non-self-charge in case (II), then by the definition of $f(\cdot)$, k is not a preemption candidate, so it cannot receive a non-self-charge in case (III).

We conclude that each job completed in $\mathcal{A}$ gets at most one charge of 1 and at most one charge of 1/2 and thus is charged a total of at most 3/2. Each job in $\mathcal{Z}$ generates a charge of 1. Thus, by summation over all jobs in $\mathcal{Z}$, we have $|\mathcal{Z}| \le 3|\mathcal{A}|/2$, completing the proof of the theorem. □

We now show that the competitive ratio of our algorithm is in fact optimal.

THEOREM 4.4. *For scheduling equal-length jobs with restarts, no deterministic algorithm is better than* 3/2*-competitive and no randomized algorithm is better than* 6/5*-competitive.*

Proof. For $p \ge 2$, consider four jobs given in the form $j = (r_j, d_j)$: $1 = (0, 3p+1)$, $2 = (1, 3p)$, $3 = (p, 2p)$, and $4 = (p+1, 2p+1)$. The instance consists of jobs 1, 2, and 3 or jobs 1, 2, and 4.

There exist schedules that schedule three jobs 1, 3, and 2 or three jobs 2, 4, and 1, in this order. (See Figure 4.3.) Therefore the optimal solution consists of three jobs.

In the deterministic case, release jobs 1 and 2. If the online algorithm starts job 2 at time 1, release job 3; otherwise, release job 4. The online algorithm completes only two jobs. As the optimal schedule has three jobs, the competitive ratio is no better than 3/2.

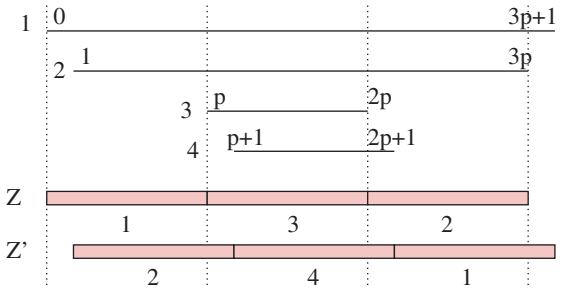


FIG. 4.3. *Jobs used in the lower bounds with restarts.*

Our proof for randomized algorithms is based on Yao's principle [24, 6]. We define a probability distribution on our two instances as follows: Always release jobs 1 and 2, and then one randomly chosen job from 3 and 4, each with probability 1/2. If $\mathcal{A}$ is any deterministic online algorithm, then the expected number of jobs completed by $\mathcal{A}$ is at most 2.5, as on one of the instances it completes only 2 jobs. Using Yao's principle, we conclude that no randomized algorithm can have a competitive ratio smaller than $3/2.5 = 6/5$. □

5. Final comments. For equal processing times, closing the gap between our upper bound of 5/3 and the lower bound of 4/3 is a challenging open problem. It would also be interesting to close these gaps for barely random algorithms which—in our view—are of their own interest (even in the case when we use only one fair random bit).

Barely random algorithms with a single random bit intuitively seem to be somewhat similar to deterministic algorithms for two machines for the same problem. In particular, one might expect that lower bounds will carry over to the problem with two machines when each job is duplicated. However, subsequent to our work, independently Ding and Zhang [10] and Goldwasser and Pedigo [16] designed 3/2-competitive deterministic algorithms for two machines. Thus, somewhat surprisingly, the answers for the two problems are different. Still, it remains a possibility that algorithms for more machines will bring some insight into the randomized scheduling on a single machine.

Beyond our simple lower bound of 6/5, nothing is known about the effect of allowing both randomness and restarts. The best upper bound of 3/2 is achieved by a deterministic algorithm. Can randomization help in the model with restarts?

Acknowledgment. We wish to express our gratitude to the anonymous referees, whose numerous and insightful suggestions helped us simplify some arguments and significantly improve the presentation of the paper.

REFERENCES

[1] S. Albers, *On randomized online scheduling*, in Proceedings of the 34th Symposium Theory of Computing (STOC), Association for Computing Machinery, New York, 2002, pp. 134–143.

[2] P. Baptiste, *Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times*, J. Sched., 2 (1999), pp. 245–252.

[3] Y. Bartal, M. Chrobak, and L. L. Larmore, *A randomized algorithm for two servers on the line*, Inform. and Comput., 158 (2000), pp. 53–69.

[4] S. K. Baruah, J. Haritsa, and N. Sharma, *On-line scheduling to maximize task completions*, in Proceedings of the 15th Real-Time Systems Symposium, IEEE Press, Piscataway, NJ, 1994, pp. 228–236.

[5] S. K. Baruah, J. Haritsa, and N. Sharma, *On-line scheduling to maximize task completions*, J. Combin. Math. Combin. Comput., 39 (2001), pp. 65–78.

[6] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, London, 1998.

[7] J. Carlier, *Problèmes d'ordonnancement à durées égales*, QUESTIO, 5 (1981), pp. 219–228.

[8] M. Chrobak, C. Dürr, W. Jawor, Ł. Kowalik, and M. Kurowski, *A note on scheduling equal-length jobs to maximize throughput*, J. Sched., 9 (2006), pp. 71–73.

[9] M. Chrobak, W. Jawor, J. Sgall, and T. Tichý, *Online scheduling of equal-length jobs: Randomization and restarts help*, in Proceedings of the 31st International Colloquium on Automata, Languages, and Programming (ICALP), Lecture Notes in Comput. Sci. 3142, Springer, Berlin, 2004, pp. 358–370.

[10] J. Ding and G. Zhang, *Online scheduling with hard deadlines on parallel machines*, in Proceedings of the 2nd International Conference on Algorithmic Aspects in Information and Management (AAIM), of Lecture Notes in Comput. Sci. 4041, Springer, Berlin, 2006, pp. 32–42.

[11] L. Epstein, J. Noga, S. S. Seiden, J. Sgall, and G. J. Woeginger, *Randomized on-line scheduling for two related machines*, J. Sched., 4 (2001), pp. 71–92.
[12] M. Garey, D. Johnson, B. Simons, and R. Tarjan, *Scheduling unit-time tasks with arbitrary release times and deadlines*, SIAM J. Comput., 10 (1981), pp. 256–269.
[13] S. A. Goldman, J. Parwatikar, and S. Suri, *Online scheduling with hard deadlines*, J. Algorithms, 34 (2000), pp. 370–389.
[14] M. H. Goldwasser, *Patience is a virtue: The effect of slack on the competitiveness for admission control*, J. Sched., 6 (2003), pp. 183–211.
[15] M. H. Goldwasser and B. Kerbikov, *Admission control with immediate notification*, J. Sched., 6 (2003), pp. 269–285.
[16] M. H. Goldwasser and M. Pedigo, *Online, non-preemptive scheduling of equal-length jobs on two identical machines*, in Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT), Lecture Notes in Comput. Sci. 4059, Springer, Berlin, 2006, pp. 113–123.
[17] H. Hoogeveen, C. N. Potts, and G. J. Woeginger, *On-line scheduling on a single machine: Maximizing the number of early jobs*, Oper. Res. Lett., 27 (2000), pp. 193–196.
[18] J. Jackson, *Scheduling a Production Line to Minimize Maximum Tardiness*, Technical report 43, Management Science Research Project, University of California, Los Angeles, 1955.
[19] B. Kalyanasundaram and K. Pruhs, *Maximizing job completions online*, J. Algorithms, 49 (2003), pp. 63–85.
[20] R. J. Lipton and A. Tomkins, *Online interval scheduling*, in Proceedings of the 5th Symposium on Discrete Algorithms (SODA), ACM/SIAM, 1994, pp. 302–311.
[21] N. Reingold, J. Westbrook, and D. D. Sleator, *Randomized competitive algorithms for the list update problem*, Algorithmica, 11 (1994), pp. 15–32.
[22] S. Seiden, *Barely random algorithms for multiprocessor scheduling*, J. Sched., 6 (2003), pp. 309–334.
[23] B. Simons, *A fast algorithm for single processor scheduling*, in Proceedings of the 19th Symposium on Foundations of Computer Science (FOCS), IEEE Press, Piscataway, NJ, 1978, pp. 246–252.
[24] A. C. C. Yao, *Probabilistic computations: Towards a unified measure of complexity*, in Proceedings of the 18th Symposium on Foundations of Computer Science (FOCS), IEEE Press, Piscataway, NJ, 1977, pp. 222–227.

SIAM J. COMPUT.
Vol. 36, No. 6, pp. 1729–1747

PHYSICAL LIMITS OF HEAT-BATH ALGORITHMIC COOLING*

LEONARD J. SCHULMAN†, TAL MOR‡, AND YOSSI WEINSTEIN‡

Abstract. Simultaneous near-certain preparation of qubits (quantum bits) in their ground states is a key hurdle in quantum computing proposals as varied as liquid-state NMR and ion traps. "Closed-system" cooling mechanisms are of limited applicability due to the need for a continual supply of ancillas for fault tolerance and to the high initial temperatures of some systems. "Open-system" mechanisms are therefore required. We describe a new, efficient initialization procedure for such open systems. With this procedure, an n-qubit device that is originally maximally mixed, but is in contact with a heat bath of bias $\varepsilon \gg 2^{-n}$, can be almost perfectly initialized. This performance is optimal due to a newly discovered threshold effect: For bias $\varepsilon \ll 2^{-n}$ no cooling procedure can, even in principle (running indefinitely without any decoherence), significantly initialize even a single qubit.

Key words. quantum computation, state preparation, thermodynamics

AMS subject classifications. 68W01, 80A99

DOI. 10.1137/050666023

1. Introduction.

Quantum computation poses a difficult experimental challenge. Simultaneous near-certain preparation of qubits (quantum bits) in their ground states is a key hurdle in proposals as varied as NMR and ion traps [8, 19, 9, 13, 10, 11]. Such "cooling" (also known as "biasing" or "polarizing") is required both for initiation of the computation [2] and in order to supply ancillas for fault tolerance as the computation proceeds.

Cooling of quantum systems has long been essential in a variety of experimental contexts unrelated to quantum computation, and is performed by processes that directly cool the system such as laser cooling in ion traps or application of strong magnetic fields in NMR. Spin exchange has also been employed in order to transfer highly cooled states into the desired system from another that is more readily directly cooled [4, 14, 24]. In all these methods, the temperature is limited by the original cooling process.

Algorithmic cooling. It is in principle possible, however, to reach even lower temperatures, by application of certain logic gates among the qubits [22]. (Even prior to quantum computation the need for signal amplification in NMR imaging led to the implementation of a basic 3-qubit logic gate [23].) In several quantum computation proposals this kind of improvement in cooling is necessary due to the requirement that a large number of qubits all be, with high probability, simultaneously in their ground states.

We distinguish between closed- and open-system algorithmic cooling methods. In the former [22] an initial phase of physical cooling is performed which reduces the

*Received by the editors March 9, 2005; accepted for publication (in revised form) October 6, 2006; published electronically March 19, 2007.
http://www.siam.org/journals/sicomp/36-6/66602.html

†California Institute of Technology, MC 256-80, Pasadena, CA 91125 (schulman@caltech.edu). The work of this author was supported in part by the NSF (PHY-0456720 and CCF-0524828), the ARO (W911NF-05-1-0294), the Mathematical Sciences Research Institute, and the Okawa Foundation.

‡Technion - Israel Institute of Technology, Haifa 32000, Israel (talmo@cs.technion.ac.il, yossiv@cs.technion.ac.il). The work of these authors was supported in part by the Israel Ministry of Defense and by the Institute for Future Defense Research at the Technion.

entropy of the system. Then in the closed phase an entropy preserving (unitary) algorithmic process is performed on the qubits. By contrast in an open process [5] some of the qubits of the system can be cooled by external interaction even during (or at interruptions in) the quantum computation. Open-system cooling places an additional experimental difficulty: Computation qubits must not decohere during the process of cooling other qubits which, at another stage, they must interact with. Nonetheless closed-system cooling appears to be insufficient for two reasons. The first applies specifically to liquid-state NMR quantum computing, where the initial entropy-reducing preparation is quite weak: the probability of the ground state of each qubit exceeds the probability of the excited state by the small factor of $e^{2\varepsilon} \approx 1+10^{-5}$. In the subsequent closed phase an ε^2 fraction of the qubits can be prepared in highly cooled states [22] (and see [23, 7] for experimental demonstrations of key steps); for information-theoretic reasons this fraction is best possible, but at the current value of ε it is too small for effective implementation of a quantum computer. The second reason applies more broadly. Any quantum computing implementation must cope with noise. Fault-tolerance mechanisms have been designed that can do so [1], if the noise level is below a specified threshold (estimated to be between 10^{-4} and 10^{-2} per qubit per operation [16]) and if a continual supply of "ancillas" (qubits which are initialized in a known state) is available. Ancilla initialization need not be perfect, but the error cannot exceed the same fault-tolerance threshold. In ion traps, for example, direct cooling can place qubits in their ground states with probability ≈ 0.95, a level that necessitates further cooling to exceed the threshold [15, 3]. Since fresh ancillas are needed in each time step, either a very large supply must be chilled in advance and maintained without substantial decoherence, or—more likely—an open-system approach must be adopted in which registers are cooled on a regular basis.

It is necessary, therefore, to study effective means for open-system algorithmic cooling. A suggested framework (called the "heat-bath" approach) was made in [5]. A heat-bath device comprises two types of qubits—some that are hard to cool (but relax slowly) and others that are readily cooled (but relax rapidly). The former are computation qubits and the latter are "refrigerants." At chosen times, the computation and refrigerant qubits can undergo joint unitary interaction (such as spin exchange). A similar framework is contemplated for ion trap quantum computers [3]—the computation ions are not cooled directly, due to the decoherence that this causes; instead they are cooled by interaction with separate refrigerant ions that have been directly laser-cooled.

Results. In this paper we establish the theoretical limits for cooling on heat-bath devices. We introduce a cooling mechanism achieving much higher bias amplification than given previously. We explicitly bound the number of cooling steps required in our amplification process, a crucial matter, since any cooling process must be carried out within the relaxation times of the computation qubits. Finally, we show that our method is optimal in terms of entropy extraction per cooling step. In the course of doing so we discover a threshold phenomenon: significant initialization cannot be achieved at all unless ε, the bias that can be imparted to the rapidly relaxing qubits, is asymptotically above 2^{-n}. The proof uses majorization inequalities to convert the problem to analysis of a certain combinatorial "chip game."

For specificity we assume that the quantum computer has $n-1$ computation qubits, and an nth refrigerant qubit that is in contact with the heat bath. The cooling step, ι, has the effect of changing the traced density matrix of the nth qubit

to

$$\rho_\varepsilon = \frac{1}{e^\varepsilon + e^{-\varepsilon}} \begin{pmatrix} e^\varepsilon & 0 \\ 0 & e^{-\varepsilon} \end{pmatrix} \tag{1.1}$$

(no matter what the previous state was). In between cooling steps, reversible (unitary) quantum logic gates can be applied to the register of n qubits. Let $\mathcal{I}_n$ be the density matrix of the maximally mixed state over the 2^n-dimensional Hilbert space ($\mathcal{I}_n = 2^{-n}\times$ the identity matrix of dimension 2^n). The question is, Starting from $\mathcal{I}_n$, and using these operations, how different from $\mathcal{I}_n$ can we make the density matrix of the device?

There is little a priori reason to expect any limit on the difference. To speak (imprecisely) in terms of temperature, we have already pointed out that the temperature of the heat bath is not a lower bound on the achievable temperature of the device, because we can use logic gates and energy to implement a heat engine (refrigerator). This being so, there is no natural lower bound on the achievable temperature short of absolute zero. It is therefore fascinating that a positive lower bound exists. The bound derives not from entropic considerations but from finite-size effects. The precise statement is not in terms of temperature but in terms of the maximum probability of any state. (For a Gibbs distribution this would be a ground state.)

THEOREM 1.1 (physical limit). *No heat-bath method can increase the probability (i.e., $|amplitude|^2$) of any basis state from its initial value, 2^{-n}, to any more than* $\min\{2^{-n}e^{\varepsilon 2^{n-1}}, 1\}$. *This conclusion holds even under the idealization that an unbounded number of cooling and logic steps can be applied without error or decoherence.*

This shows that if $\varepsilon \ll 2^{-n}$, then the variation distance between the uniform distribution, and any distribution reachable by cooling, is $\ll 1$.

On the flip side, it was shown in [12] how to produce (at small ε) a qubit of bias $(3/2)^{(n-2)/2}\varepsilon$. We improve on this result and establish a converse to Theorem 1.1, using a specific cooling procedure, the PPA, described below. For convenience let $\tilde{\varepsilon} = \tanh\varepsilon$. (For small ε, $\tilde{\varepsilon} \approx \varepsilon$.) We present the converse in two slightly incomparable forms.

THEOREM 1.2 (threshold effect). *If $\tilde{\varepsilon} \geq 2^{4-n}$, the PPA increases the variation distance from uniform to $\Theta(1)$. This occurs within $\tilde{\varepsilon}^{-2}$ cooling steps.*

THEOREM 1.3 (cold qubit extraction). *Within $4n\tilde{\varepsilon}^{-2}(1+\log(1/\tilde{\varepsilon}))$ cooling steps, the PPA creates a probability distribution in which with probability at least $1-O(\frac{1}{1+\log 1/\tilde{\varepsilon}})$, all of the first $n-(1+o(1))\log_2 1/\tilde{\varepsilon}$ bits are $|0\rangle$'s (where $o(1)$ denotes a term tending to 0 as $\tilde{\varepsilon}$ tends to 0).*

This extraction procedure is useful for quantum computing (it extracts qubits of bias almost 1, i.e., that are almost certainly in their ground state) so long as ε is above $n2^{-n}$.

The notion that the computation qubits are entirely insulated from the environment is of course merely a simplification good for moderate time spans. To be useful, algorithms must converge to the desired state within the relaxation time of the computation qubits. Next we show that the PPA is near-optimal in terms of the number of cooling steps.

THEOREM 1.4 (cooling steps required). *Any algorithm which creates a bit of constant bias requires $\Omega(\tilde{\varepsilon}^{-2})$ cooling steps.*

Finally, since the computations in the PPA vary in a complex way depending upon the value of n, we accompany the above results with another simpler cooling

procedure that applies transpositions and reversible 3-qubit majorities in a recursive pattern, and performs fairly effective cooling. This procedure is a slight modification, to achieve better asymptotics, of one given in [12]. Let $\phi = (1+\sqrt{5})/2$, let F_k be the kth Fibonacci number, and let $N = \min\{n, \lceil \log_\phi 1/\tilde{\varepsilon} \rceil\}$; the cooling algorithm $\mathcal{F}$ mentioned in the theorem is described in section 8.

THEOREM 1.5 (simple cooling algorithm). *The cooling algorithm $\mathcal{F}$ is NC1-uniform and, when run on an N-bit device, creates a bit of bias $\Omega(\tilde{\varepsilon} F_N)$ within runtime (counting both cooling steps and logic gates)* $\exp(O(N \log N))$.

Comment. The reader will have noticed that while we speak of "cooling," the algorithms are characterized not in terms of the final temperatures achieved in the qubits but in terms of other desirable properties of the final probability distributions. There are two reasons for this. The first is that other properties, especially as in Theorem 1.3, are more germane to the application to quantum computing. The second is that unambiguous assignment of a temperature to a probability distribution depends on the latter being a Gibbs distribution for some Hamiltonian describing the system; but the distributions produced by algorithmic cooling need not be Gibbs distributions. In particular, the PPA does not produce a Gibbs distribution.

Other applications of algorithmic cooling. A central point of this paper is the firm limit that Theorem 1.1 sets on the cooling parameter ε in order that the heat-bath method be useful for quantum computation. However, it is important to note that heat-bath cooling algorithms (the PPA or others) may be viable for other applications even at smaller ε. Specifically, algorithmic cooling is likely to find significant application in the scientific and medical imaging applications for which NMR technology is already in wide use. The signal-to-noise ratio in NMR imaging is proportional to the polarization of the nuclear spins and to the square root of the duration of the scan; since the duration is often limited in medicine by the need to immobilize the patient, improved sensitivity demands increased polarization. In other applications the benefit of increased polarization is in decreased scan times. Algorithmic cooling of a few nuclear spins may therefore be highly beneficial even in the range $\varepsilon \ll 2^{-n}$ that is not adequate for quantum computation. For example, perfect implementation of the PPA on a 5-qubit molecule (four computation qubits and one refrigerant) would yield a qubit of bias 8ε, implying a 64-fold decrease in scan duration compared to cooling without algorithmic amplification.

An abridged version of this paper appeared in [21].

2. Reduction of quantum to classical cooling. In preparation for the proofs of Theorems 1.1–1.5 we start with a reduction that significantly simplifies the rest of our task. Recall that the heat-bath quantum computer is assumed to start in the maximally mixed density matrix, $\mathcal{I}_n$. Any cooling step ι changes the traced density matrix on the nth qubit to the matrix given in (1.1). To see how this affects the entire density matrix, suppose that before the cooling step, the quantum computer is in a $2^n \times 2^n$ density matrix

$$M = \begin{pmatrix} M_{11} & M_{12} \\ M_{12}^\dagger & M_{22} \end{pmatrix}, \tag{2.1}$$

where the states $|0\rangle$ and $|1\rangle$ of the nth qubit partition the density matrix into these four parts. Application of ι effects the following transformation:

$$M \xrightarrow{\iota} \rho_\varepsilon \otimes (M_{11} + M_{22}) = \frac{1}{e^\varepsilon + e^{-\varepsilon}} \begin{pmatrix} e^\varepsilon (M_{11} + M_{22}) & 0 \\ 0 & e^{-\varepsilon}(M_{11} + M_{22}) \end{pmatrix}. \tag{2.2}$$

Between cooling steps, quantum logic gates can be applied to the system. These act on the density matrix as conjugations by unitary operators. If there are $r+1$ cooling steps, let these unitaries be $u_1, \ldots, u_r$. These unitary operators are constrained to be implementable by local quantum logic gates; for the limit on achievable cooling (Theorem 1.1), we may ignore this constraint and allow the unitaries to be arbitrary. For unitary u let $\overline{u}$ denote the corresponding conjugation operator.

The eigenvalues of a density matrix are the probabilities with which the spectral basis states are measured; by an inequality of Schur, the spectral basis gives measurement probabilities that are furthest from uniform, in the sense of majorization (see [18, section 9B]). A probability vector $p = (p_1, \ldots)$ is said to majorize another $p' = (p'_1, \ldots)$ (written $p \succeq p'$) if there exists a doubly stochastic matrix D such that $(p_1, \ldots)D = (p'_1, \ldots)$. This is a partial (pre-)order on probability distributions in which the singular distribution $(1, 0, 0, \ldots)$ dominates all others, while the uniform distribution is dominated by all. Schur's inequality is that the eigenvalues of a Hermitian matrix majorize its diagonal entries. A density matrix h is said to majorize another h' (written $h \succeq h'$) if the eigenvalues of h majorize those of h'.

Domination in majorization implies domination in any of the other measures we are interested in, such as variation distance from uniform, or the sum of the largest K probabilities (for a fixed K). So our concern is the following: If $\overline{u_1}, \ldots, \overline{u_r}$ represent the reversible actions of an algorithm between its cooling steps (each acting on the density matrix as conjugation by a unitary operator), how different can the eigenvalues of $\iota\, \overline{u_r}\, \iota\, \cdots\, \overline{u_1}\, \iota\, \mathcal{I}_n$ be from those of $\mathcal{I}_n$ (in which all equal 2^{-n})?

A *classical* cooling algorithm is one that uses only reversible (deterministic) classical logic gates between cooling steps. In this case each operator $\overline{u_i}$ acts on the density matrix as conjugation by a permutation matrix. Observe that a $2^n \times 2^n$ diagonal density matrix represents a probability distribution over the basis states $|0 \ldots 0\rangle, \ldots, |1 \ldots 1\rangle$.

PROPOSITION 2.1 (classical cooling). *Let h be a $2^n \times 2^n$ diagonal density matrix. Given any quantum logic steps $\overline{u_1}, \ldots, \overline{u_r}$, there are classical steps $\overline{\pi_1}, \ldots, \overline{\pi_r}$ such that $\iota\, \overline{\pi_r}\, \iota\, \cdots\, \overline{\pi_1}\, \iota\, h$ majorizes $\iota\, \overline{u_r}\, \iota\, \cdots\, \overline{u_1}\, \iota\, h$.*

For a density matrix M with eigenvectors $v_1, \ldots, v_{2^n}$ listed in decreasing order of their eigenvalues $\lambda_1 \geq \cdots \geq \lambda_{2^n}$, let w be a unitary operator which arranges the eigenvectors so that they correspond, in order, to the vectors $|0..00\rangle, |0..01\rangle, |0..10\rangle, \ldots, |1..11\rangle$ (recall that the "cooling bit" that is in contact with the reservoir is the nth or rightmost bit). Then, acting on M with $\overline{w}$, and representing the new matrix as in (2.1), it will have the diagonal entries $\lambda_1, \lambda_3, \ldots, \lambda_{2^n-1}$ in order in the upper left and the diagonal entries $\lambda_2, \lambda_4, \ldots, \lambda_{2^n}$ in order in the lower right. To prove the proposition we use the following lemma.

LEMMA 2.2. *Let M and M' be density matrices and let $M \succeq M'$. Then $\iota\overline{w}M \succeq \iota M'$.*

Proof of Proposition 2.1. Consider any sequence of conjugations $\overline{u_1}, \ldots, \overline{u_r}$. Applying the lemma, induction on r shows that

$$\iota\, \overline{w}\, \iota\, \cdots\, \overline{w}\, \iota\, h \succeq \iota\, \overline{u_r}\, \iota\, \cdots\, \overline{u_1}\, \iota\, h. \tag{2.3}$$

Observe that for each r, the left-hand side of this expression is a diagonal density matrix. Hence each application of $\overline{w}$ is a classical operation, a permutation of the basis states. ∎

Proof of Lemma 2.2. For a density matrix

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{12}^\dagger & A_{22} \end{pmatrix}, \tag{2.4}$$

let $\alpha_1, \ldots, \alpha_{2^{n-1}}$ be the eigenvalues of $A_{11} + A_{22}$. Then the eigenvalues of ιA are $\frac{e^\varepsilon}{e^\varepsilon+e^{-\varepsilon}}\alpha_1, \frac{e^{-\varepsilon}}{e^\varepsilon+e^{-\varepsilon}}\alpha_1, \ldots, \frac{e^\varepsilon}{e^\varepsilon+e^{-\varepsilon}}\alpha_{2^{n-1}}, \frac{e^{-\varepsilon}}{e^\varepsilon+e^{-\varepsilon}}\alpha_{2^{n-1}}$. It follows that if another density matrix B is given (and partitioned in the same way) and if $A_{11} + A_{22} \succeq B_{11} + B_{22}$, then $\iota A \succeq \iota B$. So it remains to show that $(\overline{w}M)_{11} + (\overline{w}M)_{22} \succeq M'_{11} + M'_{22}$.

Let $\lambda_1 \geq \cdots \geq \lambda_{2^n}$ be the eigenvalues of M and let $\lambda'_1 \geq \ldots \geq \lambda'_{2^n}$ be the eigenvalues of M'. Then $(\overline{w}M)_{11}$ is the diagonal matrix with diagonal $(\lambda_1, \lambda_3, \ldots, \lambda_{2^n-1})$, and $(\overline{w}M)_{22}$ is the diagonal matrix with diagonal $(\lambda_2, \lambda_4, \ldots, \lambda_{2^n})$. The eigenvalues of $(\overline{w}M)_{11} + (\overline{w}M)_{22}$ are $(\lambda_1 + \lambda_2, \lambda_3 + \lambda_4, \ldots, \lambda_{2^n-1} + \lambda_{2^n})$; by the assumption that $M \succeq M'$, this majorizes the sequence $(\lambda'_1 + \lambda'_2, \lambda'_3 + \lambda'_4, \ldots, \lambda'_{2^n-1} + \lambda'_{2^n})$. It remains to show that the latter majorizes the eigenvalues of $M'_{11} + M'_{22}$.

A simple inequality (see [18, section 9G]) states that the eigenvalues of $M'_{11} + M'_{22}$ are majorized by the sequence $(\beta_1 + \gamma_1, \ldots, \beta_{2^{n-1}} + \gamma_{2^{n-1}})$, where $\beta_1 \geq \cdots \geq \beta_{2^{n-1}}$ are the eigenvalues of M'_{11} and $\gamma_1 \geq \cdots \geq \gamma_{2^{n-1}}$ are the eigenvalues of M'_{22}. The argument is completed by an inequality of Fan (see [18, section 9C]) which states that for any Hermitian H,

$$\begin{pmatrix} H_{11} & H_{12} \\ H_{12}^\dagger & H_{22} \end{pmatrix} \succeq \begin{pmatrix} H_{11} & 0 \\ 0 & H_{22} \end{pmatrix}; \tag{2.5}$$

applied to $H = M'$, this yields $(\lambda'_1, \lambda'_2, \ldots, \lambda'_{2^n}) \succeq (\beta_1, \ldots, \beta_{2^{n-1}}, \gamma_1, \ldots, \gamma_{2^{n-1}})$. ∎

We may therefore restrict our attention to classical cooling algorithms. Observe that every intermediate density matrix created by a classical algorithm is diagonal. Hence the classical cooling steps are equivalent to the following discrete process on probability distributions on the set $\{0,1\}^n$: begin with the uniform distribution on $\{0,1\}^n$. The only tool for modifying the probability distribution is "discrete cooling steps," which have the effect of transforming the current distribution (denoted p) to a new distribution (denoted p'), related to p by

$$\left.\begin{aligned} p'_{w0} &= (p_{w0} + p_{w1})\tfrac{e^\varepsilon}{e^\varepsilon+e^{-\varepsilon}} \\ p'_{w1} &= (p_{w0} + p_{w1})\tfrac{e^{-\varepsilon}}{e^\varepsilon+e^{-\varepsilon}} \end{aligned}\right\} \quad \begin{array}{l}\text{for each binary string} \\ w \text{ of length } n-1.\end{array} \tag{2.6}$$

There is no way of *directly* cooling the first $n-1$ bits, but in between cooling steps we can perform arbitrary permutations of the binary strings. In the discrete process, the role of a permutation of the basis states is to properly pair off the current probabilities before the next cooling step.

Due to Proposition 2.1, Theorem 1.1 is equivalent to showing that the above discrete process cannot increase any probability from its initial value, 2^{-n}, to any more than $2^{-n}e^{\varepsilon 2^{n-1}}$, while Theorem 1.4 is equivalent to showing that the discrete process cannot create a bit of constant bias in less than $\Omega(\tilde{\varepsilon}^{-2})$ cooling steps.

3. Preliminaries.

3.1. Special configurations. The set of probabilities of the basis states, $\{P(w) : w \in \{0,1\}^n\}$, will be referred to as the configuration of the computer.

Definition 3.1. *A "special" configuration is one of*

(a) *a configuration that can be created (out of any configuration, and by any pairing) by a cooling step;*

(b) *the starting configuration, in which all probabilities equal* 2^{-n}.

Note that in a special configuration of type (a), two states that were paired in the previous round, and now have probabilities p and p', satisfy $|\log p - \log p'| = 2\varepsilon$.

3.2. The PPA. If the basis states of the computer are relabeled so that their probabilities are $p_0 \geq \cdots \geq p_{2^n-1}$ (ties broken in arbitrary but fixed fashion), then for each even i we will refer to the states i and $i+1$ as each other's "partners."

The PPA or "partner-pairing algorithm" is simply the following process: In each cooling step, pair partners together. (This completely specifies the algorithm save only for the number of iterations.)

LEMMA 3.2. *In a special configuration, if states with probabilities p and p' are partners, then* $|\log p - \log p'| \leq 2\varepsilon$.

Proof. For the configuration of type (b) this is automatic; for those of type (a) let p be the probability of a state for which the lemma is violated and let q be the probability of the state with which it was paired in the previous round. Suppose $q > p$; the other case is similar. So p is now paired with a probability r for some $r < pe^{-2\varepsilon}$, and the interval (r, p) is empty of state probabilities. The interval $(-\infty, r]$ therefore contains only intact pairs from the previous round and hence an even number of state probabilities. So it cannot be that p's partner in this round is r. ∎

The next step in demonstrating Theorems 1.1 and 1.4 concerns the relation between the output of an arbitrary cooling algorithm B and that of the PPA.

COROLLARY 3.3. *Given any initial probability distribution* $p = \{p_0, \ldots, p_{2^n-1}\}$, *and any cooling algorithm B, the distribution which results from applying the PPA for r cooling steps majorizes the distribution which results from applying B for r cooling steps.*

Proof. This follows from Proposition 2.1 because the PPA is the restriction to probability distributions of the operator $\overline{w}$ defined in section 2. ∎

As a consequence, in pursuit of a bound on the maximum achievable probability of any one string, we can focus on the PPA. (The same lesson applies to any Schur-convex function of the probabilities, of which the maximum probability is but one example; see [18].)

4. Proof of Theorem 1.1.

4.1. Dynamics of cooling algorithms: Assemblies of chips. It is useful to apply the map $p \to \log(2^n p)$ to all the probabilities of a configuration, to obtain a set of 2^n "chips" arrayed on the real axis. Two chips at z_1 and z_2 which are paired by a cooling step are carried to two new chips at $T(z_1, z_2) \pm \varepsilon$, where T is given by

$$e^{T(z_1,z_2)+\varepsilon} + e^{T(z_1,z_2)-\varepsilon} = e^{z_1} + e^{z_2}. \tag{4.1}$$

We now need to understand more about the dynamics of the PPA. A central tool will be to designate certain subsets of the chips as *assemblies.* With an assembly S we associate a *center* $c(S)$ which is the arithmetic mean of the chips, a *radius* $r(S)$ which is $\varepsilon/2$ times the number of chips in the assembly, and an *interval* I_S which is the closed interval $[c(S) - r(S), c(S) + r(S)]$. (We define assemblies only for special configurations.) A set of chips qualifies as an assembly if either

1. it is a pair of chips z_1 and z_2 which are partners (note that the center of this assembly is $(z_1 + z_2)/2$ and its radius is ε);
2. it is the union of two assemblies whose intervals intersect (we will refer to this as merging the two assemblies).

A maximal assembly is one which cannot be merged with any other assembly.

4.2. The modified PPA. The nonlinear map T (defined in (4.1)) is difficult to work with directly, but it has a linearization which suits our needs. In the modified process, chips at z_1 and z_2 are carried to the pair $M(z_1, z_2) \pm \varepsilon$, where

$$M(z_1, z_2) = (z_1 + z_2)/2. \tag{4.2}$$

(The modified process does not preserve the identity $2^{-n} \sum e^{z_i} = 1$.) In the modified PPA, partners are defined among the chips just as before, but the map M rather than the map T is applied to each pair. That the modified process is a useful approximation to the true process is due to the twin facts that ε is small and that in a special configuration, partners z_1 and z_2 are close. The bearing of the modified process on the true process is expressed in the following lemma.

LEMMA 4.1. *Consider two sets of chips in special configurations $x_0 \geq \cdots \geq x_{2^n-1}$ and $y_0 \geq \cdots \geq y_{2^n-1}$, such that $x_i \leq y_i$ for all i. Apply a step of the true PPA to $x_0, \ldots, x_{2^n-1}$, resulting in the set of chips $x'_0 \geq \cdots \geq x'_{2^n-1}$. Apply a step of the modified PPA to $y_0, \ldots, y_{2^n-1}$, resulting in the set of chips $y'_0 \geq \cdots \geq y'_{2^n-1}$. Then $x'_i \leq y'_i$ for all i.*

Proof. We have only to show that for any even i, $T(x_i, x_{i+1}) \leq M(y_i, y_{i+1})$. We have that

$$e^{T(x_i,x_{i+1})} = e^{M(x_i,x_{i+1})} \frac{\cosh((x_i - x_{i+1})/2)}{\cosh(\varepsilon)}. \tag{4.3}$$

Since the configuration $x_1, \ldots, x_{2^n}$ is special, $|x_i - x_{i+1}| \leq 2\varepsilon$ by Lemma 3.2, and so $T(x_i, x_{i+1}) \leq M(x_i, x_{i+1})$. Since $M(x_i, x_{i+1}) \leq M(y_i, y_{i+1})$ directly from the assumptions, we conclude that $T(x_i, x_{i+1}) \leq M(y_i, y_{i+1})$. ∎

Applying each of the processes T and M repeatedly starting from a common special configuration, we conclude by induction that after any number of iterations, the greatest achievable probability of any state in the modified process is an upper bound on the probability of any state in the true process.

The chip game. The modified process given by (4.2) describes the following chip game: 2^n chips are placed initially at the origin of the real line. In each step you choose a pairing of the chips, and then the positions of each pair of chips (say z_1 and z_2) are moved to $(z_1 + z_2)/2 \pm \varepsilon$. Your goal is to move any one chip as far to the right as possible. Theorem 1.1 has been reduced to showing that no chip can be moved to distance more than $\varepsilon 2^{n-1}$ from the origin. Section 4.3 is devoted to a somewhat lengthy combinatorial proof of this fact.

Fortunately, there is a simpler proof of a bound that is weaker by a factor of 2: i.e., no chip can be moved to distance more than $\varepsilon 2^n$ from the origin. This is sufficient to establish our fundamental physical conclusions—to wit: unbounded cooling is impossible using finitely many computation qubits at a fixed heat-bath temperature; moreover, for large n there is a threshold at $(-\log_2 \tilde{\varepsilon})/n = 1$ for the feasibility of cooling. The reader interested only in these conclusions can read the following and skip section 4.3.

The bound of $\varepsilon 2^n$ rests on showing that the modified PPA, starting from the initial configuration having all chips at the origin, never creates a separation of more than 2ε between adjacent chips: Suppose the gaps within a set of chips $x_0 \geq \cdots \geq x_{2^n-1}$ are bounded by 2ε, and that a step of the modified PPA is applied to these chips, carrying x_{2i} to $x'_{2i} = (x_{2i} + x_{2i+1})/2 + \varepsilon$, and x_{2i+1} to $x'_{2i+1} = (x_{2i} + x_{2i+1})/2 - \varepsilon$. Note that for even i, $x'_i \geq x_i$, while for odd i, $x'_i \leq x_i$. The $\{x'_i\}$ are generally not in sorted order, but x'_{2^n-1} is a smallest chip, and so it is enough to show that for every

$i < 2^n - 1$, $x'_{i+1} \geq x'_i - 2\varepsilon$ (i.e., there are no descents by more than 2ε in the sequence $x'_0, \ldots, x'_{2^n-1}$). For even i, $x'_{i+1} \geq x'_i - 2\varepsilon$ is satisfied with equality. For odd i, using the inductive hypothesis, $x'_{i+1} \geq x_{i+1} \geq x_i - 2\varepsilon \geq x'_i - 2\varepsilon$.

Finally, a configuration of chips whose mean is 0 and in which all gaps are bounded by 2ε has no chip beyond distance $\varepsilon 2^n$ from the origin. For if any gap is less than 2ε, the configuration does not achieve greatest possible distance, since the chips to the right and left of this gap can be shifted outward while preserving the mean, while a configuration in which all the gaps are exactly 2ε is an arithmetic sequence centered at the origin.

We return to the proof of the full statement of Theorem 1.1.

4.3. Preservation of maximal assemblies. The most lengthy technical portion of this paper goes into establishing the following proposition.

PROPOSITION 4.2.

1. *The maximal assemblies of a special configuration partition the set of chips. (Equivalently, we can arrive at the list of all maximal assemblies by merging assemblies in any order until no further mergers are possible.)*
2. *Maximal assemblies are preserved by the modified PPA (i.e., the partition of the chips of a special configuration into maximal assemblies is unchanged by a cooling step).*

We begin with a sequence of arguments that do not depend on whether the true or modified PPA is applied but only on the fact that each step pairs partners together.

LEMMA 4.3. *In a special configuration, if $a, b \in \mathbb{R}$, $a \leq b$, and the intervals $[a - 2\varepsilon, a)$ and $(b, b + 2\varepsilon]$ are empty of chips, then the interval $[a, b]$ contains an even number of chips.*

Proof. In a special configuration, two chips that were paired in the previous round are separated by 2ε. The fact that $[a - 2\varepsilon, a)$ is empty of chips therefore implies that there are an even number of chips in $(-\infty, a)$; similar reasoning shows there are an even number of chips in (b, ∞). □

LEMMA 4.4. *Let $k \geq 0$, k even, and let D be an assembly of cardinality at most k.*

1. *(Bounded gap). If $S \subseteq D$ consists of some of the partner pairs of D, then $I_S \cap I_{D-S} \neq \emptyset$.*
2. *(Monotonicity). If an assembly B is a subset of D, then I_B is contained in I_D.*

Proof. The proof is by induction on k.

Part 1, Bounded gap: Let $P_1, \ldots, P_{k/2}$ be the partner pairs of D, listed in their order on the line. Suppose the lemma fails for $S = P_1 \cup \cdots \cup P_\ell$, for some $0 < \ell < k/2$. Fix a sequence of mergers that forms D out of $P_1, \ldots, P_{k/2}$. We may assume these mergers always combine adjacent assemblies, since if an assembly B is between A and C which are being merged, I_B must intersect one of I_A or I_C (say I_A); B can be merged with A. By part 2 of the lemma (for $k - 2$), all subsequent merger steps which are supposed to be performed with the assembly containing A or B can still be performed (in particular the very next step of merging $A \cup B$ with C). So the mergers describe a binary tree T_0 of assemblies, whose leaves are the partner pairs and whose internal nodes are the assemblies constructed during the merging process; the left and right children of any internal node are always two disjoint assemblies which are adjacent to each other in the left-right order on the line. Moreover, the children of any internal node are two disjoint assemblies whose intervals intersect, since this is a tree of mergers. The root of T_0 is D.

We will also use other trees in the proof. The internal nodes of these trees might not be assemblies, but each internal node will still be a sequence of partner pairs that are laid out consecutively on the line; the left and right children of an internal node will still consist of two sequences which are disjoint, adjacent to each other on the line, and in the same left-right order. While the set at an internal node may not be an assembly, we will still associate with such a set of partner pairs a center, a radius, and an interval, all defined just as they are for an assembly.

We will say that an internal node is "cohesive" if the intervals of its two children intersect. Every internal node in T_0 is cohesive. The claim will follow from the existence of another tree T_F in which the left child of the root is S, the right child is $D - S$, and the root is cohesive.

We will show the existence of T_F by converting T_0 into it through a sequence of "tree rotations." In a tree rotation a tree T' is changed into a tree T'' as follows. Let A, B, and C each be a sequence of consecutive partner pairs, and let these sequences be disjoint, and arranged adjacent to each other on the line from left to right in the order A, B, C. Suppose that each occurs as a node in T' and that there are internal nodes $A \cup B$ and $A \cup B \cup C$. Then a right tree rotation "at $A \cup B \cup C$" is the conversion of T' into the tree T'' that differs only in that instead of an internal node $A \cup B$, it has an internal node $B \cup C$. (A left tree rotation would be the replacement of a node $B \cup C$ by a node $A \cup B$.) We will demonstrate the following property of right tree rotations; the analogous property holds for left tree rotations and is shown in the same way.

(*) If $A \cup B$ and $A \cup B \cup C$ are cohesive in T', then $A \cup B \cup C$ is cohesive in T''.

Using (*) we will obtain the desired tree T_F by beginning with T_0, in which all internal nodes are cohesive, and repeatedly doing the following: Find the least common ancestor J of P_ℓ and $P_{\ell+1}$, let K be its parent, and rotate at K. After the rotation, K becomes the new least common ancestor of P_ℓ and $P_{\ell+1}$; by (*), it is still cohesive. The cohesiveness of nodes outside the subtree rooted at K is unaffected by the rotation. Hence the process continues until a last rotation at the root, at which time the root is cohesive, and is the least common ancestor of P_ℓ and $P_{\ell+1}$.

Finally, we show (*). For simplicity of notation and without loss of generality we will assume the center of B is 0. Let A have center $-r_1$ and radius s_1; let B have radius s_2; and let C have center r_3 and radius s_3. Note $s_1, s_2, s_3, r_1, r_3 \geq 0$. Cohesiveness of $A \cup B$ in T' means that

$$r_1 \leq s_1 + s_2, \tag{4.4}$$

while cohesiveness of $A \cup B \cup C$ in T' means that

$$r_3 + \frac{r_1 s_1}{s_1 + s_2} \leq s_1 + s_2 + s_3. \tag{4.5}$$

Sum these inequalities with the respective nonnegative coefficients $\frac{s_2(s_1+s_2+s_3)}{(s_1+s_2)(s_2+s_3)}$ and $\frac{s_3}{(s_1+s_2)(s_2+s_3)}$ to obtain

$$r_1 + \frac{r_3 s_3}{s_2 + s_3} \leq s_1 + s_2 + s_3, \tag{4.6}$$

which indicates the cohesiveness of $A \cup B \cup C$ in T''.

Part 2, Monotonicity: The proof is in two sections. (a) We argue that we can form D in a sequence of strict mergers that create B as an intermediate step. (A

strict merger is one that forms the union of two assemblies neither of which contains the other.) (b) We argue that in any strict merger, forming assembly $C = B_1 \cup B_2$ from B_1 and B_2, the interval of C contains those of B_1 and B_2.

Proof of (a). First, carry out the mergers that create B from the original pairs. Now consider the mergers that create D from the original pairs. Carry out those steps, each time replacing the arguments E_1 and E_2 of a desired merger $E_1 \cup E_2$, with the present (greatest) assemblies that contain E_1 and E_2. We must check that this makes sense: that each of E_1 and E_2 are a subset of a present assembly, and that the intervals of those assemblies intersect. The latter claim holds by induction because, until D has been formed, those assemblies are smaller than D (and because after D has been formed, all mergers are trivial). To see the former claim, observe (by induction on the step number) that at any time, the present greatest assembly containing E_i is either E_i, or $E_i \cup B$, depending on whether any of the pairs in E_i intersects B.

Proof of (b). Let $s = |B_1 \cap B_2|$, $s_1 = |B_1 - B_2|$, and $s_2 = |B_2 - B_1|$. Let c be the arithmetic mean of $B_1 \cap B_2$, c_1 the arithmetic mean of $B_1 - B_2$, and c_2 the arithmetic mean of $B_2 - B_1$.

The arithmetic mean of B_1 is $\overline{c}_1 = (cs + c_1 s_1)/(s + s_1)$, and the arithmetic mean of B_2 is $\overline{c}_2 = (cs + c_2 s_2)/(s + s_2)$. Let $\overline{c}$ be the arithmetic mean of $B_2 \cup B_1$, so $\overline{c} = (cs + c_1 s_1 + c_2 s_2)/(s + s_1 + s_2)$. To demonstrate the containment of intervals, we show that the left-hand boundary of I_C, $\overline{c} - s - s_1 - s_2$, is to the left of the left-hand boundary of I_{B_1}, $\overline{c}_1 - s - s_1$; in other words, $\overline{c}_1 - s - s_1 \geq \overline{c} - s - s_1 - s_2$. The remaining three cases are similar.

By part 1 of the lemma we know

$$s + s_2 \geq c_2 - c, \tag{4.7}$$

$$s + s_1 \geq c - c_1. \tag{4.8}$$

Inequality (4.7) is equivalent to $c_2 - \overline{c}_1 \leq s + s_2 + c - \overline{c}_1$. Inequality (4.8) is equivalent to $c - \overline{c}_1 \leq s_1$. Together these give

$$c_2 - \overline{c}_1 \leq s + s_1 + s_2, \tag{4.9}$$

which is equivalent to

$$\overline{c} - \overline{c}_1 \leq s_2. \tag{4.10}$$ □

We now prove Proposition 4.2(1): *The maximal assemblies of a special configuration partition the set of chips. (Equivalently, we can arrive at the list of all maximal assemblies by merging assemblies in any order until no further mergers are possible.)*

Proof. The initial pairing of chips is fixed. Let $S_1, \ldots, S_k$ be the maximal assemblies obtained by a particular sequence of mergers. Write, in terms of the initial pairs, $S_1 = P_{11} \cup \cdots \cup P_{1\ell_1}$, $S_2 = P_{21} \cup \cdots \cup P_{2\ell_1}$, and so forth. Fixing an alternate merger sequence, consider the first step in which that sequence joins pairs from some two different S_i's; suppose those are S_1 and S_2. Let $S_1' \subseteq S_1$ and $S_2' \subseteq S_2$ be the two assemblies merged in this step. Then the intervals of S_1' and S_2' intersect, which contradicts Lemma 4.4(2), since the intervals of S_1 and S_2 do not intersect. □

COROLLARY 4.5. *Let D be an assembly and let $S \subseteq D$ be a set of even cardinality. Then I_S is contained in I_D.*

Proof. We show that the right end of I_S is less than the right end of I_D; a similar argument shows that the left end of I_S is greater than the left end of I_D. The set S' consisting of the rightmost $|S|$ chips in D consists of several partner pairs. By Lemma 4.4(1), $I_{S'}$ intersects $I_{D-S'}$; this implies that $I_{S'} \subseteq I_D$. ∎

LEMMA 4.6. *Given a cooling step, form a corresponding set of intervals S as follows. For each two chips paired by the cooling step, S contains the interval between the poststep positions of those chips. Also, for each pair of partners in the poststep configuration, S contains the interval between the partners. Consider any point that coincides with no poststep chips. Then there is an even number of intervals of S containing that point.*

Proof. Moving from left to right, at every chip the number of partner intervals covering the line alternates between 0 and 1. The parity of the contribution of the pair intervals also alternates at every chip. Therefore, between chips, the parity is the same as it is beyond the last chip: 0. ∎

LEMMA 4.7. *Two chips which are paired in a cooling step (of any algorithm) are in a common maximal assembly after that cooling step.*

Proof. For specificity suppose this step was numbered t. Let x be such that the positions of the two chips after step t are $x \pm \varepsilon$. We will use the terms "righties" and "lefties" to refer to members of pairs depending on whether they are, respectively, the higher or lower probability chip (after the step); e.g., $x + \varepsilon$ is the righty (or t-righty, to specify the step) and $x - \varepsilon$ is the t-lefty of their pair.

We consider several cases.

Case 1. $x \pm \varepsilon$ are partners poststep. The lemma follows.

Otherwise, for $-\varepsilon \le s_1, s_2 \le \varepsilon$, let $x - \varepsilon$ be partnered with a chip at $x - \varepsilon + 2s_1$ and let $x+\varepsilon$ be partnered with a chip at $x+\varepsilon+2s_2$. The number of chips, m, between the pairs $\{x-\varepsilon, x-\varepsilon+2s_1\}$ and $\{x+\varepsilon, x+\varepsilon+2s_2\}$ is even; we consider several cases.

Case 2. $s_1 \ge s_2$. In this case the assembly $\{x-\varepsilon, x-\varepsilon+2s_1\}$ (whose right-hand boundary is at $x+s_1$) and the assembly $\{x+\varepsilon, x+\varepsilon+2s_2\}$ (whose left-hand boundary is at $x + s_2$) intersect geometrically, and the lemma follows.

Case 3. $s_1 < s_2$. We start by showing that $m > 0$, which is to say that the interval $[x-\varepsilon+2s_1, x+\varepsilon+2s_2]$ contains chips other than $x-\varepsilon$ or $x+\varepsilon$. The interval between $\max\{x-\varepsilon, x-\varepsilon+2s_1\}$ and $\min\{x+\varepsilon, x+\varepsilon+2s_2\}$ is nonempty, and since it is contained in $[x-\varepsilon, x+\varepsilon]$, it must by Lemma 4.6 intersect some interval between two chips that were paired in the last cooling step; neither $x-\varepsilon+2s_1$ nor $x+\varepsilon+2s_2$ can be one of the chips generating such an interval, since the distance between them is greater than 2ε. Hence m is positive; we continue with two cases depending on its value.

Case 3a. $s_1 < s_2$ and $m = 2$. Let the two points be z_1 and z_2, with $z_1 \le z_2$; note that these are paired together poststep and that $x-\varepsilon \le z_1 \le z_2 \le x+\varepsilon$. By the parity argument of Lemma 4.6, there must be two chips that were paired in step t, for which the interval between the poststep chip positions covers the interval between $x-\varepsilon+2s_1$ and z_1; therefore $z_1 \le x+\varepsilon+2s_1$. For a similar reason, $z_2 \ge x-\varepsilon+2s_2$. We examine three pair assemblies: $A = \{x-\varepsilon, x-\varepsilon+2s_1\}$, $B = \{z_1, z_2\}$, and $C = \{x+\varepsilon, x+\varepsilon+2s_2\}$. Their intervals are $I_A = [x-2\varepsilon+s_1, x+s_1]$, $I_B = [(z_1+z_2)/2-\varepsilon, (z_1+z_2)/2+\varepsilon]$, and $I_C = [x+s_2, x+2\varepsilon+s_2]$. If I_B does not intersect I_A, then $x+s_1+\varepsilon < (z_1+z_2)/2$. If in addition I_B does not intersect I_C, then $(z_1+z_2)/2 < x+s_2-\varepsilon$, together implying $s_1+2\varepsilon < s_2$, which is impossible, since $-\varepsilon \le s_1, s_2 \le \varepsilon$. Hence I_B intersects at least one of I_A or I_C. The rest of the argument is symmetric for these two cases, and so we spell out only the case that I_B intersects I_A.

If I_B intersects I_A, the four points of A and B form an assembly $A \cup B$ whose right-hand boundary is at $(2(x-\varepsilon)+2s_1+z_1+z_2)/4+2\varepsilon$ which, by the lower bounds for z_1 and z_2, is at least $x+\varepsilon+(s_1+s_2)/2$. Subtracting the left-hand boundary of I_C gives $\varepsilon+(s_1-s_2)/2$, which by the constraints on s_1 and s_2 is at least 0. Hence the interval of the assembly $A \cup B$ intersects that of the assembly C, and the lemma follows.

Case 3b. $s_1 < s_2$ and $m \geq 4$. In this case there are at least four points $z_1, \ldots, z_m$ arranged as $x-\varepsilon \leq z_1 \leq \cdots \leq z_m \leq x+\varepsilon$; the same argument used for Case 3a shows that either the assembly of $\{z_1, z_2\}$ intersects that of $\{x-\varepsilon, x-\varepsilon+2s_1\}$, or the assembly of $\{z_{m-1}, z_m\}$ intersects that of $\{x+\varepsilon, x+\varepsilon+2s_2\}$. The cases are symmetric, and so suppose that the first of these occurs. Then the assembly formed by $D = \{z_1, z_2, x-\varepsilon, x-\varepsilon+2s_1\}$ has its right-hand boundary at $(2(x-\varepsilon)+2s_1+z_1+z_2)/4+2\varepsilon$. Using the lower bound $x-\varepsilon$ for z_1 and z_2 places a lower bound of $x+\varepsilon+s_1/2$ on this boundary. For the interval of the assembly $\{z_{m-1}, z_m\}$ not to intersect this, we must have $(z_{m-1}+z_m)/2 > x+s_1/2+2\varepsilon$. The right-hand boundary of the assembly $\{z_{m-1}, z_m\}$ must therefore be at a position greater than $x+s_1/2+3\varepsilon$, which in turn is at least $x+5\varepsilon/2$. Hence the four points $E = \{z_{m-1}, z_m, x+\varepsilon, x+\varepsilon+2s_2\}$ form an assembly. Using the upper bound $x+\varepsilon$ on z_{m-1} and z_m places an upper bound of $x-\varepsilon+s_2/2$ on the left-hand boundary of this assembly. The intervals of the assemblies D and E intersect because $(x+\varepsilon+s_1/2)-(x-\varepsilon+s_2/2) = 2\varepsilon+(s_1-s_2)/2 \geq \varepsilon \geq 0$. The lemma follows. ∎

We can now finally prove Proposition 4.2(2): *Maximal assemblies are preserved by the modified PPA.*

We show, equivalently, that every prestep assembly is contained in a poststep assembly. Lemma 4.7 establishes this for prestep pairs. Now suppose that the prestep assembly D was formed by merging assemblies B and C. By induction B and C are each contained in a poststep assembly; call these B' and C'. By Corollary 4.5, the intervals of B' and C' contain those of the poststep sets of chips B and C. In the modified chip process, these last two intervals are identical, respectively, to the intervals of the prestep assemblies B and C. Therefore $I_B \subseteq I_{B'}$ and $I_C \subseteq I_{C'}$. Since I_B intersects I_C, $B' \cup C'$ is an assembly, and it contains D. ∎

Proof of Theorem 1.1. Proposition 4.2 implies that in a configuration reachable from the start state by the modified chip process there is just a single maximal assembly, whose interval is $[-\varepsilon 2^{n-1}, \varepsilon 2^{n-1}]$. Consider a chip that is furthest from the origin: by Lemma 3.2, it lies within the interval of the assembly formed by itself and its partner; by Lemma 4.4(2), this interval is contained within the interval of the maximum assembly. Hence all chips lie within distance $\varepsilon 2^{n-1}$ of the origin. Due to Corollary 3.3 (applied to the initial uniform distribution which corresponds to the maximally mixed state $\mathcal{I}_n$) and Lemma 4.1, this shows that no cooling process can increase any probability above $2^{-n}e^{\varepsilon 2^{n-1}}$, establishing the theorem. ∎

5. Proof of Theorem 1.2. Here we prove Theorem 1.2, which is a complement to the "impossibility" result of Theorem 1.1: *For $\tilde{\varepsilon} \geq 2^{4-n}$, heat-bath cooling using the PPA produces a distribution at variation distance $\Theta(1)$ from uniform, within $T = \tilde{\varepsilon}^{-2}$ cooling steps.* To a state with probability p assign the potential $g(p) = \log\cosh(2^n(p-2^{-n}))$, and to a configuration c assign the potential $g(c) = \sum_p g(p)$. Observe that $g(\text{initial configuration}) = 0$.

Let c be any special configuration and let c' be the configuration it is carried to by the PPA. Let $\Delta g(c) = g(c') - g(c)$. If p_1 and p_2 are paired in c, then their

contribution to $\Delta g(c)$ is

$$g(p_1') + g(p_2') - g(p_1) - g(p_2), \tag{5.1}$$

where without loss of generality $p_1 \le p_2$, and we have written $p_1' = (p_1+p_2)(1-\tilde{\varepsilon})/2$ and $p_2' = (p_1+p_2)(1+\tilde{\varepsilon})/2$. This is nonnegative because g is convex, $p_1+p_2 = p_1'+p_2'$, and because due to Lemma 3.2, $[p_1,p_2] \subseteq [p_1',p_2']$.

Since g is strictly convex, the potential of a special configuration increases strictly unless each of its pairs $\{p_1,p_2\}$ satisfies $|\log p_2 - \log p_1| = 2\varepsilon$.

If sometime within T rounds it happens that there are at least $2^{n-1}-2$ probabilities outside of the interval $[2^{-n-1}, 3\cdot 2^{-n-1}]$, then we are done.

Otherwise, suppose that c is a special configuration having at least $2^{n-1}+2$ probabilities within the interval $[2^{-n-1}, 3\cdot 2^{-n-1}]$. We want to show a lower bound on $\Delta g(c)$. The PPA must form at least 2^{n-2} pairs among these probabilities , and at least 2^{n-3} of those pairs must be of length (separation between the probabilities) at most 2^{3-2n}. The contribution of such a pair to $\Delta g(c)$ is least if its length is indeed 2^{3-2n}; for a lower bound on $\Delta g(c)$ we also assume that the poststep probabilities are as close to each other as possible, which (since their ratio is fixed at $e^{2\varepsilon}$) occurs when the average of the probabilities is as small as possible, namely $2^{-n-1}\cdot 2\tilde{\varepsilon} = 2^{-n}\tilde{\varepsilon}$. Letting the probabilities of the pair, before the cooling step, be $y \pm 2^{2-2n}$, and letting Δ_1 be the contribution of these two probabilities to $\Delta g(c)$, we can write

$$\Delta_1 = \log \frac{\cosh 2^n(y(1+\tilde{\varepsilon}) - 2^{-n}) \cosh 2^n(y(1-\tilde{\varepsilon}) - 2^{-n})}{\cosh 2^n(y + 2^{2-2n} - 2^{-n}) \cosh 2^n(y - 2^{2-2n} - 2^{-n})}. \tag{5.2}$$

Let $x = 2^n y - 1$; note that $|x| \le 1/2$. Let $\eta = 2^n y\tilde{\varepsilon}$; since $y \ge 2^{-n-1}$, $\eta \ge \tilde{\varepsilon}/2$.

$$\Delta_1 = \log \frac{\cosh(x+\eta)\cosh(x-\eta)}{\cosh(x+2^{2-n})\cosh(x-2^{2-n})} = \log \frac{\cosh 2x + \cosh 2\eta}{\cosh 2x + \cosh 2^{3-n}}. \tag{5.3}$$

Since this is increasing in η for $\eta > 0$, we have

$$\Delta_1 \ge \log \frac{\cosh 2x + \cosh \tilde{\varepsilon}}{\cosh 2x + \cosh 2^{3-n}}. \tag{5.4}$$

Now let $h(z) = \log \frac{\cosh 2x + \cosh(\tilde{\varepsilon}/2+z)}{\cosh 2x + \cosh 2^{3-n}}$. Then $\Delta_1 \ge h(\tilde{\varepsilon}/2)$. Now

$$h'(z) = \frac{\sinh(\tilde{\varepsilon}/2+z)}{\cosh 2x + \cosh(\tilde{\varepsilon}/2+z)}, \tag{5.5}$$

$$h''(z) = \frac{1+\cosh(\tilde{\varepsilon}/2+z)\cosh 2x}{(\cosh 2x + \cosh(\tilde{\varepsilon}/2+z))^2} \ge 0. \tag{5.6}$$

Thus $h(z) \ge h(0) + zh'(0)$, and in particular, since $\tilde{\varepsilon}/2 \ge 2^{3-n}$, $h(0) \ge 0$, and

$$\Delta_1 \ge h\left(\frac{\tilde{\varepsilon}}{2}\right) \ge 0 + \frac{\tilde{\varepsilon}}{2}\frac{\sinh(\tilde{\varepsilon}/2)}{\cosh 2x + \cosh(\tilde{\varepsilon}/2)} \ge \frac{\tilde{\varepsilon}^2}{8\cosh 1}, \tag{5.7}$$

the last inequality being implied by $|x| \le 1/2$, $\tilde{\varepsilon} \le 1$, and $\sinh(\tilde{\varepsilon}/2) \ge \tilde{\varepsilon}/2$. Consequently, if for T rounds it does not occur that at least $2^{n-1}-2$ probabilities are

outside of the interval $[2^{-n-1}, 3 \cdot 2^{-n-1}]$, then due to the 2^{n-3} pairs to which this analysis applies, g increases to at least

$$\frac{T 2^{n-6} \tilde{\varepsilon}^2}{\cosh 1}. \tag{5.8}$$

Observe now that for any p, $|p - 2^{-n}| \geq g(p) 2^{-n}$. So the variation distance from uniform after $T = \tilde{\varepsilon}^{-2}$ steps rises to at least

$$\frac{2^{-6}}{\cosh 1}. \tag{5.9}$$ □

6. Proof of Theorem 1.3. Here we prove Theorem 1.3, the second form (and the one more directly relevant to quantum computation) of the complement to the "impossibility" result of Theorem 1.1. *Within* $4n\tilde{\varepsilon}^{-2}(1 + \log(1/\tilde{\varepsilon}))$ *cooling steps, the PPA creates a probability distribution in which with probability at least* $1 - O(\frac{1}{1+\log 1/\tilde{\varepsilon}})$, *all of the first* $n - (1 + o(1)) \log_2 1/\tilde{\varepsilon}$ *bits are* $|0\rangle$*'s.*

Proof. As in the previous section we use a potential function, but now we use a different function—the entropy of the distribution—and we use it only for the runtime analysis, rather than using low entropy to imply that many cold bits are extracted.

Let $\mathcal{H}$ be the entropy function, and for $0 \leq \delta \leq 1$ let $H(\delta) = \mathcal{H}(\{(1-\delta)/2, (1+\delta)/2\}) = \frac{1-\delta}{2} \log \frac{2}{1-\delta} + \frac{1+\delta}{2} \log \frac{2}{1+\delta}$. Let $(1 \pm \delta)p/2$ be two probabilities paired in a cooling step. The change in their contribution to the distribution entropy due to the cooling step is $(H(\tilde{\varepsilon}) - H(\delta))p$; due to Lemma 3.2, $\delta \leq \tilde{\varepsilon}$, and so this contribution is nonpositive. Thus the distribution entropy is weakly decreasing in each cooling step.

LEMMA 6.1. *Within* $\frac{n \log 2}{(H(\delta) - H(\tilde{\varepsilon}))\gamma}$ *cooling steps, at least* $1 - \gamma$ *of the probability resides in partners* $\{p_1, p_2\}$ *for which* $|\log p_1 - \log p_2| \geq 2\delta$.

Proof. So long as the condition is unfulfilled, at least γ of the probability resides in partners for which $|\log p_1 - \log p_2| \leq 2\delta$, and so the distribution entropy (which begins as $n \log 2$) decreases in each cooling step by at least $(H(\delta) - H(\tilde{\varepsilon}))\gamma$. □

LEMMA 6.2. *If at least* $1 - \gamma$ *of the probability resides in partners* $\{p_1, p_2\}$ *for which* $|\log p_1 - \log p_2| \geq 2\delta$, *then for positive even* y, *at least* $(1-\gamma)(1 - e^{-(y+2)\delta})$ *of the probability resides in just* y *of the states.*

Proof. The probability of the y most likely states is at least equal to the probability of the y most likely states in partner pairs for which $|\log p_1 - \log p_2| \geq 2\delta$. That probability is maximized by the distribution in which the partners pairs are adjacent, which is to say that each probability occurs twice (except at the ends), once as the smaller and once as the larger of two partners. A short calculation shows that the sum of the top y probabilities is at least $(1-\gamma)(1 - e^{-(y+2)\delta})$. □

Finally, we can establish Theorem 1.3. Let $\gamma = \frac{\log 2}{1+\log 1/\tilde{\varepsilon}}$, $y = \frac{2 \log 1/\gamma}{\tilde{\varepsilon}}$, and $\delta = \tilde{\varepsilon}/2$. The total probability of these y most likely states is $1 - O(\frac{1}{1+\log 1/\tilde{\varepsilon}})$, and once indexed lexicographically in decreasing likelihood from 0 to $2^n - 1$, they all share $|0\rangle$'s in their first $n - \lg y \geq n - (1 + o(1)) \lg 1/\tilde{\varepsilon}$ bits. □

7. Proof of Theorem 1.4. We demonstrate here the lower bound of $\Omega(\tilde{\varepsilon}^{-2})$ on the number of cooling steps required in order to create even a single bit of constant bias. As in section 6, we examine the entropy of the distribution. The initial entropy is $n \log 2$. A distribution in which some bit has bias bounded away from 0 has entropy $(n - \Omega(1)) \log 2$. From the calculations in section 6 we see that the entropy of the distribution can decrease by at most $\log 2 - H(\tilde{\varepsilon}) \leq \tilde{\varepsilon}^2$ in a single cooling step. Hence a total of $\Omega(\tilde{\varepsilon}^{-2})$ cooling steps is required. □

8. Proof of Theorem 1.5. To this point we have concentrated on what can be achieved by alternating *arbitrary* permutations with cooling steps. It is not known whether the quality of initialization achieved in Theorems 1.2 and 1.3 can also be efficiently produced if the permutations must be implemented with one of the standard bases of reversible gates. However, we now outline why slightly weaker cooling can indeed be achieved using a simple sequence of standard reversible gates. Theorems 1.2 and 1.3 guarantee good initialization, provided, respectively, that $\tilde{\varepsilon} \geq 2^{4-n}$ and $\tilde{\varepsilon} \in \Omega(n2^{-n})$; the simple procedure provided in this section initializes a bit with bias $\Omega(1)$ within time $O((1/\tilde{\varepsilon})^{\log\log 1/\tilde{\varepsilon}})$, provided that $\tilde{\varepsilon} \in \Omega(\phi^{-n})$. (Recall that $\phi = (1+\sqrt{5})/2$.) More generally, for $N \leq \min\{n, \lceil \log_\phi 1/\tilde{\varepsilon} \rceil\}$, the procedure prepares a bit of bias $\Omega(\tilde{\varepsilon}F_N)$ within time $\exp(O(N \log N))$ using an N-bit device. In what follows set $N = \min\{n, \lceil \log_\phi 1/\tilde{\varepsilon} \rceil\}$.

Recall that $F_k = (2\phi/5 - 1/5)\phi^k - (2\phi/5 - 1/5)(1-\phi)^k$. For notational convenience we assume in this section that bit 1 (rather than n) is the special bit that can be directly cooled by the heat bath.

The procedure $\mathcal{F}$, taking argument $2 \leq N \leq \lceil \log_\phi 1/\tilde{\varepsilon} \rceil$, produces statistically independent bits $1, \ldots, N$, such that bit k (for every $1 \leq k \leq N$) has bias $\approx \tilde{\varepsilon}F_k$, or more specifically, bias $\geq \tilde{\varepsilon}F_k(1 - 2^{k-N-1})$. The sequence of quantum gates to be applied in this simple recursive procedure is easily generated by an NC1 circuit (whose input is the elapsed time in the cooling procedure).

Procedure $\mathcal{F}(N)$*:* Run $\mathcal{F}'(N, N)$.

Procedure $\mathcal{F}'(N, k)$*:*

(a) If $k = 2$, run the cooling step on bits 1 and (by exchange) 2.

(b) If $k > 2$, repeat steps (b1) and (b2) $O(N - k)$ times until the bias of bit k is at least $\tilde{\varepsilon}F_k(1 - 2^{k-N-1})$:

  (b1) Use a reversible majority gate to set bit k to be the majority of bits $k-2, k-1$, and k.

  (b2) Run $\mathcal{F}'(N, k-1)$.

(There are various ways to implement a reversible majority gate. Conceptually perhaps the simplest is the transformation of a triple of bits (a, b, c) into the triple $(\mathrm{MAJ}(a, b, c), a \oplus b, a \oplus c)$.)

We start with an imprecise version of the analysis. The effect of the majority gate in (b1) is, roughly, to transform bits with biases $\tilde{\varepsilon}\phi^{k-2}, \tilde{\varepsilon}\phi^{k-1}$, and $\tilde{\varepsilon}x$ into a bit of bias $\approx \tilde{\varepsilon}(\phi^{k-2} + \phi^{k-1} + x)/2$ (this is an approximation accurate for biases $\ll 1$). In each iteration within step (b), x converges toward the unique fixed point of this transformation, $x = \phi^k$. Convergence of the loop inside step (b) is rapid: in each iteration, the Lyapunov function $(\phi^k - x)^2$ decreases by a factor of almost 4. (When step (b) is very close to completion the factor is no longer close to 4 but remains bounded away from 1.) This is why $O(N - k)$ repetitions are enough.

The more careful analysis of $\mathcal{F}'(N, k)$ is this: by definition, the last recursive call to $\mathcal{F}'(N, k-1)$ terminated with bits $k-1$ and $k-2$ being independent and having biases at least $\tilde{\varepsilon}F_k(1 - 2^{k-N-2})$ and $\tilde{\varepsilon}F_k(1 - 2^{k-N-3})$. A few lines of calculation show that if the bias of bit k was $\tilde{\varepsilon}F_k(1-y)$ (before application of (b1)), then after the application it is at least $\tilde{\varepsilon}F_k(1-cy)$ for a fixed positive constant $c < 1$. Therefore $O(N - k)$ rounds suffice to drive the bias up to $\tilde{\varepsilon}F_k(1 - 2^{k-N-1})$.

Since procedure $\mathcal{F}'(N, k)$ makes $O(N - k)$ recursive calls to $\mathcal{F}'(N, k-1)$, the overall runtime of $\mathcal{F}'(N, N)$, and hence $\mathcal{F}(N)$, is $(N!)^{O(1)} = \exp(O(N \log N))$. This establishes Theorem 1.5. ∎

Historical notes. The application of majority gates to amplify bias began at least with von Neumann's work on fault tolerance [27]. The idea was later used as part of the design for algorithmic heat engines in [22]. An experiment to demonstrate the three-bit-majority primitive was conducted in [7]; a similar experiment was conducted by Sørensen [23], for NMR imaging amplification, before NMR quantum computers had been suggested. A simpler two-bit process, also pioneered by von Neumann [26] (for the quite different purpose of extracting fair from biased coin flips), was used for cooling in [22] and then in [5]. However, since the two-bit process amplifies bias only by order $\tilde{\varepsilon}^2$ rather than by order $\tilde{\varepsilon}$, majority gates were subsequently employed in [12]. In this section we have followed that approach but use a slightly different recursive procedure $\mathcal{F}$ to achieve scaling of the bias in powers of ϕ.

9. Discussion. *Numerical estimates.* We depict a specific way of using the PPA. Consider an ion trap quantum computer in which four qubits are reserved for preparation of ancillas, all others being devoted to the main quantum algorithm (including the fault-tolerance mechanism). Of the reserved qubits, three are "computation qubits" and one is the "refrigerant." Ion trap technology is capable of placing the refrigerant in its ground state with probability 0.95 (i.e., $\tilde{\varepsilon} = \operatorname{arctanh} 0.9 \approx 1.47$). Calculation shows that application of the PPA on the quadruple for just nine cooling steps suffices to prepare one of the qubits in the ground state with probability $1 - 10^{-4}$. This is at the conservative end of the estimates of between 10^{-4} and 10^{-2} for the fault-tolerance threshold for quantum computation. Hence after every nine cooling steps the PPA can prepare an ancilla, ready to be moved by spin exchange into the main bank of qubits (in place of a "warm" qubit generated by the fault-tolerance mechanism).

Implementation objectives. It is necessary to study the sensitivity of the model to imperfections in the cooling steps as well as in the logic gates between cooling steps, in specific experimental implementations.

Experimental algorithmic cooling also has the opportunity to produce a physically meaningful result well before producing a quantum computer. A series of papers [28, 25, 6] shows that if k qubits have bias less than 2^{-2k}, then their joint state is separable. Conversely, in the ball of radius $2^{-k/2}$ there exist nonseparable states. Liquid-state NMR experiments have not, to date, produced a demonstrably nonseparable state. Achieving this goal will require some combination of an increase in the number of coherently manipulated qubits and an increase in the individual polarization of these qubits. The latter demands implementation of new cooling techniques.

In the simple model adopted in this paper we have assumed that there is only a single refrigerant qubit. One may ask how the model is affected if the number of such qubits is proportional to the number of computation qubits. (In liquid-state NMR, for example, we can expect that nuclei of various types will be present in fixed proportions.) The answer is that while some gain is likely, the fundamental limits of the model are unchanged because with a slowdown in the cooling process by a factor of $O(n)$, the same effect can be achieved by spin exchange with a single refrigerant qubit.

The present paper leaves open whether there is a simple implementation of the PPA or whether some other simply implemented algorithm can achieve the same $\varepsilon \approx 2^{-n}$ threshold.

The necessity of cooling many qubits for quantum computation. In view of the difficulty of cooling certain kinds of quantum computers, the question was posed of whether this was truly necessary [17]. Since a uniformly mixed state is unchanged by reversible (unitary) operations, computation is impossible (the statistics of the final

state do not depend on the computation steps) unless the initial mixture can be transformed into something other than the uniform mixture. Interestingly, this does not rule out the possibility of quantum-over-classical computational speedups on devices that are initialized in a highly (though not completely) mixed state. A key example was provided in [17]: the trace of a unitary operator of dimension 2^n can be computed on a device with n qubits, of which just one is strongly biased while the others are maximally mixed. (For related recent work see [20].) However, it was demonstrated in [2] that there is no way of directly simulating general quantum computers on highly mixed devices such as this. Hence computations on such devices can be accomplished, if at all, only with tailor-made algorithms. The available evidence suggests that such devices, even if noise-free, would be strictly weaker than general-purpose quantum computers, and so the suggestion in [17] is unlikely to circumvent the need for effective cooling. The necessity of using ancillas to compensate for noise buttresses this conclusion.

Summary. We have studied the fundamental limits of open-system "heat-bath" cooling, with a view to the significance of such methods for quantum computation as well as for imaging tasks limited by imperfect state preparation. We have provided a cooling (bias amplification) method and shown the following: (a) The bias it achieves is substantially higher than in previous methods, and the ground-state probability after any number of cooling steps is highest possible. (b) The number of cooling steps it requires is asymptotically close to best possible. (c) There is a sharp threshold for the heat-bath temperature, above which substantial cooling is impossible in any method, and below which it is achieved by ours.

Acknowledgments. Thanks go to R. Laflamme and J. Fernandez for helpful discussions, and to an anonymous referee for a careful reading of the manuscript.

REFERENCES

[1] D. Aharonov and M. Ben-Or, *Fault-tolerant quantum computation with constant error*, in Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, 1997, pp. 176–188.

[2] A. Ambainis, L. J. Schulman, and U. Vazirani, *Computing with highly mixed states*, in Proceedings of the Thirty-Second Annual ACM Symposium on the Theory of Computing, 2000, pp. 705–714.

[3] M. D. Barrett, B. DeMarco, T. Schaetz, V. Mayer, D. Leibfried, J. Britton, J. Chiaverini, W. M. Itano, B. Jelenkovic, J. D. Jost, C. Langer, T. Rosenband, and D. J. Wineland, *Sympathetic cooling of* $^9be^+$ *and* $^24mg^+$ *for quantum logic*, Phys. Rev. A, 68 (2003), 042302.

[4] C. M. Bowden, J. P. Dowling, and S. P. Hotaling, *Quantum computing using electron-nuclear double resonances*, in SPIE Proceedings 3076: Photonic Quantum Computing, 1997, pp. 173–182.

[5] P. O. Boykin, T. Mor, V. Roychowdhury, F. Vatan, and R. Vrijen, *Algorithmic cooling and scalable NMR quantum computers*, Proc. Natl. Acad. Sci. USA, 99 (2002), pp. 3388–3393.

[6] S. L. Braunstein, C. M. Caves, R. Josza, N. Linden, S. Popescu, and R. Schack, *Separability of very noisy mixed states and implications for NMR quantum computing*, Phys. Rev. Lett., 83 (1999), pp. 1054–1057.

[7] D. E. Chang, L. M. K. Vandersypen, and M. Steffen, *NMR implementation of a building block for scalable quantum computation*, Chem. Phys. Lett., 338 (2001), pp. 337–344.

[8] J. I. Cirac and P. Zoller, *Quantum computations with cold trapped ions*, Phys. Rev. Lett., 74 (1995), pp. 4091–4094.

[9] D. G. Cory, A. F. Fahmy, and T. F. Havel, *Ensemble quantum computing by nuclear magnetic resonance spectroscopy*, Proc. Nat. Acad. Sci. USA, 94 (1997), pp. 1634–1639.

[10] D. P. DiVincenzo, *Topics in quantum computers*, in Mesoscopic Electron Transport, Kluwer, Dordrecht, 1997, pp. 657–667.
[11] D. P. DiVincenzo, *The physical implementation of quantum computation*, Fortschr. Phys., 48 (2000), pp. 771–783.
[12] J. M. Fernandez, S. Lloyd, T. Mor, and V. Roychowdhury, *Algorithmic cooling of spins: A practicable method for increasing polarization*, Internat. J. Quantum Inf., 2 (2004), pp. 461–477.
[13] N. A. Gershenfeld and I. L. Chuang, *Bulk spin-resonance quantum computation*, Science, 275 (1997), pp. 350–356.
[14] M. Iinuma, Y. Takahashi, I. Shaké, M. Oda, A. Masaike, T. Yabuzaki, and H. M. Shimizu, *High proton polarization by microwave-induced optical nuclear polarization at* 77 *K*, Phys. Rev. Lett., 84 (2000), pp. 171–174.
[15] B. E. King, C. S. Wood, C. J. Myatt, Q. A. Turchette, D. Leibfried, W. M. Itano, C. Monroe, and D. J. Wineland, *Cooling the collective motion of trapped ions to initialize a quantum register*, Phys. Rev. Lett., 81 (1998), pp. 1525–1528.
[16] E. Knill, *Fault-Tolerant Postselected Quantum Computation: Threshold Analysis*, http://arxiv.org/abs/quant-ph/0404104 (2004).
[17] E. Knill and R. Laflamme, *On the power of one bit of quantum information*, Phys. Rev. Lett., 81 (1998), pp. 5672–5675.
[18] A. W. Marshall and I. Olkin, *Inequalities: Theory of Majorization and its Applications*, Academic Press, New York, London, 1979.
[19] C. Monroe, D. M. Meekhof, B. E. King, W. M. Itano, and D. J. Wineland, *Demonstration of a fundamental quantum logic gate*, Phys. Rev. Lett., 75 (1995), pp. 4714–4717.
[20] D. Poulin, R. Blume-Kohout, R. Laflamme, and H. Ollivier, *Exponential speed-up with a single bit of quantum information: Measuring the average fidelity decay*, Phys. Rev. Lett., 92 (2004), 177906.
[21] L. J. Schulman, T. Mor, and Y. Weinstein, *Physical limits of heat-bath algorithmic cooling*, Phys. Rev. Lett., 94 (2005), 120501.
[22] L. J. Schulman and U. Vazirani, *Molecular scale heat engines and scalable quantum computation*, in Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing, 1999, pp. 322–329.
[23] O. W. Sørensen, *Polarization transfer experiments in high-resolution NMR spectroscopy*, Prog. NMR Spec., 21 (1989), pp. 504–569.
[24] A. S. Verhulst, O. Liivak, M. H. Sherwood, H.-M. Vieth, and I. L. Chuang, *Non-thermal nuclear magnetic resonance quantum computing using hyperpolarized xenon*, Appl. Phys. Lett., 79 (2001), pp. 2480–2482.
[25] G. Vidal and R. Tarrach, *Robustness of entanglement*, Phys. Rev. A, 59 (1999), pp. 141–155.
[26] J. von Neumann, *Various techniques used in connection with random digits*, in The Monte Carlo Method, Nat. Bur. Standards Appl. Math. Ser. 12, U.S. Government Printing Office, Washington, D.C., 1951, pp. 36–38.
[27] J. von Neumann, *Probabilistic logics and the synthesis of reliable organisms from unreliable components*, in Automata Studies, C. E. Shannon and J. McCarthy, eds., Princeton University Press, Princeton, NJ, 1956, pp. 43–98.
[28] K. Zyczkowski, P. Horodecki, A. Sanpera, and M. Lewenstein, *Volume of the set of separable states*, Phys. Rev. A, 58 (1998), pp. 883–892.

SIAM J. COMPUT.
Vol. 36, No. 6, pp. 1748–1763

WHOLE GENOME DUPLICATIONS AND CONTRACTED BREAKPOINT GRAPHS*

MAX A. ALEKSEYEV† AND PAVEL A. PEVZNER†

Abstract. The genome halving problem, motivated by the whole genome duplication events in molecular evolution, was solved by El-Mabrouk and Sankoff in the pioneering paper [*SIAM J. Comput.*, 32 (2003), pp. 754–792]. The El-Mabrouk–Sankoff algorithm is rather complex, inspiring a quest for a simpler solution. An alternative approach to the genome halving problem based on the notion of the contracted breakpoint graph was recently proposed in [M. A. Alekseyev and P. A. Pevzner, *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 4 (2007), pp. 98–107]. This new technique reveals that while the El-Mabrouk–Sankoff result is correct in most cases, it does not hold in the case of unichromosomal genomes. This raises a problem of correcting a flaw in the El-Mabrouk–Sankoff analysis and devising an algorithm that deals adequately with all genomes. In this paper we efficiently classify all genomes into two classes and show that while the El-Mabrouk–Sankoff theorem holds for the first class, it is incorrect for the second class. The crux of our analysis is a new combinatorial invariant defined on duplicated permutations. Using this invariant we were able to come up with a full proof of the genome halving theorem and a polynomial algorithm for the genome halving problem.

Key words. genome duplication, genome halving, genome rearrangement, breakpoint graph, de Bruijn graph

AMS subject classifications. 90C27, 90C35, 90C46, 94C15

DOI. 10.1137/05064727X

1. Introduction.

In 1970 Susumu Ohno came up with two fundamental theories of chromosome evolution that were the subjects of many controversies in the last 35 years [31]. The first, random breakage theory, was embraced by biologists from the very beginning but was refuted by Pevzner and Tesler in 2003 [35] and Murphy et al. in 2005 [30]. The second, whole genome duplication theory, postulated a new type of evolutionary event and had a very different fate. It was subject to controversy in the first 35 years and only recently was proven to be correct [27, 15]. Kellis, Birren, and Lander in 2004 [27] sequenced the yeast *K. waltii* genome, compared it with the yeast *S. cerevisiae* genome, and demonstrated that nearly every region in *K. waltii* corresponds to two regions in *S. cerevisiae*, thus proving that there was a whole genome duplication event in the course of yeast evolution. This discovery was quickly followed by the discovery of whole genome duplications in vertebrates [24, 36, 12] and plants [21]. Finally, in September, 2005 Dehal and Boore [14] found evidence of two rounds of whole genome duplications on the evolutionary path from early vertebrates to humans. Shortly afterwards, Meyer and Van de Peer [28] found evidence of yet another (third) round of whole genome duplications in ray-finned fishes, thus implying that nearly every human gene could have existed in as many as eight copies at different stages of evolution.

These recent studies provided irrefutable evidence that whole genome duplications represent a new type of event that may explain phenomena which classical evolutionary studies have had difficulty explaining (e.g., emergence of new metabolic

*Received by the editors December 12, 2005; accepted for publication (in revised form) October 19, 2006; published electronically March 19, 2007.
http://www.siam.org/journals/sicomp/36-6/64727.html
†Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093 (maxal@cs.ucsd.edu, ppevzner@cs.ucsd.edu).

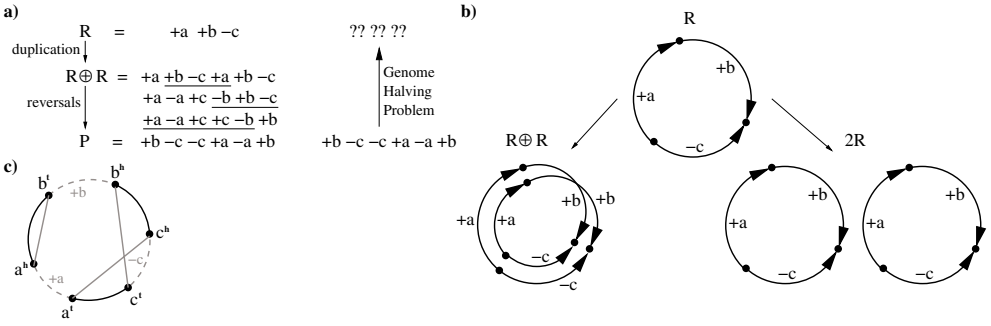


FIG. 1. (a) *Whole genome duplication of genome* $R = +a + b - c$ *into a perfect duplicated genome* $R \oplus R = +a + b - c + a + b - c$ *followed by three reversals.* (b) *Whole genome duplication of a circular genome* R *(center) resulting in* $R \oplus R$ *(left) or* $2R$ *(right).* (c) *Breakpoint graph of genomes* $+a + b - c$ *and* $+a + b + c$.

pathways [27]). At the same time, they raised the problem of reconstructing the genomic architecture of the ancestral preduplicated genomes. Unfortunately, since the El-Mabrouk–Sankoff algorithm for solving this problem [20] has not yet resulted in a software tool, the recent studies of whole genome duplications did not attempt to rigorously reconstruct the architecture of the preduplicated genomes. We revisited the El-Mabrouk–Sankoff result, found a flaw in their approach, reformulated and proved the genome halving theorem, and developed a new algorithm and software tool for studies of genome duplications.

Whole genome duplications double the gene content of a genome R and result in a *perfect duplicated genome* $R \oplus R$ that contains two identical copies of each chromosome. The genome then becomes subject to rearrangements that shuffle the genes in $R \oplus R$, resulting in some *rearranged duplicated genome* P. The *genome halving problem* is to reconstruct the ancestral *preduplicated genome* R from the rearranged duplicated genome P (Figure 1(a)).

From an algorithmic perspective, the genome is a collection of chromosomes, and each chromosome is a *signed sequence* over a finite alphabet. DNA has two strands, and genes on a chromosome have directionality that reflects the strand of the genes. We represent the order and directions of the genes on each chromosome as a sequence of *signed elements*, i.e., elements with signs "+" and "-". In this paper we focus on the basic *unichromosomal* case, where the genomes consist of just one chromosome, and assume that the genomes are *circular*. A unichromosomal genome, where each gene appears in a single copy, is referred to as *signed permutation*. For unichromosomal genomes the rearrangements are limited to *reversals* that "flip" genes $x_i \dots x_j$ in a genome $x_1 x_2 \dots x_n$ as follows:

$$x_1 \dots x_{i-1} \underrightarrow{x_i \quad x_{i+1} \dots \quad x_j} x_{j+1} \dots x_n \longrightarrow x_1 \dots x_{i-1} \underleftarrow{-x_j - x_{j-1} \cdots - x_i} x_{j+1} \dots x_n.$$

The *reversal distance* between two genomes is defined as the minimum number of reversals required to transform one genome into the other (see Chapter 10 of [34] for a review of genome rearrangement algorithms).

We represent a circular genome R as a cycle formed by directed edges encoding the genes and their directions (Figure 2(b), center). There are two natural ways to represent duplication of the genome R resulting in a unichromosomal genome $R \oplus R$ (Figure 1(b), left) and a multichromosomal genome $2R$ (Figure 1(b), right) but only the former is applicable to unichromosomal genomes.

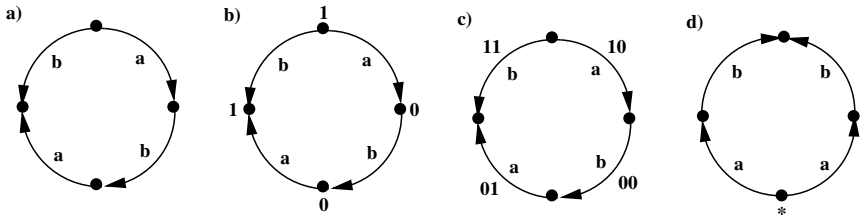


FIG. 2. (a) *Circular genome* $P = +a - b + a + b$ *represented as a cycle with directed edges.* (b) 01*-labeling of the vertices of the cycle defined by* P. (c) *Induced labeling of the genes of* P *that is consistent.* (d) *For some genomes consistent labelings do not exist: for genome* $Q = +a + b - b - a$ *the labels of both copies of gene* a *start with the same digit ("*$*$*") so they cannot be inversions of each other.*

For unichromosomal genomes, *whole genome duplication* is a concatenation of the genome R with itself, resulting in a *perfect duplicated genome* $R \oplus R$. The genome $R \oplus R$ becomes subject to reversals that change the order and signs of the genes and transform $R \oplus R$ into a *duplicated* genome P. The genome halving problem is formulated as follows.

GENOME HALVING PROBLEM. *Given a duplicated genome* P*, recover an ancestral preduplicated genome* R *minimizing the reversal distance* $d(P, R \oplus R)$ *from the perfect duplicated genome* $R \oplus R$ *to* P.

The genome halving problem was solved in a series of papers [18, 19, 17] culminating in a rather complex algorithm by El-Mabrouk and Sankoff in [20]. The El-Mabrouk–Sankoff algorithm is one of the most technically challenging results in computational biology and its proof spans over 30 pages in [20]. Recently Alekseyev and Pevzner [1] revisited the El-Mabrouk–Sankoff work and presented an alternative approach based on the notion of a *contracted breakpoint graph*.

After paper [1] was submitted, our studies of the contracted breakpoint graph led us to realize that the El-Mabrouk–Sankoff analysis has a flaw and that the problem of finding $\min_R d(P, R \oplus R)$ remains unsolved in the simplest case when P is a unichromosomal genome. Below we show that this flaw is a rule rather than a pathological case: it affects a large family of duplicated genomes. We further proceed to give a full analysis of the genome halving problem that is based on introducing an invariant that divides the set of all rearranged duplicated genomes into two classes. We show that the El-Mabrouk–Sankoff formula is correct for the first class but is off by 1 for the second class. We remark that our approach is very different from [20] and we do not know whether the technique in [20] can be adjusted to address the described complication.

To introduce a new combinatorial invariant of duplicated genomes, consider labelings of vertices in the cycle defined by the duplicated rearranged genome P with numbers 0 and 1 (Figure 2(b)). Every such labeling induces a two-digit labeling of the genes (edges): a label of each gene is formed by the labels of the incident vertices (Figure 2(c)). A 01-labeling of the vertices is called *consistent* if for every pair of identical genes in P the label of one copy is an inversion of the other. If there exists a consistent labeling of genome P, we define the *parity index* of P as the number of genes labeled "01" modulo 2. Below we prove that the parity index is well defined, i.e., the parity index is the same for all consistent labelings of a genome. It turns out that the El-Mabrouk–Sankoff theorem fails on genomes with the parity index 0.

The paper is organized as follows. Section 2 presents the concept of contracted

breakpoint graph and reviews some results from [1]. Section 3 describes a flaw in the El-Mabrouk–Sankoff analysis. Section 4 presents a solution to the genome halving problem (for unichromosomal circular genomes) and a classification of the genomes for which the original El-Mabrouk–Sankoff theorem is incorrect. Section 5 outlines our genome halving algorithm. Finally, section 6 discusses potential biological applications of the presented algorithm.

2. Reversal distance between duplicated genomes and contracted breakpoint graphs. A duality theorem and a polynomial algorithm for computing reversal distance between two signed permutations was proposed by Hannenhalli and Pevzner [23] and later was generalized for multichromosomal genomes [22]. The algorithm was further simplified and improved in a series of papers [6, 25, 3, 7, 42, 26] and applied in a variety of biological studies [29, 10, 8, 33, 5].

A signed permutation on n elements can be transformed into an unsigned permutation on $2n$ elements (see [4]) by substituting every element x in the signed permutation with two elements x^t and x^h in the unsigned permutation (indices t and h stand for tail and head, respectively). Each element $+x$ in the permutation P is replaced with $x^t x^h$, and each element $-x$ is replaced with $x^h x^t$, resulting in an unsigned permutation $\pi(P)$. For example, a permutation $+a+b-c$ will be transformed into $a^t a^h b^t b^h c^h c^t$. Element x^t is called the *obverse* of element x^h, and vice versa.

Let P and Q be two circular signed permutations on the same set of elements $\mathcal{G}$, and let $\pi(P)$ and $\pi(Q)$ be corresponding unsigned permutations. The *breakpoint graph* $G = G(P, Q)$ is defined on the set of vertices $V = \{x^t, x^h \mid x \in \mathcal{G}\}$ with edges of three colors: obverse, black, and gray (Figure 1(c)). Edges of each color form a matching on V as follows:

- pairs of obverse elements form an *obverse matching*;
- adjacent elements in $\pi(P)$, other than obverses, form a *black matching*;
- adjacent elements in $\pi(Q)$, other than obverses, form a *gray matching*.

Every pair of matchings forms a collection of *alternating* cycles in G, called *black-gray*, *black-obverse*, and *gray-obverse*, respectively (a cycle is alternating if colors of its edges alternate). The permutation $\pi(P)$ can be read along a single black-obverse cycle, while the permutation $\pi(Q)$ can be read along a single gray-obverse cycle in G. The black-gray cycles in the breakpoint graph play an important role in computing the reversal distance. According to the Hannenhalli–Pevzner theorem, the reversal distance between permutations P and Q is given by the formula

$$d(P, Q) = |P| - c(P, Q) + h(P, Q), \tag{1}$$

where $|P| = |Q|$ is the size of P and Q, $c(P, Q) = c(G(P, Q))$ is the number of black-gray cycles in the breakpoint graph G, and $h(G)$ is an easily computable combinatorial parameter. While this result leads to a fast algorithm for computing reversal distance between two signed permutations, the problem of computing reversal distance between two genomes with duplicated genes remains unsolved.

Let P and Q be duplicated genomes on the same set of genes $\mathcal{G}$. If one labels copies of each gene x as x_1 and x_2, then the genomes P and Q become signed permutations and (1) applies. The breakpoint graph $G(P, Q)$ of the labeled genomes P and Q has a vertex set $V = \{x_1^t, x_1^h, x_2^t, x_2^h \mid x \in \mathcal{G}\}$ and uniquely defines permutations $\pi(P)$ and $\pi(Q)$. We remark that different labelings may lead to different breakpoint graphs (on the same vertex set) for the same genomes P and Q (Figure 3) and it is not clear how to choose a labeling that results in the minimum reversal distance between the labeled copies of P and Q. We also remark that pairs of vertices x_1^j and x_2^j form yet another

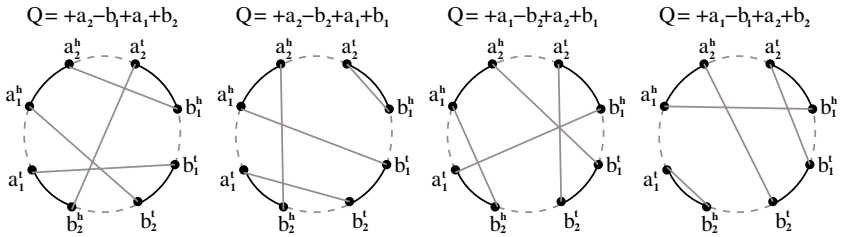


FIG. 3. *Breakpoint graphs for $P = +a - a - b + b$ and four different labelings $Q = +a - b + a + b$ (we assume that the labeling of $P = +a_1 - a_2 - b_1 + b_2$ is fixed). Two out of four breakpoint graphs have $c(G) = 1$, while two others have $c(G) = 2$. The counterpart matching in these graphs is formed by pairs (a_1^t, a_2^t), (a_1^h, a_2^h), (b_1^t, b_2^t), (b_1^h, b_2^h).*

matching in the breakpoint graph G called *counterpart*. Counterpart of a vertex v is denoted $\bar{v}$ so that $\bar{x}_1^j = x_2^j$ and $\bar{x}_2^j = x_1^j$ (see legend for Figure 3).

Recently there were many attempts to generalize the Hannenhalli–Pevzner theory for genomes with duplicated and deleted genes [9, 11, 16, 37, 40, 41]. However, the only known option for solving the reversal distance problem for duplicated genomes *exactly* is to consider all possible labelings, to compute the reversal distance problem for each labeling, and to choose the labeling with the minimal reversal distance. For duplicated genomes with n genes this leads to 2^n invocations of the Hannenhalli–Pevzner algorithm, rendering this approach impractical. Moreover, the problem remains open if one of the genomes is perfectly duplicated (i.e., computing $d(P, R \oplus R)$). Surprisingly, the problem of computing $\min_R d(P, R \oplus R)$ that we address in this paper is solvable in polynomial time.

Using the concept of the breakpoint graph and formula (1), the genome halving problem can be posed as follows. For a duplicated genome P, find a perfect duplicated genome $R \oplus R$ and a labeling of gene copies such that the breakpoint graph $G(P, R \oplus R)$ of the labeled genomes P and $R \oplus R$ attains the minimum value of $|P| - c(P, Q) + h(P, Q)$. Since $|P|$ is constant and $h(G)$ is typically small (see [34]), the value of $d(P, Q)$ depends mainly on $c(P, Q)$. El-Mabrouk and Sankoff [20] established that the problems of maximizing $c(P, Q)$ and minimizing $h(P, Q)$ can be solved separately in a consecutive manner.[1] In this paper we focus on the former and harder problem as follows.

WEAK GENOME HALVING PROBLEM. *For a given duplicated genome P, find a perfect duplicated genome $R \oplus R$ and a labeling of gene copies that maximizes the number of black-gray cycles $c(P, R \oplus R)$ in the breakpoint graph $G(P, R \oplus R)$ of the labeled genomes P and $R \oplus R$.*

From now on, we will find it convenient to represent a circular signed permutation as an alternating cycle formed by edges of two colors with one color reserved for obverse edges. For example, Figures 4(a),(b) show a black-obverse cycle representation of permutation $P = +a - a - b + b$ and a gray-obverse cycle representation of permutation $Q = +a - b + a + b$ (the obverse edges in these cycles are labeled and directed). Given a set of edge-labeled graphs, the *de Bruijn graph* of this set is defined as the result of "gluing"[2] edges with the same label in all graphs in the set (compare with Pevzner,

[1] In paper [2] we describe an analogue of formula (1) without the "$h(G)$" term and give a short proof of the genome halving theorem (for multichromosomal genomes) that does not rely on the analysis in [20].

[2] Gluing takes into account the directions of edges; i.e., tails (or heads) of all edges with a given label are glued into a single vertex.

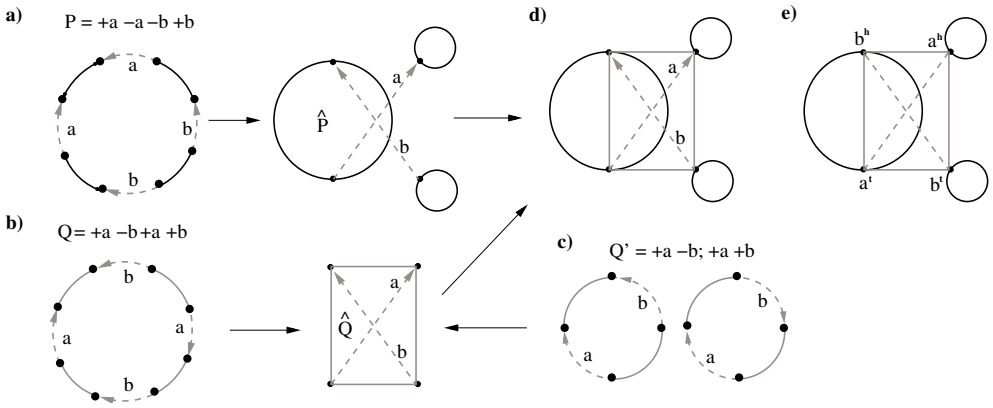


FIG. 4. (a) *Genome $P = +a - a - b + b$ as a black-obverse cycle and its transformation into $\hat{P}$ by gluing identically labeled edges.* (b) *Genome $Q = +a - b + a + b$ as a gray-obverse cycle and its transformation into $\hat{Q}$ by gluing identically labeled edges.* (c) *Two-chromosomal genome $Q' = (+a - b)(+a + b)$ that is equivalent to the genome Q (i.e., $\hat{Q}' = \hat{Q}$).* (d) *The de Bruijn graph of genomes P and Q.* (e) *The contracted breakpoint graph $G'(P, Q)$.*

Tang, and Tesler [32]). The de Bruijn graph for two cycles in Figures 4(a),(b) is shown in Figure 4(d).

For any genome P (represented as a cycle) we define $\hat{P}$ as the graph obtained from P by gluing identically labeled edges. Obviously, the de Bruijn graph of P and Q coincides with the de Bruijn graph of $\hat{P}$ and $\hat{Q}$ (Figure 4). For a duplicated genome P, the black edges of $\hat{P}$ form a set of vertex-disjoint black cycles. We denote by $b_e(P)$ the number of even black cycles in $\hat{P}$.

The conventional breakpoint graph [4] of signed permutations P and Q on n elements can be defined as the gluing of n *pairs* of identically labeled obverse edges in the corresponding permutations (represented as black-obverse and gray-obverse cycles). The *contracted breakpoint graph* of duplicated genomes P and Q on n elements is simply the gluing of n *quartets* of obverse edges. Below we give a somewhat more formal definition of the contracted breakpoint graph.

Let P and Q be duplicated genomes on the same set of genes $\mathcal{G}$ and let G be a breakpoint graph defined by some labeling of P and Q. The *contracted breakpoint graph* $G'(P, Q)$ is the result of contracting every pair of vertices x_1^j, x_2^j (where $x \in \mathcal{G}$, $j \in \{t, h\}$) in the breakpoint graph G into a single vertex x^j. So the contracted breakpoint graph $G' = G'(P, Q)$ is a graph on the set of vertices $V' = \{x^t, x^h \mid x \in \mathcal{G}\}$ with each vertex incident to two black, two gray, and a *pair* of parallel obverse edges (Figure 4(e)). The contracted breakpoint graph $G'(P, Q)$ does not depend on a particular labeling of P and Q. The following theorem gives a characterization of the contracted breakpoint graphs (for unichromosomal genomes).

THEOREM 1 (see [1]). *A graph H with black, gray, and obverse edges is a contracted breakpoint graph for some duplicated genomes if and only if*

- *each vertex in H is incident to two black edges, two gray edges, and two parallel obverse edges;*
- *H is connected with respect to black and obverse edges (black-obverse connected);*
- *H is connected with respect to gray and obverse edges (gray-obverse connected).*

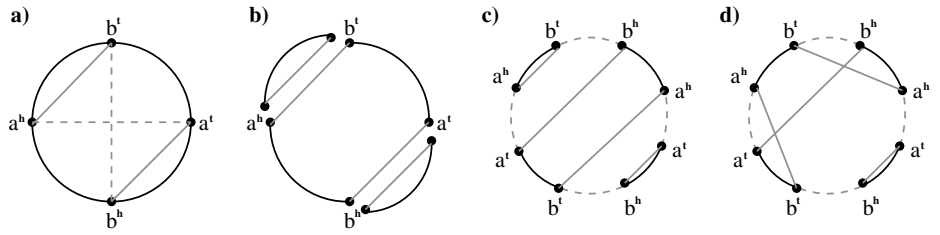


FIG. 5. (a) *Contracted breakpoint graph* $G'(P, R \oplus R)$ *for* $P = +a + b - a - b$ *and* $R = +a + b$. (b) *Black-gray cycle decomposition* C *of* G' *which is not induced by any labeling of* P *and* $R \oplus R$. (c) *Breakpoint graph* $G(P, 2R)$ *inducing* C. (d) *Breakpoint graph* $G(P, R \oplus R)$ *(unique up to relabeling of vertices) with* $c(G) = 2 < |C| = 3$.

In the case when $Q = R \oplus R$ is a perfect duplicated genome, the gray edges in the contracted breakpoint graph $G'(P, Q)$ form pairs of parallel gray edges that we refer to as *double* gray edges. Similar to the double obverse edges, the double gray edges form a matching in G' (Figure 5(a)).

Let $G(P, Q)$ be a breakpoint graph for some labeling of P and Q. A set of black-gray cycles in $G(P, Q)$ is contracted into a set of black-gray cycles in the contracted breakpoint graph $G'(P, Q)$, thus forming a black-gray cycle decomposition of $G'(P, Q)$. Therefore, each labeling induces a black-gray cycle decomposition of $G'(P, Q)$. We are interested in the reverse problem as follows.

LABELING PROBLEM. *Given a black-gray cycle decomposition of the contracted breakpoint graph* $G'(P, Q)$ *of duplicated genomes* P *and* Q*, find a labeling of* P *and* Q *that induces this cycle decomposition.*

This problem may not always have a solution for unichromosomal genomes (Figure 5) and this is exactly the factor that leads to a counterexample in section 3. This complication will be addressed in section 4 using the following three theorems proved in [1].

THEOREM 2 (see [1]). *Let* P *and* $R \oplus R$ *be unichromosomal duplicated genomes and* C *be a black-gray cycle decomposition of the contracted breakpoint graph* $G'(P, R \oplus R)$. *Then there exists some labeling of* P *and either* $R \oplus R$ *or* $2R$ *that induces the cycle decomposition* C.

Let $c_{max}(P, R \oplus R) = c_{max}(G'(P, R \oplus R))$ be the number of cycles in a maximal black-gray cycle decompositions of the contracted breakpoint graph $G'(P, R \oplus R)$. Theorem 2 motivates the following problem that will later help us to solve the weak genome halving problem.

CYCLE DECOMPOSITION PROBLEM. *For a given duplicated genome* P*, find a perfect duplicated genome* $R \oplus R$ *maximizing* $c_{max}(P, R \oplus R)$.

Although the maximal black-gray cycle decomposition of $G'(P, R \oplus R)$ may correspond to a breakpoint graph $G(P, 2R)$ (Figure 5), we will prove below that there exists a breakpoint graph $G(P, R \oplus R)$ having "almost" the same number of black-gray cycle as $G(P, 2R)$ (Figure 5(d)). Later we will classify all the cases in which there exists a labeled genome $R' \oplus R'$ such that $c(P, R' \oplus R') = c(P, 2R)$.

The solution to the cycle decomposition problem is given by the following two theorems.

THEOREM 3 (see [1]). *For a given duplicated genome* P *and any perfect duplicated genome* $R \oplus R$,

$$c_{max}(P, R \oplus R) \leq |P|/2 + b_e(P),$$

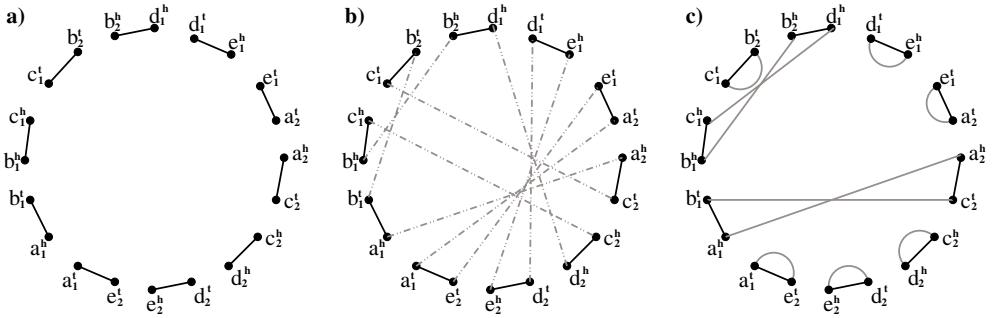


FIG. 6. (a) *A set of black edges forming the partial graph* $\mathcal{G}(\mathbf{V}, A)$ *corresponding to the genome* $P = +a+b-c+b-d-e+a+c-d-e$. (b) *Natural graphs as connected components in the partial graph with counterpart edges.* (c) *A completed graph* $\mathcal{G}(\mathbf{V}, A, \Gamma)$ *with maximum number of cycles* $c(G) = 8$. $\mathcal{G}(\mathbf{V}, A, \Gamma)$ *is a breakpoint graph of the circular genome* $P = +a_1+b_1-c_1+b_2-d_1-e_1+a_2+c_2-d_2-e_2$ *and a perfect duplicated genome* $(-a_1 + e_2 + d_2 - c_2 + b_1)(-b_2 + c_1 - d_1 - e_1 + a_2)$ *(of the form* $R \ominus R$*).*

where $|P|/2$ *represents the number of unique genes in* P *and* $b_e(P)$ *is the number of even black cycles in* $\hat{P}$*. Moreover, if* $c_{max}(P, R \oplus R) = |P|/2 + b_e(P)$*, then each black-gray connected component of* $G'(P, R \oplus R)$ *contains either a single even black cycle (simple component) or a pair of odd black cycles (paired component).*

THEOREM 4 (see [1]). *For any duplicated genome* P*, there exists a perfect duplicated genome* $R \oplus R$ *such that*

$$c_{max}(P, R \oplus R) = |P|/2 + b_e(P)$$

and each paired component of $G'(P, R \oplus R)$ *contains a single interedge (a double gray edge connecting distinct black cycles).*

3. A flaw in the El-Mabrouk–Sankoff analysis. El-Mabrouk and Sankoff came up with a theorem describing the minimum distance from the given rearranged duplicated genome to a perfect duplicated genome. Given a rearranged duplicated genome P, the crux of their approach is an algorithm for computing $c(G)$—the number of cycles of a so-called *maximal completed graph*, i.e., a breakpoint graph[3] with the maximum number of black-gray cycles. In [20] they demonstrate that $c(G)$ equals the number of genes plus $\gamma(G)$, where $\gamma(G)$ is the parameter defined below. We illustrate the concepts from [20] using the genome $P = +a + b - c + b - d - e + a + c - d - e$ on the set of genes $\mathcal{B} = \{a, b, c, d, e\}$ (page 757 in [20]). El-Mabrouk and Sankoff first arbitrarily label two copies of each gene x as x_1 and x_2 for each $x \in \mathcal{B}$ and further transform the signed permutation G into an unsigned permutation $a_1^t a_1^h b_1^t b_1^h c_1^h c_1^t b_2^t b_2^h d_1^h d_1^t e_1^h e_1^t a_2^t a_2^h c_2^t c_2^h d_2^h d_2^t e_2^h e_2^t$.

Let $\mathbf{V} = \{x_1^t, x_1^h, x_2^t, x_2^h \mid x \in \mathcal{B}\}$. The *partial graph* $\mathcal{G}(\mathbf{V}, A)$ associated with P has the edge set A of black edges linking adjacent terms (other than obverses x_i^t and x_i^h) in the corresponding unsigned permutation (Figure 6(a)).

Black edges together with counterpart edges (i.e., edges between x_1^t and x_2^t or between x_1^h and x_2^h) form a graph shown in Figure 6(b). The connected components of this graph are called *natural graphs* in [20]. There are four connected components (natural graphs) in the graph in Figure 6(b), two of them have three black edges (odd

[3]Following El-Mabrouk and Sankoff [20] we ignore obverse edges in breakpoint graphs throughout section 3.

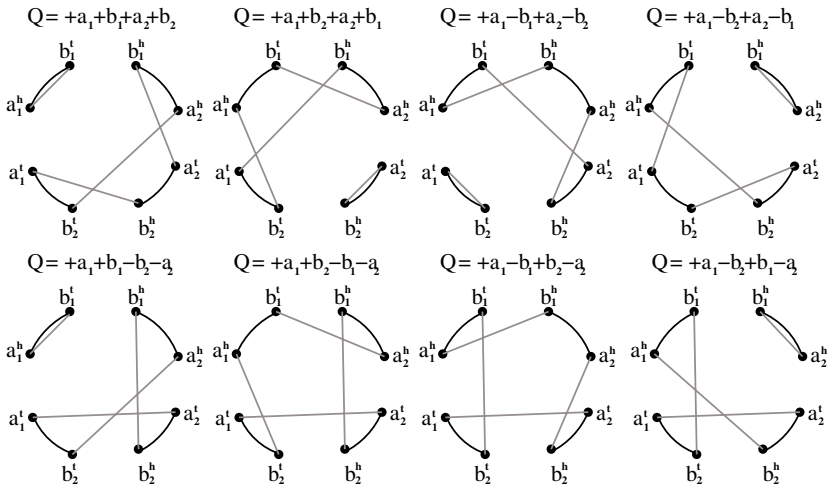


FIG. 7. *Breakpoint graphs of the circular genome $P = +a + b - a - b$ and all possible labelings of all possible perfect duplicated genomes Q (without loss of generality, we assume that labeling of $P = +a_1 + b_1 - a_2 - b_2$ is fixed). In terms of* [20]*, the top four graphs correspond to an $R \oplus R$ duplication pattern, while the bottom four graphs correspond to an $R \ominus R$ duplication pattern.*

natural graphs) and two of them have two black edges (even natural graphs). Let NE be the number of even natural graphs ($NE = 2$ in Figure 6(b)).

El-Mabrouk and Sankoff define the parameter

$$\gamma(G) = \begin{cases} NE & \text{if all natural graphs are even,} \\ NE + 1 & \text{otherwise.} \end{cases}$$

A graph $\mathcal{G}(\mathbf{V}, A, \Gamma)$ obtained from the partial graph $\mathcal{G}(\mathbf{V}, A)$ by introducing a set of gray edges Γ is called a *completed graph* if $\mathcal{G}(\mathbf{V}, A, \Gamma)$ is a breakpoint graph for some genomes on the set of genes $\{x_1, x_2 \mid x \in \mathcal{B}\}$. The following theorem (Theorem 7.7 in [20]) characterizes the maximum number of cycles in the completed graph $\mathcal{G}(\mathbf{V}, A, \Gamma)$.

THEOREM. *The maximal number of cycles in a completed graph of $\mathcal{G}(\mathbf{V}, A)$ is $c(G) = \frac{|A|}{2} + \gamma(G)$.*

For the genome in Figure 6 we have $\gamma(G) = NE + 1 = 3$ and $c(G) = \frac{|A|}{2} + \gamma(G) = \frac{10}{2} + 3 = 8$. A completed graph $\mathcal{G}(\mathbf{V}, A, \Gamma)$ with eight cycles is shown at Figure 6(c).[4] Below we provide a counterexample to Theorem 7.7 from [20].

Consider a circular genome $P = +a + b - a - b$ labeled as $+a_1 + b_1 - a_2 - b_2$. The genome P defines a partial graph $\mathcal{G}(\mathbf{V}, A)$ with a single natural graph of even size implying $\gamma(G) = 1$. It follows from Theorem 7.7 in [20] that there exists a perfect duplicated genome Q such that the breakpoint graph $G = G(P, Q)$ consists of $\frac{|A|}{2} + \gamma(G) = 3$ cycles. However, the direct enumeration of all possible perfect duplicated genomes Q shows that there is no breakpoint graph $G(P, Q)$ with three cycles. There exist eight distinct labeled perfect duplicated genomes Q giving rise to eight breakpoint graphs $G(P, Q)$ shown in Figure 7. All of them have less than three cycles. In the next section we explain what particular property of the genome $+a + b - a - b$ was not addressed properly in the El-Mabrouk–Sankoff analysis.

[4] While we do not explicitly consider $R \ominus R$ duplications shown in this figure (see [20] for details), our counterexample works for both $R \oplus R$ and $R \ominus R$ duplications.

4. Classification of duplicated genomes. The labeling problem can be addressed by considering *multichromosomal* genomes.[5] A multichromosomal duplicated genome is a set of circular chromosomes with every gene present in two copies. For example, Figure 4(c) presents a multichromosomal duplicated genome Q' consisting of two circular chromosomes $+a-b$ and $+a+b$, each of which forms a gray-obverse cycle. We remark that the de Bruijn graph of the genome Q' coincides with the de Bruijn graph of a unichromosomal genome $Q = +a-b+a+b$ (Figure 4(b)) and, hence, the contracted breakpoint graphs $G'(P,Q)$ and $G'(P,Q')$ are the same for any genome P (Figure 4(d)). We call genomes Q and Q' *equivalent* if their de Bruijn graphs are the same, i.e., $\hat{Q} = \hat{Q}'$.

It is easy to see that $R \oplus R$ is equivalent to $2R$ and, thus, $G'(P, R \oplus R) = G'(P, 2R)$ for any duplicated genome P. But in contrast to the breakpoint graph $G(P, R \oplus R)$ (for any labeling of P and $R \oplus R$) that contains a single gray-obverse cycle, the breakpoint graph $G(P, 2R)$ contains two gray-obverse cycles. The following theorem reveals the relationship between $G(P, R \oplus R)$ and $G(P, 2R)$.

THEOREM 5. *For any labeling of the genomes P and $2R$, there exists a labeling of the genome $R \oplus R$ such that $|c(P, R \oplus R) - c(P, 2R)| \leq 1$. Moreover, if there are two gray edges (x, y) and $(\bar{x}, \bar{y})$ belonging to the same black-gray cycle in $G(P, 2R)$, then there exists a labeling of $R \oplus R$ with $c(P, R \oplus R) \geq c(P, 2R)$.*

Proof. Let (x, y) be a gray edge in the breakpoint graph $G(P, 2R)$. Since the genome $2R$ is perfect duplicated there exists a gray edge $(\bar{x}, \bar{y})$ connecting counterparts of x and y. Define a graph H having the same vertices and edges as $G(P, 2R)$ except the gray edges (x, y) and $(\bar{x}, \bar{y})$ that are replaced with the gray edges $(x, \bar{y})$ and $(\bar{x}, y)$. Since the graph $G(P, 2R)$ consists of two gray-obverse cycles, the gray edges (x, y) and $(\bar{x}, \bar{y})$ belong to different gray-obverse cycles. Therefore, the graph H contains a single gray-obverse cycle (as well as a single black-obverse cycle inherited from $G(P, 2R)$). This implies that H is a breakpoint of the genomes P and $R \oplus R$ (i.e., $H = G(P, R \oplus R)$), where the labeling of P is the same as in $G(P, 2R)$.

If the gray edges (x, y) and $(\bar{x}, \bar{y})$ belong to the same black-gray cycle in $G(P, 2R)$, then this cycle may be split into two in H, while the other black-gray cycles are not affected. Conversely, if the gray edges (x, y) and $(\bar{x}, \bar{y})$ belong to different black-gray cycles in $G(P, 2R)$, then these cycles may be joined into a single cycle in H. In either case the difference $|c(P, R \oplus R) - c(P, 2R)|$ does not exceed 1. ∎

We redefine the notion of parity of a genome P in terms of the de Bruijn graph $\hat{P}$. A genome P is called *singular* if all black cycles in $\hat{P}$ are even. For a nonsingular genome P, define parity$(P) = \infty$. For a singular genome P, we clockwise label edges of each black cycle in $\hat{P}$ with alternating numbers $\{0, 1\}$ so that every two adjacent edges are labeled differently (Figure 8(a)). Labels of black edges in cycle P classify obverse edges in P into two classes: *even* if its flanking black edges have the same labels, and *odd* if its flanking black edges have different labels (Figure 8(b)). Let m_{even} and m_{odd} be the number of even/odd obverse edges in P correspondingly. Obviously, both m_{even} and m_{odd} are even numbers. We define parity$(P) = m_{odd}/2 \bmod 2$.

This definition of the parity index coincides with the one given in the introduction. To establish a correspondence between them one can consider a genome P as a black-obverse cycle and contract each black edge into a single vertex that inherits the label from the black edge. Since every pair of adjacent black edges of $\hat{P}$ is labeled differently, every pair of counterpart vertices is labeled differently as well. This implies that two-

[5] We emphasize that in this paper we consider only the genome halving problem for unichromosomal genomes and use multichromosomal genomes only to prove some auxiliary results.

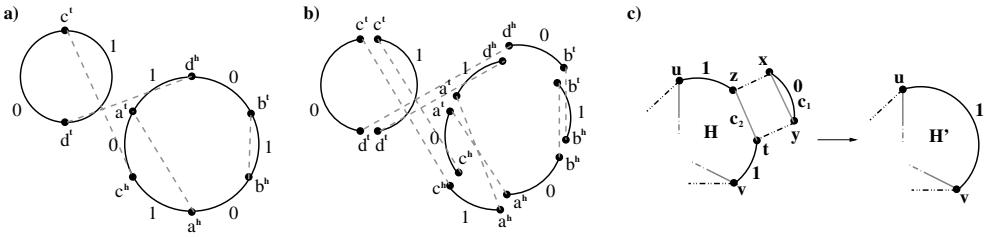


FIG. 8. *For the genome* $P = +a-b-b-d+c-a-d+c$, (a) 01*-labeling of the de Bruijn graph* $\hat{P}$; (b) *induced labeling of black-obverse cycle* P *with* $m_{odd} = 4$ *and* $m_{even} = 4$; (c) *transformation of the graph* H *into* H' *by removing vertices* x, y, z, t *and incident edges and adding a black edge* (u, v) *labeled the same as* (u, x) *and* (v, t).

digit labels of every pair of obverse edges are inversions of each other.

THEOREM 6. *The parity index of a singular genome is well defined.*

Proof. Let P be a singular genome. If $\hat{P}$ has k black cycles, then there are 2^k different 01-labelings of its black edges (two possible labelings per cycle). Therefore, it is sufficient to show that a change of 01-labeling of a particular black cycle c does not affect parity(P).

Let m^c_{even} and m^c_{odd} be the number of even/odd obverse edges in cycle P connecting black edges of c with black edges outside c. Since double obverse edges form a matching in the de Bruijn graph $\hat{P}$, the total number of double obverse edges connecting c with other black cycles is even and, thus, $m^c_{even} + m^c_{odd}$ is a multiple of 4.

Changing the 01-labeling of the black cycle c reverses the labels $0 \leftrightarrow 1$ in c. Reversed labeling of c does not change parity of obverse edges connecting two black edges in c (since both endpoint labels change) or two black edges outside of c (since neither of the endpoint labels changes). At the same time, each of the $m^c_{even} + m^c_{odd}$ obverse edges connecting black edges in c with black edges outside of c changes its parity (i.e., even edges become odd and vice versa). Then m_{odd} changes as follows:

$$m'_{odd} = m_{odd} - m^c_{odd} + m^c_{even} = m_{odd} - (m^c_{odd} + m^c_{even}) + 2m^c_{even}.$$

Since both $m^c_{odd} + m^c_{even}$ and $2m^c_{even}$ are multiples of 4, the parity of $m'_{odd}/2$ and $m_{odd}/2$ is the same, implying that parity(P) is well defined. □

Our goal is to prove the following theorem.

THEOREM 7. *For a duplicated genome* P,

$$\max_R c(P, R \oplus R) = \begin{cases} |P|/2 + b_e(P) & \text{if parity}(P) \neq 0, \\ |P|/2 + b_e(P) - 1 & \text{otherwise.} \end{cases}$$

The proof of Theorem 7 is split into two cases depending on whether P is singular or nonsingular.

THEOREM 8. *For a nonsingular genome* P, $\max_R c(P, R \oplus R) = |P|/2 + b_e(P)$.

Proof. If P is a nonsingular genome, then $\hat{P}$ has an odd black cycle. According to Theorem 4 there exists a perfect duplicated genome $R \oplus R$ such that $c_{max}(P, R \oplus R) = |P|/2 + b_e(P)$. Theorem 2 ensures that the maximum cycle decomposition of the contracted breakpoint graph $G'(P, R \oplus R)$ is induced by a labeling of either $R \oplus R$ or $2R$. If it is $R \oplus R$, then the theorem holds. Otherwise, consider a paired component in $G'(P, R \oplus R)$ (which exists since $\hat{P}$ has an odd black cycle) and an interedge e in it. Let (x, y) and $(\bar{x}, \bar{y})$ be gray edges in $G(P, 2R)$ corresponding to the interedge e in $G'(P, R \oplus R)$. Since e is the only bridge between two different black cycles (Theorem 4)

in $G'(P, R \oplus R)$, the gray edges (x, y) and $(\bar{x}, \bar{y})$ must belong to the same black-gray cycle in $G(P, 2R)$. Applying Theorem 5 to these gray edges, we obtain a labeled genome $R \oplus R$ with $c(P, R \oplus R) = c(P, 2R) = |P|/2 + b_e(P)$. □

For a singular genome P, we first fix some alternating 01-labeling of black edges in every black cycle of $\hat{P}$. The labeling of edges imposes labeling of vertices of any breakpoint graph $G(P, Q)$ (for any genome Q) so that each vertex inherits a label from an incident black edge. Note that every pair of counterpart vertices get different labels, as their incident black edges are adjacent in $\hat{P}$. A labeling of vertices of $G(P, Q)$ is called *uniform* if endpoints of every gray edge have identical labels (i.e., every gray edge is even). We will need the following theorem.

THEOREM 9. *Let P be a singular genome and Q be a perfect duplicated genome with $c(P, Q) = |P|/2 + b_e(P)$. Then every alternating* 01*-labeling of $\hat{P}$ imposes a uniform labeling on vertices of $G(P, Q)$.*

While the definition of the breakpoint graph does not explicitly specify the counterpart edges, one can derive them for $G(P, Q)$ in Theorem 9 from the vertex labels. Also, it is easy to see that gray and counterpart edges in $G(P, Q)$ form cycles of length 4 as soon as Q is a perfect duplicated genome. We take the liberty of restating the condition $c(P, Q) = |P|/2 + b_e(P)$ as $c_{bg}(G) = n + c_{bc}(G)$, where $c_{bg}(G)$ is the number of black-gray edges in G, n is the number of unique genes in P, and $c_{bc}(G)$ is the number of black-counterpart cycles in G. Also, every alternating 01-labeling of $\hat{P}$ corresponds to an alternating labeling of black edges within black-counterpart cycles. This leads to the following reformulation of Theorem 9.

THEOREM 10. *Let H be a graph on $4n$ vertices consisting of three perfect matchings black, gray, and counterpart such that* (i) *gray and counterpart matchings form cycles of length* 4*, and* (ii) *$c_{bg}(H) = n + c_{bc}(H)$. Then every alternating* 01*-labeling of black edges within black-counterpart cycles imposes a uniform labeling on vertices of H.*

Proof. The proof is done by induction on n. If $n = 1$, then the graph H consists of a gray-counterpart cycle with two black edges parallel to the gray edges, and the theorem holds. Assume that the theorem holds for graphs with less than $4n$ vertices.

Since H has $2n$ black edges and $c_{bg}(H) = n + c_{bc}(H) > n$, the pigeonhole principle implies that there exists a black-gray cycle c_1 of length 2 in H. Let $e_1 = (x, y)$ be a gray edge in the cycle c_1 (thus, e_1 is even) and let (x, z) and (y, t) be adjacent counterpart edges. Then there is a gray edge $e_2 = (z, t)$ belonging to the same gray-counterpart cycle as e_1. Let c_2 be a black-gray cycle c_2 containing the gray edge e_2.

If the cycle c_2 has length 2, then the endpoints of e_2 have identical labels. In this case we define a new graph H' as the graph H without vertices x, y, z, t and all incident edges. It is easy to see that H' is a graph on $4(n-1)$ vertices satisfying the conditions of the theorem. Indeed, the number of black-gray cycles in H' is reduced by 2 and the number of black-counterpart cycles is reduced by 1 (as compared to H), i.e., $c_{bg}(H') = c_{bg}(H) - 2$ and $c_{bc}(H') = c_{bc}(H) - 1$. Therefore, $c_{bg}(H') = (n-1) + c_{bc}(H')$. By the induction assumption, every alternating 01-labeling of H' imposes uniform labeling on vertices of H'. It implies that every alternating 01-labeling of H imposes uniform labeling on vertices of H.

If the cycle c_2 has length greater than 2, let (u, z) and (t, v) be black edges adjacent to e_2. These black edges are neighbors of the black edge (x, y) on a black-counterpart cycle (passing through the vertices u, z, x, y, t, v), so they have the same label l which is different from the label of (x, y). Therefore, the endpoints of the gray edge e_2 have identical labels. We define a new graph H' as the graph H with vertices x, y, z, t and all incident edges removed but with a single black edge (u, v) labeled l added

(Figure 8(c)). The graph H' has $4(n-1)$ vertices, $c_{bc}(H') = c_{bc}(H)$ black-counterpart cycles, and $c_{bg}(H') = c_{bg}(H) - 1$ black-gray cycles; thus, $c_{bg}(H') = n - 1 + c_{bc}(H')$ and the induction applies. □

To complete the proof of Theorem 7 we need one more theorem.

THEOREM 11. *For a singular genome P and a perfect duplicated genome Q with $c(P,Q) = |P|/2 + b_e(P)$,*

- *$Q = R \oplus R$ if and only if parity$(P) = 1$;*
- *$Q = 2R$ if and only if parity$(P) = 0$.*

Proof. According to Theorem 2, the graph $G(P,Q)$ has either a single gray-obverse cycle (i.e., $Q = R \oplus R$) or two symmetric gray-obverse cycles (i.e., $Q = 2R$). Theorem 9 implies that all gray edges in G are even (i.e., have identically labeled endpoints) for every alternating 01-labeling of black edges of P.

Case 1. Graph G has a single gray-obverse cycle c. Consider an arbitrary vertex v in G and its counterpart $\overline{v}$. Vertices v and $\overline{v}$ break c into two paths: c' (from v to $\overline{v}$) and c'' (from $\overline{v}$ to v). For every path (cycle) c denote c_{odd} as the number of odd obverse edges in c. Note that obverse edges are evenly divided between c' and c'', i.e., for every pair of obverse edges connecting counterpart vertices, one edge belongs to c' and the other edge belongs to c''. Therefore, $c'_{odd} = c''_{odd}$. Note that the start (vertex v) and the end (vertex $\overline{v}$) vertices of path c' are labeled differently. Since the total number of odd edges is odd for every path with differently labeled ends, and since all gray edges are even (Theorem 9), the total number of odd obverse edges in the path c' is odd. Therefore, $c_{odd}/2 = c'_{odd}$ is odd, implying that parity$(P) = 1$.

Case 2. Graph G has two gray-obverse cycles c' and c''. Note that obverse edges are evenly divided between c' and c''; i.e., for every pair of obverse edges connecting counterpart vertices, one edge belongs to c' and the other edge belongs to c''. Therefore, $c'_{odd} = c''_{odd}$. Since the total number of odd edges in every cycle is even, and since all gray edges are even (Theorem 9), the total number of odd obverse edges in every cycle is even. Since c'_{odd} is even, the overall number of odd obverse edges is a multiple of 4, implying that parity$(P) = 0$. □

For a singular genome P with parity$(P) = 1$ Theorem 11 implies Theorem 7, while for a singular genome P with parity$(P) = 0$ it implies that there is no genome R for which $c(P, R \oplus R) = |P|/2 + b_e(P)$. In the latter case, there exists a genome R and a labeling of P and $2R$ for which $c(P, 2R) = |P|/2 + b_e(P)$ (Theorem 4). The genome $2R$ can be transformed into a labeled genome $R \oplus R$ with $c(P, R \oplus R) = c(P, 2R) - 1 = |P|/2 + b_e(P) - 1$ (Theorem 5). This completes the proof of Theorem 7.

5. Genome halving algorithm. The classification of circular genomes leads to the following algorithm for the weak genome halving problem.[6]

1. For a given duplicated genome P, find a perfect duplicated genome $R \oplus R$ such that $c_{max}(P, R \oplus R) = |P|/2 + b_e(P)$ (Theorem 4) and decompose $G'(P, R \oplus R)$ into the maximum number of black-gray cycles [1].
2. Find a labeling of the genomes P and Q ($Q = R \oplus R$ or $Q = 2R$) and a breakpoint graph $G(P,Q)$ inducing the maximum black-gray cycle decomposition of $G'(P, R \oplus R)$ (Theorem 2).
3. If $Q = R \oplus R$, then output the breakpoint graph $G(P, R \oplus R)$.
4. If $Q = 2R$ and P is nonsingular, then there is a paired component in $G'(P, R \oplus R)$ with a single interedge (Theorem 4) that corresponds to two gray

[6] The algorithm outputs the breakpoint graph $G(P, R \oplus R)$ (in addition to the preduplicated genome R). This allows one to reconstruct a sequence of reversals transforming $R \oplus R$ into P with the reversal distance algorithm.

edges (x, y) and $(\bar{x}, \bar{y})$ in $G(P, 2R)$. Find a labeling of the genome $R \oplus R$ for which $c(P, R \oplus R) = c(P, 2R)$ (Theorems 5 and 8) and output $G(P, R \oplus R)$.

5. If $Q = 2R$ and P is singular, then parity$(P) = 0$ (Theorem 11). Find a labeling of the genome $R \oplus R$ for which $c(P, R \oplus R) = c(P, 2R) - 1$ (Theorem 5) and output $G(P, R \oplus R)$.

We illustrate the algorithm for a genome $P = +a + b - a - b$ (assuming a fixed labeling of P as $(+a_1 + b_1 - a_2 - b_2)$) using Figure 5. At step 1, we use the algorithm from [1] to construct $R = +a + b$ such that the contracted breakpoint graph $G'(P, R \oplus R)$ (Figure 5(a)) has a black-gray cycle decomposition with $|P|/2 + b_e(P) = 3$ cycles (Figure 5(b)). At step 2, we find a breakpoint graph $G(P, Q)$ (Figure 5(c)) inducing this cycle decomposition (see [1]). The breakpoint graph $G(P, Q)$ has two gray-obverse cycles (Figure 5(c)), thus implying that $Q = 2R$. Since the genome P is singular, we proceed to step 5 and perform a transformation of $Q = 2R$ into a new genome, as described in Theorem 5. We first find a pair of gray edges from two different cycles in $2R$ as described in Theorem 5, for example, a pair of edges labeled (a^h, b^t) in Figure 5(c). Afterwards, we replace these edges with a new pair of gray edges also labeled (a^h, b^t), as shown in Figure 5(d). This operation transforms two cycles in the genome Q into a single cycle (unichromosomal genome) that we represent as $R \oplus R$. According to Theorems 5 and 7, this genome represents a solution to the weak genome halving problem.

The first two steps of the genome halving algorithm can be implemented in $O(|P|^2)$ time (see [1]) while the remaining steps fit this time bound as well. In practice, our genome halving software takes less than a second to halve a "random" duplicated genome with 1000 unique genes on a standard Intel PIII-900MHz CPU.

6. Conclusion. While whole genome duplications in multichromosomal genomes are well established, there are relatively few examples of whole genome duplications in unichromosomal genomes. Undoubtedly, bacterial genomes have undergone a large number of duplications, but it is difficult to distinguish between partial and whole genome duplication scenarios in the case of these genomes (Coissac, Maillier, and Netter [13]). Matters are further complicated by the fact that even a few rearrangements can quickly "randomize" gene orders in bacterial genomes (due to a relatively small number of genes).

Recently, Sugaya et al. [38] studied *Cyanobacterium Anabaena* and came to the conclusion that it has undergone whole genome duplications rather than a series of (tandem) segmental duplications. Indeed, the arrangement of genes in *Cyanobacterium Anabaena* points to whole genome duplications as the most likely scenario (it also has an unusually large genome as compared to other organisms in the *Cyanobacteria* phylum). Also, the recent discovery and sequencing of *Acanthamoeba polyphaga* (the largest known virus to date) revealed an unusually large number of duplicated regions that point to a large duplication event (Suhre [39]). While it remains unclear whether this large segmental duplication represents whole genome duplications or extremely large partial duplication(s), one can argue that it is only a matter of time until ongoing sequencing efforts will reveal traces of whole genome duplications in many unichromosomal (bacterial and viral) genomes.

These recent discoveries raise a new algorithmic challenge that we refer to as the partial genome duplication problem. Let R be a genome with n unique genes (represented as a signed permutation) and R' be a set of m consecutive genes in this genome. We define $R \oplus R'$ as a *perfect partially duplicated genome*. Let P be a genome with $n + m$ genes in which m genes from R' are duplicated and the remaining

$n - m$ genes are unique. Given a genome P, the partial genome duplication problem is to find a perfect partially duplicated genome $R \oplus R'$ such that the reversal distance between $R \oplus R'$ and P is minimal.

Acknowledgments. We are grateful to Mohan Paturi and Dekel Tsur for many insightful comments.

REFERENCES

[1] M. A. Alekseyev and P. A. Pevzner, *Colored de Bruijn graphs and the genome halving problem*, IEEE/ACM Trans. Comput. Biol. Bioinformatics, 4 (2007), pp. 98–107.

[2] M. A. Alekseyev and P. A. Pevzner, *Whole genome duplications, multi-break rearrangements, and genome halving problem*, in Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, Philadelphia, ACM, New York, 2007, pp. 665–679.

[3] D. A. Bader, B. M. E. Moret, and M. Yan, *A linear-time algorithm for computing inversion distances between signed permutations with an experimental study*, J. Comput. Biol., 8 (2001), pp. 483–491.

[4] V. Bafna and P. A. Pevzner, *Genome rearrangements and sorting by reversals*, SIAM J. Comput., 25 (1996), pp. 272–289.

[5] E. Belda, A. Moya, and F. J. Silva, *Genome rearrangement distances and gene order phylogeny in γ-proteobacteria*, Mol. Biol. Evol., 22 (2005), pp. 1456–1467.

[6] A. Bergeron, *A very elementary presentation of the Hannenhalli–Pevzner theory*, in Combinatorial Pattern Matching, Lecture Notes in Comput. Sci. 2089, Springer, Berlin, 2001, pp. 106–117.

[7] A. Bergeron, J. Mixtacki, and J. Stoye, *Reversal distance without hurdles and fortresses*, in Combinatorial Pattern Matching, Lecture Notes in Comput. Sci. 3109, Springer, Berlin, 2004, pp. 388–399.

[8] G. Bourque, P. A. Pevzner, and G. Tesler, *Reconstructing the genomic architecture of ancestral mammals: Lessons from human, mouse, and rat genomes*, Genome Res., 14 (2004), pp. 507–516.

[9] G. Bourque, Y. Yacef, and N. El-Mabrouk, *Maximizing synteny blocks to identify ancestral homologs*, in Comparative Genomics, Lecture Notes in Comput. Sci. 3678, Springer, Berlin, 2005, pp. 21–34.

[10] G. Bourque, E. M. Zdobnov, P. Bork, P. A. Pevzner, and G. Tesler, *Comparative architectures of mammalian and chicken genomes reveal highly variable rates of genomic rearrangements across different lineages*, Genome Res., 15 (2005), pp. 98–110.

[11] X. Chen, J. Zheng, P. Nan, Z. Fu, Y. Zhong, S. Lonardi, and T. Jiang, *Computing the assignment of orthologous genes via genome rearrangement*, in Proceedings of the 3rd Asia Pacific Bioinformatics Conference, Imperial College Press, London, 2005, pp. 363–378.

[12] A. Christoffels, E. G. L. Koh, J. Chia, S. Brenner, S. Aparicio, and B. Venkatesh, *Fugu genome analysis provides evidence for a whole-genome duplication early during the evolution of ray-finned fishes*, Mol. Biol. Evol., 21 (2004), pp. 1146–1151.

[13] E. Coissac, E. Maillier, and P. Netter, *A comparative study of duplications in bacteria and eukaryotes: The importance of telomeres*, Mol. Biol. Evol., 14 (1997), pp. 1062–1074.

[14] P. Dehal and J. L. Boore, *Two rounds of genome duplication in the ancestral vertebrate genome*, PLoS Biol., 3 (2005), e314.

[15] F. S. Dietrich, S. Voegeli, S. Brachat, A. Lerch, K. Gates, S. Steiner, C. Mohr, R. Pöhlmann, P. Luedi, S. Choi, R. A. Wing, A. Flavier, T. D. Gaffney, and P. Philippsen, *The* Ashbya gossypii *genome as a tool for mapping the ancient* Saccharomyces cerevisiae *genome*, Science, 304 (2004), pp. 304–307.

[16] N. El-Mabrouk, *Genome rearrangement by reversals and insertions/deletions of contiguous segments*, in Combinatorial Pattern Matching, Lecture Notes in Comput. Sci. 1848, Springer, Berlin, 2000, pp. 222–234.

[17] N. El-Mabrouk, D. Bryant, and D. Sankoff, *Reconstructing the pre-doubling genome*, in Proceedings of the Third Annual International Conference on Computational Molecular Biology (RECOMB), ACM, New York, 1999, pp. 154–163.

[18] N. El-Mabrouk, J. H. Nadeau, and D. Sankoff, *Genome halving*, in Combinatorial Pattern Matching, Lecture Notes in Comput. Sci. 1448, Springer, Berlin, 1998, pp. 235–250.

[19] N. El-Mabrouk and D. Sankoff, *On the reconstruction of ancient doubled circular genomes*, Genome Informatics, 10 (1999), pp. 83–93.

[20] N. El-Mabrouk and D. Sankoff, *The reconstruction of doubled genomes*, SIAM J. Comput., 32 (2003), pp. 754–792.
[21] R. Guyot and B. Keller, *Ancestral genome duplication in rice*, Genome, 47 (2004), pp. 610–614.
[22] S. Hannenhalli and P. Pevzner, *Transforming men into mouse (polynomial algorithm for genomic distance problem)*, in Proceedings of the 36th Annual Symposium on Foundations of Computer Science, 1995, IEEE, Piscataway, NJ, pp. 581–592.
[23] S. Hannenhalli and P. Pevzner, *Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals)*, J. ACM, 46 (1999), pp. 1–27.
[24] O. Jaillon et al., *Genome duplication in the teleost fish* Tetraodon nigroviridis *reveals the early vertebrate proto-karyotype*, Nature, 431 (2004), pp. 946–957.
[25] H. Kaplan, R. Shamir, and R. E. Tarjan, *A faster and simpler algorithm for sorting signed permutations by reversals*, SIAM J. Comput., 29 (1999), pp. 880–892.
[26] H. Kaplan and E. Verbin, *Sorting signed permutations by reversals, revisited*, J. Comput. System Sci., 70 (2005), pp. 321–341.
[27] M. Kellis, B. W. Birren, and E. S. Lander, *Proof and evolutionary analysis of ancient genome duplication in the yeast* Saccharomyces cerevisiae, Nature, 428 (2004), pp. 617–624.
[28] A. Meyer and Y. Van de Peer, *From* $2R$ *to* $3R$*: Evidence for a fish-specific genome duplication (FSGD)*, BioEssays, 27 (2005), pp. 937–945.
[29] W. J. Murphy, G. Bourque, G. Tesler, P. Pevzner, and S. J. O'Brien, *Reconstructing the genomic architecture of mammalian ancestors using multispecies comparative maps*, Human Genomics, 1 (2003), pp. 30–40.
[30] W. J. Murphy, D. M. Larkin, A. Everts van der Wind, G. Bourque, G. Tesler, L. Auvil, J. E. Beever, B. P. Chowdhary, F. Galibert, L. Gatzke, C. Hitte, C. N. Meyers, D. Milan, E. A. Ostrander, G. Pape, H. G. Parker, T. Raudsepp, M. B. Rogatcheva, L. B. Schook, L. C. Skow, M. Welge, J. E. Womack, S. J. O'Brien, P. A. Pevzner, and H. A. Lewin, *Dynamics of mammalian chromosome evolution inferred from multispecies comparative map*, Science, 309 (2005), pp. 613–617.
[31] S. Ohno, *Evolution by Gene Duplication*, Springer, Berlin, 1970.
[32] P. Pevzner, H. Tang, and G. Tesler, *De novo repeat classification and fragment assembly*, Genome Res., 14 (2004), pp. 1786–1796.
[33] P. Pevzner and G. Tesler, *Genome rearrangements in mammalian evolution: Lessons from human and mouse genomes*, Genome Res., 13 (2003), pp. 37–45.
[34] P. A. Pevzner, *Computational Molecular Biology: An Algorithmic Approach*, The MIT Press, Cambridge, MA, 2000.
[35] P. A. Pevzner and G. Tesler, *Human and mouse genomic sequences reveal extensive breakpoint reuse in mammalian evolution*, Proc. Nat. Acad. Sci., 100 (2003), pp. 7672–7677.
[36] M. Robinson-Rechavi, O. Marchand, H. Escriva, and V. Laudet, *An ancestral whole-genome duplication may not have been responsible for the abundance of duplicated fish genes*, Curr. Biol., 11 (2001), pp. 458–459.
[37] D. Sankoff, *Genome rearrangement with gene families*, Bioinformatics, 15 (1999), pp. 909–917.
[38] N. Sugaya, M. Sato, H. Murakami, A. Imaizumi, S. Aburatani, and K. Horimoto, *Causes for the large genome size in a Cyanobacterium* Anabaena *sp. PCC*7120, Genome Informatics, 15 (2004), pp. 229–238.
[39] K. Suhre, *Gene and genome duplication in Acanthamoeba polyphaga Mimivirus*, J. Virology, 79 (2005), pp. 14095–14101.
[40] K. M. Swenson, M. Marron, J. V. Earnest-DeYoung, and B. M. E. Moret, *Approximating the true evolutionary distance between two genomes*, in Proceedings of the Seventh Workshop on Algorithm Engineering and Experiments and the Second Workshop on Analytic Algorithmics and Combinatorics, C. Demetrescu, R. Sedgewick, and R. Tamassia, eds., SIAM, Philadelphia, 2005, pp. 121–129.
[41] K. M. Swenson, N. D. Pattengale, and B. M. E. Moret, *A framework for orthology assignment from gene rearrangement data*, in Comparative Genomics, Lecture Notes in Comput. Sci. 3678, Springer, Berlin, 2005, pp. 153–166.
[42] E. Tannier and M. F. Sagot, *Sorting by reversals in subquadratic time*, in Combinatorial Pattern Matching, Lecture Notes in Comput. Sci. 3109, Springer, Berlin, 2004, pp. 1–13.

SIAM J. COMPUT.
Vol. 36, No. 6, pp. 1764–1776

APPROXIMATING THE RADII OF POINT SETS*

KASTURI VARADARAJAN†, S. VENKATESH‡, YINYU YE§, AND JIAWEI ZHANG¶

Abstract. We consider the problem of computing the outer-radii of point sets. In this problem, we are given integers n, d, and k, where $k \leq d$, and a set P of n points in $\Re^d$. The goal is to compute the *outer k-radius* of P, denoted by $\mathcal{R}_k(P)$, which is the minimum over all $(d-k)$-dimensional flats F of $\max_{p \in P} d(p, F)$, where $d(p, F)$ is the Euclidean distance between the point p and flat F. Computing the radii of point sets is a fundamental problem in computational convexity with many significant applications. The problem admits a polynomial time algorithm when the dimension d is constant [U. Faigle, W. Kern, and M. Streng, *Math. Program.*, 73 (1996), pp. 1–5]. Here we are interested in the general case in which the dimension d is not fixed and can be as large as n, where the problem becomes NP-hard even for $k = 1$. It is known that $R_k(P)$ can be approximated in polynomial time by a factor of $(1 + \varepsilon)$ for any $\varepsilon > 0$ when $d - k$ is a fixed constant [M. Bădoiu, S. Har-Peled, and P. Indyk, in *Proceedings of the ACM Symposium on the Theory of Computing*, 2002; S. Har-Peled and K. Varadarajan, in *Proceedings of the ACM Symposium on Computing Geometry*, 2002]. A polynomial time algorithm that guarantees a factor of $O(\sqrt{\log n})$ approximation for $R_1(P)$, the width of the point set P, is implied by the results of Nemirovski, Roos, and Terlaky [*Math. Program.*, 86 (1999), pp. 463–473] and Nesterov [*Handbook of Semidefinite Programming Theory, Algorithms*, Kluwer Academic Publishers, Norwell, MA, 2000]. In this paper, we show that $R_k(P)$ can be approximated by a ratio of $O(\sqrt{\log n})$ for any $1 \leq k \leq d$, thus matching the previously best known ratio for approximating the special case $R_1(P)$, the width of point set P. Our algorithm is based on semidefinite programming relaxation with a new mixed deterministic and randomized rounding procedure. We also prove an inapproximability result that gives evidence that our approximation algorithm is doing well for a large range of k. We show that there exists a constant $\delta > 0$ such that the following holds for any $0 < \varepsilon < 1$: there is no polynomial time algorithm that approximates $R_k(P)$ within $(\log n)^\delta$ for all k such that $k \leq d - d^\varepsilon$ unless NP $\subseteq$ DTIME $[2^{(\log m)^{O(1)}}]$. Our inapproximability result for $R_k(P)$ extends a previously known hardness result of Brieden [*Discrete Comput. Geom.*, 28 (2002), pp. 201–209] and is proved by modifying Brieden's construction using basic ideas from probabilistically checkable proofs (PCP) theory.

Key words. approximation algorithms, semidefinite programming, computational convexity

AMS subject classifications. 68W20, 68W25, 68W40

DOI. 10.1137/050627472

1. Introduction.

Computing the outer k-radius of a point set is a fundamental problem in computational convexity with applications in global optimization, data

*Received by the editors March 24, 2005; accepted for publication (in revised form) November 2, 2006; published electronically March 19, 2007. A preliminary version of this paper appeared as (i) K. R. Varadarajan, S. Venkatesh, and J. Zhang, *Approximating the radii of point sets in high dimensions*, in Proceedings of the 43rd IEEE Symposium on the Foundations of Computer Science, 2002 and (ii) Y. Ye and J. Zhang, *An improved algorithm for approximating the radii of point sets*, in Proceedings of Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques (APPROX, 2003), Springer, 2003.

http://www.siam.org/journals/sicomp/36-6/62747.html

†Department of Computer Science, The University of Iowa, Iowa City, IA 52242-1419 (kvaradar@cs.uiowa.edu, www: http://www.cs.uiowa.edu/~kvaradar/). The research of this author was supported by NSF CAREER award CCR-0237431.

‡Department of Computer Science, University of Victoria, PO Box 3055, STN CSC, Victoria V8W 3P6, BC, Canada (venkat@cs.uvic.ca, www: http://www.cs.uvic.ca/~venkat). The research of this author was supported by an NSERC discovery grant.

§Management Science and Engineering and, by courtesy, Electrical Engineering, Stanford University, Stanford, CA 94305 (yinyu-ye@stanford.edu). The research of this authors was supported by NSF grant DMI-0231600.

¶IOMS-Operations Management, Stern School of Business, New York University, 44 W. 4th Street, Suite 8-66, New York, NY 10012-1126 (jzhang@stern.nyu.edu, www: http://www.stern.nyu.edu/~jzhang). The research of this authors was supported by NSF grant DMI-0231600.

mining, statistics, and clustering, and it has received considerable attention in the computational geometry literature [20, 21, 22]. In this problem, we are given integers n, d, and k, where $k \leq d$, and a set P of n points in $\Re^d$. A *flat* or *affine subspace* F in $\Re^d$ is specified by a point $q \in \Re^d$ and a linear subspace H; it is defined as $F = \{q + h | h \in H\}$. The dimension of the flat F is defined to be the dimension of the linear subspace H. For any flat F, let $R(P, F) = \max_{p \in P} d(p, F)$ denote the *radius* of the flat F with respect to P, where $d(p, F)$ is the Euclidean distance between the point p and the flat F. The goal is to compute the *outer k-radius* of P, denoted by $R_k(P)$, which is the minimum of $R(P, F)$ over all $(d-k)$-dimensional flats F. A $(d-k)$-flat is simply a flat of dimension $d-k$. Roughly speaking, the outer k-radius $R_k(P)$ measures how well the point set P can be approximated by an affine subspace of dimension $d-k$. A few special cases of $R_k(P)$ which have received particular attention include $R_1(P)$, half of the *width* of P; $R_d(P)$, the radius of the minimum enclosing ball of P; and $R_{d-1}(P)$, the radius of the minimum enclosing cylinder of P.

When the dimension d is a fixed constant, $R_k(P)$ can be computed exactly in polynomial time [15]. It is also known that $R_k(P)$ can be approximated by a factor of $(1+\varepsilon)$ for any $\varepsilon > 0$ in $O(n + f_d(\frac{1}{\varepsilon}))$ time [2, 6], where f_d is a polynomial for every fixed d. In this paper, we are interested in the general scenario when the dimensions k and d are not fixed and d can be as large as n.

When the dimensions k and d are part of the input, the complexity of computing/approximating $R_k(P)$ depends on the parameter $d-k$. It is well known that the problem is polynomial time solvable when $d-k=0$, i.e., the minimum enclosing ball of a set of points can be computed in polynomial time (Gritzmann and Klee [20]). Megiddo [25] shows that the problem of determining whether there is a line that intersects a set of balls is NP-hard. In his reduction, the balls have the same radius, which implies that computing the radius $R_{d-1}(P)$ of the min-enclosing cylinder of a set of points P is NP-hard. Bădoiu, Har-Peled, and Indyk [7] show that $R_{d-1}(P)$ can be approximated in polynomial time by a factor of $(1+\varepsilon)$ for any $\varepsilon > 0$. Har-Peled and Varadarajan [22, 23] generalize the result and show that $R_k(P)$ can be approximated by a factor of $(1+\varepsilon)$ for any $\varepsilon > 0$ when $d-k$ is constant.[1]

More hardness results are known when $d-k$ becomes large or when k becomes small. Bodlaender et al. [10] show that the problem is NP-hard when $k=1$. This is true even for the case $n = d+1$ [20]. Gritzmann and Klee [20] also show that it is NP-hard to compute $R_k(P)$ if $k \leq c \cdot d$ for any fixed $0 < c < 1$. These negative results are further improved by Brieden, Gritzmann, and Klee [11] and Brieden [14], the latter of which has shown that it is NP-hard to approximate $R_1(P)$, the width of a point set, to within *any* constant factor.

On the positive side, the algorithms of Nemirovski, Roos, and Terlaky [26] and Nesterov [27] imply that $R_1(P)$, or equivalently the width of the point set P, can be approximated within a factor of $O(\sqrt{\log n})$. Another algorithm for approximating the width of a point set is given by Brieden et al. [12, 13], and their algorithm has a performance guarantee $\sqrt{d/\log d}$ that is measured in the dimension d. Their algorithm in fact works for any convex body given in terms of appropriate "oracles"; the number of calls to the oracle is polynomial in the dimension d. They also show that this is the best possible result in the oracle model even if randomization is allowed. (Their algorithm actually gives a $\sqrt{d/\log n}$ approximation with poly(n) calls to the oracle, where n is the number of points in the set.) It is not clear if their algorithm can be extended to compute $R_k(P)$.

[1] Note that R_k^{opt} in [23] is the same as R_{d-k} in this paper

The problem of efficiently computing the low-rank approximation of matrices has received considerable attention recently; see [1, 5, 17] and the references cited in these papers. This problem corresponds to computing the best $(d-k)$-dimensional subspace that fits a point set, where the quality of a subspace is the *sum* of the square of the distance of each point from the flat. The problem is therefore related to the one we study in this paper, where the quality of a flat is the maximum over the point-flat distances. However, the low-rank approximation problem can be solved in polynomial time for any $1 \leq k \leq d$.

Our results and an overview. We show that $R_k(P)$ can be approximated in polynomial time by a factor of $O(\sqrt{\log n})$ for all $1 \leq k \leq d$, thereby generalizing the result of Nemirovski, Roos, and Terlaky [26] to all values of k. Our algorithm is based on a semidefinite programming (SDP) relaxation with a mixed deterministic and randomized rounding procedure, in contrast to all other purely randomized rounding procedures used for semidefinite programming approximation.

Generally speaking, the problem of computing $R_k(P)$ can be formulated as a quadratic minimization problem. SDP problems (where the unknowns are represented by positive semidefinite matrices) have recently been developed for approximating such problems; see, for example, Goemans and Williamson [18]. In the case of $k = 1$, computing $R_1(P)$ corresponds to a SDP problem plus an additional requirement that the rank of the unknown matrix equals 1. Removing the rank requirement, the SDP problem becomes a relaxation of the original problem and can be solved within any given accuracy $\epsilon > 0$ in time polynomial in $\ln \frac{1}{\epsilon}$ and the dimension of the data specifying the problem. Once obtaining an optimal solution, say X, of the SDP relaxation, one would like to generate a rank-1 matrix, say $\hat{X} = yy^T$, from X, where y is a column vector and serves as a solution to the original problem. Such rank reduction is called "rounding." Many rounding procedures are proposed, and almost all of them are randomized; see, for example, [9].

One particular procedure has been proposed by Nemirovski, Roos, and Terlaky [26] which can be used for approximating $R_1(P)$. Their procedure is a simple randomized rounding that can be described as follows: an optimal solution X of the SDP relaxation, whose rank could be as large as d, can be represented as (for example, by eigenvector decomposition)

$$X = \lambda_1 v_1 v_1^T + \lambda_2 v_2 v_2^T + \cdots + \lambda_d v_d v_d^T.$$

Then one can generate a single vector y by taking a random linear combination of the vectors $\sqrt{\lambda_1} v_1, \sqrt{\lambda_2} v_2, \ldots, \sqrt{\lambda_d} v_d$, where the coefficients of the combination take values of -1 or 1 uniformly and independently.

For the case $k \geq 2$, the SDP relaxation that we describe is best viewed as a direct relaxation of the problem of computing $R_k(P)$, rather than one that is obtained via a quadratic program formulation of $R_k(P)$. We then need to generate k rank-1 matrices from X, the optimal solution of the SDP relaxation, such that

$$\hat{X} = \sum_{i=1}^{k} y_i y_i^T,$$

where y_is are orthogonal to each other. Our rounding procedure works as follows: having obtained an optimal solution for the SDP relaxation with

$$X = \lambda_1 v_1 v_1^T + \lambda_2 v_2 v_2^T + \cdots + \lambda_d v_d v_d^T,$$

we deterministically partition the vectors $v_1, v_2, \ldots, v_d$ into k groups where group j may contain n_j vectors and each group can be seen as a single semidefinite matrix with rank n_j. We then generate one vector from each group using the randomized rounding procedure similar to that of Nemirovski, Roos, and Terlaky [26]. The k vectors generated by this rounding procedure will automatically satisfy the condition that any pair of them must be orthogonal to each other. We then manage to show that the quality of these vectors yields an approximation ratio of no more than $O(\sqrt{\log n})$.

We also prove an inapproximability result that gives evidence that our approximation algorithm is close to the best possible for a large range of k. We show that there exists a constant $\delta > 0$ such that the following holds for any $0 < \varepsilon < 1$: there is no polynomial time algorithm that approximates $R_k(P)$ within $(\log n)^\delta$ for all k such that $k \leq d - d^\varepsilon$ unless NP $\subseteq \tilde{P}$. $\tilde{P}$ denotes the complexity class DTIME$[2^{(\log m)^{O(1)}}]$, which is sometimes referred to as deterministic quasi-polynomial time. That is, $\tilde{P}$ contains the set of all problems for which there is an algorithm that runs in time $2^{(\log m)^{O(1)}}$ on inputs of size m.

To prove the lower bound result, we start with a two-prover protocol for 3SAT in which the verifier has very low error probability. Such a protocol is obtained as a consequence of the probabilistically checkable proofs (PCP) theorem of Arora et al. [3], and Arora and Safra [4] and the parallel repetition theorem of Raz [28]. The construction of Brieden [14] then implies a reduction from Max-3SAT to width computation such that the ratio of the width of point sets that correspond to satisfiable instances to those that correspond to unsatisfiable instances is large. This separation gives us the inapproximability result for the width. This result can then be extended to an inapproximability result for $R_k(P)$ for a large range of k.

The remainder of this paper is organized as follows: in section 2, we present our algorithm for approximating the outer k-radius $R_k(P)$ of a point set P. In section 3, we describe our inapproximability results. We make some concluding remarks in section 4.

2. Approximating the radius. We now present the quadratic program formulation of the outer k-radius problem and its SDP relaxation. It will be helpful to first introduce some notation that will be used later. The trace of a given square matrix A, denoted by $\mathrm{Tr}(A)$, is the sum of the entries on the main diagonal of A. We use I to denote the identity matrix whose dimension will be clear in the context. The inner product of two vectors p and q is denoted by $\langle p, q\rangle$. The 2-norm of a vector x, denoted by $\|x\|$, is defined by $\sqrt{\langle x, x\rangle}$. For a matrix X, we use the notation $X \succeq 0$ to mean that X is a positive semidefinite matrix. For simplicity, we assume that P is symmetric in the sense that if $p \in P$, then $-p \in P$. This is without loss of generality for the following reason: we may, by performing a translation if necessary, assume that $0 \in P$. Denote the set $\{-p | p \in P\}$ by $-P$, and let $Q = P \cup -P$. It is clear that $R_k(P) \leq R_k(Q) \leq 2R_k(P)$. Therefore, if we found a good approximation for $R_k(Q)$, then it must also be a good approximation for $R_k(P)$.

Since P is a symmetric point set, the best $(d-k)$-flat for P contains the origin so that it is a subspace. Thus, the square of $R_k(P)$ can be defined by the optimal value of the following quadratic minimization problem:

$$
\begin{array}{rlll}
R_k(P)^2 := & \text{Minimize} & \alpha & \\
& \text{Subject to} & \sum_{i=1}^{k} \langle p, x_i\rangle^2 \leq \alpha \ \forall p \in P, & \\
& & \|x_i\|^2 = 1, \ i = 1, \ldots, k, & \\
& & \langle x_i, x_j\rangle = 0 \ \forall i \neq j. &
\end{array} \tag{1}
$$

Assume that $x_1, x_2, \ldots, x_k \in \Re^d$ is the optimal solution of (1). Then one can easily verify that the matrix $X = x_1x_1^T + x_2x_2^T + \cdots + x_kx_k^T$ is a feasible solution for the following semidefinite program:

$$\begin{array}{rll} \alpha_k^* := & \text{Minimize} & \alpha \\ & \text{Subject to} & \mathrm{Tr}(pp^TX)\ (= p^TXp) \le \alpha\ \forall p \in P, \\ & & \mathrm{Tr}(X) = k, \\ & & I - X \succeq 0,\ X \succeq 0. \end{array} \tag{2}$$

It follows that $\alpha_k^* \le R_k(P)^2$. The following lemma follows from the above observations.

LEMMA 1. *There exists an integer $r \ge k$ such that we can compute, in polynomial time, r nonnegative reals $\lambda_1, \lambda_2, \ldots, \lambda_r$ and r orthogonal unit vectors $v_1, v_2, \ldots, v_r$ such that*

1. $\sum_{i=1}^r \lambda_i = k$.
2. $\max_{1 \le i \le r} \lambda_i \le 1$.
3. $\sum_{i=1}^r \lambda_i \langle p, v_i \rangle^2 \le R_k(P)^2$ *for any* $p \in P$.

Proof. We solve the semidefinite program (2) and let X^* be an optimal solution of (2). We claim that the rank of X^*, say r, is at least k. This follows from the fact that $\mathrm{Tr}(X^*) = k$ and $I - X^* \succeq 0$. In other words, $\mathrm{Tr}(X^*) = k$ implies that the sum of the eigenvalues of X^* is equal to k, and $I - X^* \succeq 0$ implies that all the eigenvalues are less than or equal to 1. Therefore, X^* has at least k nonzero eigenvalues, which implies that the rank of X^* is at least k. Let $\lambda_1, \lambda_2, \ldots, \lambda_r$ be the r nonnegative eigenvalues and $v_1, v_2, \ldots, v_r$ be the corresponding eigenvectors (see [26, p. 466] for details on computing the eigenvalues and eigenvectors in polynomial time). Then we have $\sum_{i=1}^r \lambda_i = k$ and $\max_{1 \le i \le r} \lambda_i \le 1$. Furthermore, for any $p \in P$,

$$\sum_{i=1}^r \lambda_i \langle p, v_i \rangle^2 = \mathrm{Tr}(pp^T \sum_{i=1}^r \lambda_i v_i v_i^T) = \mathrm{Tr}(pp^TX^*) \le \alpha_k^* \le R_k(P)^2. \qquad \square$$

2.1. Deterministic first rounding. In this section, we prove a lemma concerning how to deterministically group the eigenvalues and their eigenvectors. The proof of the lemma is elementary, but it plays an important role for proving our main result.

LEMMA 2. *The index set $\{1, 2, \ldots, r\}$ can be partitioned into k sets $I_1, I_2, \ldots, I_k$ such that, for any $i : 1 \le i \le k$, $\sum_{j \in I_i} \lambda_j \ge \frac{1}{2}$.*

Proof. Recall that $\sum_{j=1}^r \lambda_j = k$ and $0 \le \lambda_j \le 1$ for all j. Without loss of generality, we can assume that $\lambda_1 \ge \lambda_2 \ge \cdots \ge \lambda_r$. Our partitioning algorithm is the same as the longest-processing-time heuristic algorithm for the parallel machine scheduling problem. The algorithm works as follows:

1. For $i = 1, 2, \ldots, k$, set $I_i = \emptyset$, and let $L_i = 0$. Let $I = \{1, 2, \ldots, r\}$.
2. While $I \ne \emptyset$,
   choose j from I with the smallest index;
   choose set i with the smallest value L_i.
   Let $I_i := I_i \cup \{j\}$, $L_i := L_i + \lambda_j$, and $I := I - \{j\}$.

It is clear that when the algorithm stops, the sets $I_1, I_2, \ldots, I_k$ are a partition of $\{1, 2, \ldots, r\}$. Now we prove the lemma by contradiction. Assume that there exists some t such that $\sum_{j \in I_t} \lambda_j < \frac{1}{2}$.

We now claim that, for all i, $\sum_{j \in I_i} \lambda_j \le 1$. Otherwise, suppose $\sum_{j \in I_{t'}} \lambda_j > 1$ for some t'. Note that $\lambda_j \le 1$, for every j, and thus there are at least two eigenvalues

assigned to $I_{t'}$. Denote the last element within $I_{t'}$ by s'. It follows that $\sum_{j\in I_{t'}} \lambda_j - \lambda_{s'} = \sum_{j\in I_{t'}\setminus\{s'\}} \lambda_j \le \sum_{j\in I_t} \lambda_j$ since, otherwise, we would have not assigned $\lambda_{s'}$ to $I_{t'}$ in the algorithm. However, since $\sum_{j\in I_t} \lambda_j < \frac{1}{2}$, we must have $\sum_{j\in I_{t'}} \lambda_j - \lambda_{s'} = \sum_{j\in I_{t'}\setminus\{s'\}} \lambda_j < \frac{1}{2}$. Thus, $\lambda_{s'} > \sum_{j\in I_{t'}} \lambda_j - \frac{1}{2} > \frac{1}{2}$. This is impossible since $\lambda_{s'}$ is the last eigenvalue assigned to $I_{t'}$, which implies $\lambda_{s'} \le \lambda_j$ for every $j \in I_{t'}$, and we have already proved that there must exist an l such that $s' \ne l \in I_{t'}$ and $\lambda_l \le \sum_{j\in I_{t'}\setminus\{s'\}} \lambda_j < \frac{1}{2}$. Therefore, $\sum_{j\in I_i} \lambda_j \le 1$ for all i, and in particular $\sum_{j\in I_t} \lambda_j < \frac{1}{2}$. It follows that $\sum_{i=1}^{k} \sum_{j\in I_i} \lambda_j < k$. However, we know that since $I_1, I_2, \ldots, I_k$ is a partition of the index set $\{1, 2, \ldots, r\}$, $\sum_{i=1}^{k} \sum_{j\in I_i} \lambda_j = \sum_{j=1}^{r} \lambda_j = k$. This results in a contradiction. Therefore, such a t does not exist, and the proof is completed. ∎

Notice that the running time of the partitioning algorithm is bounded by $O(r\cdot k)$.[2]

2.2. Randomized second rounding. Let us now assume that we have found $I_1, I_2, \ldots, I_k$. Then our next randomized rounding procedure works as follows:

1. Generate an r-dimensional random vector ϕ such that each entry of ϕ takes value, independently, -1 or 1 with probability $\frac{1}{2}$ each way.
2. For $i = 1, 2, \ldots, k$, let

$$x_i = \frac{\sum_{j\in I_i} \phi_j \sqrt{\lambda_j}\cdot v_j}{\sqrt{\sum_{j\in I_i} \lambda_j}}.$$

The following lemmas show that $x_1, x_2, \ldots, x_k$ form a feasible solution for the original problem. In other words, they are k orthogonal unit vectors.

LEMMA 3. *For $i = 1, 2, \ldots, k$, $\|x_i\| = 1$.*

Proof. Recall that $\langle v_l, v_j\rangle = 0$ for any $l \ne j$ and $\|v_j\| = 1$. By definition,

$$\begin{aligned}
\|x_i\|^2 &= \left\langle \frac{\sum_{j\in I_i} \phi_j \sqrt{\lambda_j} v_j}{\sqrt{\sum_{j\in I_i} \lambda_j}}, \frac{\sum_{j\in I_i} \phi_j \sqrt{\lambda_j} v_j}{\sqrt{\sum_{j\in I_i} \lambda_j}} \right\rangle \\
&= \frac{1}{\sum_{j\in I_i} \lambda_j} \sum_{j\in I_i} \langle \phi_j \sqrt{\lambda_j} v_j, \phi_j \sqrt{\lambda_j} v_j\rangle \\
&= \frac{1}{\sum_{j\in I_i} \lambda_j} \sum_{j\in I_i} (\phi_j)^2 \lambda_j \|v_j\|^2 \\
&= 1. \qquad \blacksquare
\end{aligned}$$

LEMMA 4. *If $s \ne t$, then $\langle x_s, x_t\rangle = 0$.*

Proof. Since for any $j \in I_s$ and $l \in I_t$, $\langle v_j, v_l\rangle = 0$,

$$\begin{aligned}
\langle x_s, x_t\rangle &= \left\langle \frac{\sum_{j\in I_s} \phi_j \sqrt{\lambda_j} v_j}{\sqrt{\sum_{j\in I_s} \lambda_j}}, \frac{\sum_{j\in I_t} \phi_j \sqrt{\lambda_j} v_j}{\sqrt{\sum_{j\in I_t} \lambda_j}} \right\rangle \\
&= \frac{1}{\sqrt{\sum_{j\in I_s} \lambda_j \cdot \sum_{j\in I_t} \lambda_j}} \left\langle \sum_{j\in I_s} \phi_j \sqrt{\lambda_j} v_j, \sum_{j\in I_t} \phi_j \sqrt{\lambda_j} v_j \right\rangle \\
&= 0. \qquad \blacksquare
\end{aligned}$$

[2] An alternative way of partitioning the eigenvalues is the following: first, put the eigenvalues that are greater than or equal to 1/2 into distinct subsets. If the number of such eigenvalues, say l, is not less than k, then we are done. Otherwise, arbitrarily put the remaining eigenvalues into $k - l$ subsets such that the sum of eigenvalues in each subset is greater than or equal to 1/2. This method was suggested by an anonymous referee of a preliminary version of this paper.

Now we establish a bound on the performance of our algorithm. First, let us introduce Bernstein's theorem (see, for example, [26]), which is a form of the Chernoff bound.

LEMMA 5. *Let ϕ be a random vector whose entries are independent and either* 1 *or* -1 *with probability* $\frac{1}{2}$ *each way. Then, for any vector* e *and* $\beta > 0$,

$$\text{prob}\{\langle \phi, e\rangle^2 > \beta \|e\|^2\} < 2 \cdot \exp\left(-\frac{\beta}{2}\right).$$

Let $C_{ip} = \sum_{j \in I_i} \lambda_j \langle p, v_j\rangle^2$. Then we have

LEMMA 6. *For each* $i = 1, 2, \ldots, k$ *and each* $p \in P$, *we have*

$$\text{prob}\{\langle p, x_i\rangle^2 > 12 \log(n) \cdot C_{ip}\} < \frac{2}{n^3}.$$

Proof. Given i and p, define an $|I_i|$-dimensional vector e such that its entries are $\sqrt{\lambda_j}\langle p, v_j\rangle$, $j \in I_i$, respectively. Furthermore, we define the $|I_i|$-dimensional vector $\phi|_{I_i}$ whose entries are those of ϕ with indices in I_i. First notice that

$$\|e\|^2 = \sum_{j \in I_i} (\sqrt{\lambda_j}\langle p, v_j\rangle)^2 = \sum_{j \in I_i} \lambda_j \cdot \langle p, v_j\rangle^2 = C_{ip}.$$

On the other hand, since $\sum_{j \in I_i} \lambda_j \geq \frac{1}{2}$,

$$\begin{aligned}
\langle p, x_i\rangle^2 &= \left\langle p, \frac{\sum_{j \in I_i} \sqrt{\lambda_j} v_j \phi_j}{\sqrt{\sum_{j \in I_i} \lambda_j}} \right\rangle^2 \\
&\leq 2 \left\langle p, \sum_{j \in I_i} \sqrt{\lambda_j} v_j \phi_j \right\rangle^2 \\
&= 2 \left(\sum_{j \in I_i} \sqrt{\lambda_j} \phi_j \langle p, v_j\rangle \right)^2 \\
&= 2 \left\langle \phi|_{I_i}, e \right\rangle^2.
\end{aligned}$$

Thus

$$\text{prob}\{\langle p, x_i\rangle^2 > 12 \log(n) C_{ip}\} \leq \text{prob}\{\langle \phi|_{I_i}, e\rangle^2 > 6 \log(n) \|e\|^2\}.$$

Therefore, the conclusion of the lemma follows by using Lemma 5 and by letting $\beta = 6 \log(n)$. ∎

THEOREM 1. *We can compute in polynomial time a* $(d-k)$*-flat such that, with probability at least* $1 - \frac{2}{n}$, *the distance between any point* $p \in P$ *and* F *is at most* $\sqrt{12 \log(n)} \cdot R_k(P)$.

Proof. For given $i = 1, 2, \ldots, k$ and $p \in P$, consider the event

$$B_{ip} = \{\phi | \langle p, x_i\rangle^2 > 12 \log(n) \cdot C_{ip}\}$$

and $B = \bigcup_{i,p} B_{ip}$. The probability that the event B happens is bounded by

$$\sum_{i,p} \text{prob}\{\langle p, x_i\rangle^2 > 12 \log(n) \cdot C_{ip}\} < \frac{2kn}{n^3} \leq \frac{2}{n}.$$

If B does not happen, then for any i and p,

$$\langle p, x_i\rangle^2 \le 12\log(n)\cdot C_{ip}.$$

Therefore, for each $p \in P$,

$$\sum_{i=1}^{k}\langle p, x_i\rangle^2 \le 12\log(n)\sum_{i=1}^{k} C_{ip} \le 12\log(n)\cdot R_k(P)^2.$$

The last inequality follows from Lemma 1. This completes the proof by taking F as the subspace which is orthogonal to the vectors $x_1, x_2, \ldots, x_k$. ∎

3. The inapproximability results. We start with formal definitions of the problems that will be used in the sequence of reductions from 3SAT to computing the outer k-radius $R_k(P)$ of a set P of points. Our starting point will be the classic 3SAT problem, in which we are given a 3CNF formula and we want to know if there is an assignment to its variables that simultaneously satisfies all its clauses. The next problem we consider is the resricted quadratic programming problem as defined by Brieden [14].

DEFINITION 1 (ζ-restricted quadratic programming). *We are given nonnegative integers λ, τ, κ, and σ and nonnegative rational numbers $c_{p,q,a,b}$ for $p \in [\lambda]$, $q \in [\tau]$, $a \in [\kappa]$, and $b \in [\sigma]$. (For a nonnegative integer n, $[n]$ denotes the set $\{1, 2, \ldots, n\}$.) Our goal is to maximize*

$$f(x) = \sum_{p,q,a,b} c_{p,q,a,b}x_{p,a}y_{q,b}$$

over the polytope $P \subseteq \Re^{\lambda\kappa+\tau\sigma}$ described by

$$\sum_{a\in[\kappa]} x_{p,a} = 1 \text{ for } p \in [\lambda],$$

$$\sum_{b\in[\sigma]} y_{q,b} = 1 \text{ for } q \in [\tau],$$

$$0 \le x_{p,a} \le 1 \text{ for } p \in [\lambda], a \in [\kappa],$$

$$0 \le y_{q,b} \le 1 \text{ for } q \in [\tau], b \in [\sigma].$$

We denote instances in which $\kappa, \sigma \le \zeta$ and $\lambda, \tau \le \Delta$ by ζ-restricted $QP[\Delta]$.

DEFINITION 2 (symmetric full-dimensional norm maximization). *We are given a string (n, m, A), where n and m are natural numbers and A is a rational $m \times n$ matrix. Our goal is to maximize*

$$f(x) = ||x||_2$$

over all vectors x that belong to the polytope $P = \{x| -1 \le Ax \le 1\}$. We denote an instance in which the number of rows of A is at most m and the number of columns is at most n by $NM[m, n]$.

We first prove an inapproximability result for the case $k = 1$ and later extend it to a large range of k using a simple reduction. The crux of the proof is the following lemma.

LEMMA 7. *There is a constant $c > 1$ such that for any sufficiently large integer parameter $t \ge 1$, there is a reduction T from 3SAT formulas of size m to computing R_1 for a point set of size $n = 2^{O(t2^{3t}\log m)}$ in $d = 2^{O(t\log m)}$ dimensions such that the following hold:*

1. *If ψ is satisfiable, then $R_1(T(\psi)) \geq w$ for some w.*
2. *If ψ is unsatisfiable, then $R_1(T(\psi)) \leq w'$ for some w'.*
3. $\frac{w}{w'} \geq c^t$.

This reduction, including the computation of w and w', runs in time $2^{O(t2^{3t}\log m)}$.

Proof. The proof involves a sequence of three reductions.

From 3SAT to quadratic programming. Bellare and Rogaway [8, section 4] give a reduction from 3SAT to quadratic programming via a two-prover protocol for 3SAT. We use their reduction, but in order to get the right parameters in the hardness of approximation result for quadratic programming, we need to replace the one-round two-prover protocol that they start off with by a different one that is described in Feige [16, section 2.2]. For completeness, we now describe this two-prover protocol for 3SAT.

The two-prover protocol. Feige [16, Proposition 2.1.2] shows there exists a polynomial time reduction T from 3CNF formulas to 3CNF formulas such that each clause of $T(\psi)$ has exactly three literals (corresponding to three different variables) and each variable appears in exactly five clauses and furthermore the following hold:

1. If ψ is satisfiable, then $T(\psi)$ is satisfiable.
2. If ψ is not satisfiable, then $T(\psi)$ is at most $(1-\epsilon)$-satisfiable for some constant $0 < \epsilon < 1$. That is, any assignment satisfies at most a fraction $(1-\varepsilon)$ of all clauses in $T(\psi)$.

Without the requirement that each variable appears in exactly five clauses, such a reduction is known to be a consequence of the PCP theorem [3]. We now describe the steps taken by the verifier in the two-prover protocol.

1. Convert ψ to $T(\psi)$.

2. Choose t clauses uniformly at random (with replacement) from $T(\psi)$. Ask prover P_1 for an assignment to the variables in each clause chosen.

3. From each chosen clause, choose one of the three variables in that clause uniformly at random. We get t *distinguished* variables, possibly with repetitions. Ask the prover P_2 for an assignment to each of these t variables.

4. Accept if, for each chosen clause, it is satisfied by the assignment received from prover P_1 and the assignments made by the two provers to the distinguished variable from the clause are consistent. (Acceptance means that the verifier declares ψ to be satisfiable.) For example, suppose $t = 2$ and the verifier chose the clauses $(\neg x \vee y \vee z)$ and $(\neg y \vee z \vee w)$ and chose x and w as the respective distingushed variables. Suppose that P_1 returned the values x_1, y_1, and z_1 for the variables in the first clause and the values y_2, z_2, and w_2 for the variables in the second clause. Suppose that P_2 returned values x_3 and w_3 for the distinguished variables. Then the verifier accepts if both $(\neg x_1 \vee y_1 \vee z_1)$ and $(\neg y_2 \vee z_2 \vee w_2)$ evaluate to true, $x_1 = x_3$, and $w_2 = w_3$.

The prover P_1 is any function that on seeing ψ and the identity of the t clauses in $T(\psi)$ returns $3t$ bits that the verifier interprets as an assignment to the $3t$ variables in these clauses. Similarly, the prover P_2 is any function that on seeing ψ and the identity of the t distinguished variables returns t bits that the verifier interprets as an assignment to the distinguished variables.

If ψ is satisfiable, so is $T(\psi)$, and there exist provers (functions) that will cause the verifier to accept on every outcome of the random choices. This can be seen by picking a satisfying assignment to $T(\psi)$ and defining the two provers so that they answer according to this assignment. If ψ is unsatisfiable, what is the maximum probability, over all choices of provers (functions) P_1 and P_2, that the verifier accepts ψ? By Raz's parallel repetition theorem [28], this *error probability* is bounded above by s^t for some $s < 1$ (where the s depends on ε). We refer the reader to Feige [16,

section 2.2] for a discussion of this and to the paper by Håstad [24] for more details on the use of two-prover protocols in inapproximability results. Also, note that in this protocol the questions to the two provers are at most $O(t \log m)$ bits long, where m is the input size, since $O(\log m)$ bits suffice to identify a clause or variable in $T(\psi)$. The answers from the two-provers P_1 and P_2 are $3t$ and t bits long.

We now plug this two-prover protocol into the reduction of Bellare and Rogaway [8, section 4] from SAT to restricted quadratic programming via two-prover protocols. Their description assumes for simplicity that the question and answer lengths of the two-prover protocol are the same, but their reduction works even if these sizes are different. Using the fact that the answer length is at most $3t$, we obtain the following.[3]

LEMMA 8 (Bellare and Rogaway [8]). *There is a constant $f > 1$ such that, for any sufficiently large integer $t \geq 1$, there is a reduction T_1 that maps 3CNF formulas of size m to 2^{3t}-restricted $QP[2^{O(t \log m)}]$ such that the following hold:*

1. *If ψ is satisfiable, then $OPT(T_1(\psi)) = w_1$ for some w_1.*
2. *If ψ is unsatisfiable, then $OPT(T_1(\psi)) \leq w_2$ for some w_2.*
3. *$\frac{w_1}{w_2} \geq f^t$.*

Moreover, this reduction, including the computation of w_1 and w_2, runs in time $2^{O(t \log m)}$.

From quadratic programming to norm maximization. Brieden [14, Theorem 3.4] describes a set of interesting reductions that converts an instance of quadratic programming to an instance of the norm maximization problem. Using this reduction, we obtain the following.

LEMMA 9 (Brieden [14]). *For any $\lambda > 0$, there is a reduction T_2 from restricted quadratic programming to symmetric full-dimensional norm maximization that maps 2^{3t}-restricted $QP[2^{O(t \log m)}]$ into $NM[2^{O(t2^{3t} \log m)}, 2^{O(t \log m)}]$ with the following property: for any input L of QP to T_2,*

$$\frac{OPT(L)}{(1+\lambda)} \leq OPT(T_2(L)) \leq (1+\lambda)OPT(L).$$

Moreover, the reduction T_2 runs in time $2^{O(t2^{3t} \log m)}$.

From norm maximization to width computation. The reduction from symmetric full-dimensional norm maximization to width computation is simple [19] and is in fact used by Brieden [14]. Let $a_i \in \Re^n$ be the vector that corresponds to the ith row of matrix A which is input to the norm-maximation problem for $1 \leq i \leq m$. Thus the norm-maximization problem is

$$\gamma := \begin{array}{ll} \text{Maximize} & ||x||_2 \\ \text{Subject to} & \langle a_i, x \rangle^2 \leq 1 \text{ for } 1 \leq i \leq m. \end{array} \tag{3}$$

The reduction T_3 simply constructs a set B of points by adding, for each $1 \leq i \leq m$, the points a_i and $-a_i$ to B. Since B is a symmetric point set, $R_1(B)^2$ is given by the program

$$\begin{array}{ll} \text{Minimize} & \alpha \\ \text{Subject to} & \langle a_i, x \rangle^2 \leq \alpha \text{ for } 1 \leq i \leq m, \\ & \|x\|^2 = 1. \end{array} \tag{4}$$

[3] In Bellare's and Rogaway's reduction, the connection between the parameters of the two-prover protocol for 3SAT and the parameters of the resulting ζ-restricted $QP[\Delta]$ instance is as follows: ζ is exponential in the answer length, Δ is exponential in the question length, and the "gap" f^t in Lemma 8 is the reciprocal of the error probability of the protocol.

It is easy to verify that $\gamma = 1/R_1(B)$.

The reduction T claimed in Lemma 7 is obtained by composing the reductions T_1, T_2, and T_3. In particular, choose λ such that $(1+\lambda)^2 < f$, and let

$$c = \frac{1}{(1+\lambda)^2} f > 1.$$

It can now be checked that Lemma 7 holds with this choice of c. □

THEOREM 2.

1. *There exists a constant $\delta > 0$ such that the following holds: there is no quasi-polynomial time algorithm that approximates $R_1(P)$ within $(\log n)^\delta$ unless NP $\subseteq \tilde{P}$.*
2. *Fix any constant $b \geq 1$. Then there is no quasi-polynomial time algorithm that approximates $R_1(P)$ within $(\log d)^b$ unless NP $\subseteq \tilde{P}$.*

Proof. To prove part 1, we apply the reduction of Lemma 7 with $t = \log\log m$ to obtain an instance of computing R_1 for a set of $n = 2^{O(t2^{3t}\log m)}$ in $d = 2^{O(t\log m)}$ dimensions. Choose $\delta' < \frac{\log c}{5}$. Then

$$\frac{c^t}{(t2^{3t})^{\delta'}} \geq \frac{c^t}{(2^{4t})^{\delta'}} \geq \left(\frac{c}{2^{4\delta'}}\right)^t > (2^{\delta'})^t \geq (\log m)^{\delta'}.$$

Thus $c^t > (t2^{3t}\log m)^{\delta'}$. Since $n = 2^{O(t2^{3t}\log m)}$, we can choose $\delta < \delta'$ such that, for n large enough,

$$c^t > (\log n)^\delta.$$

To prove part 2, we apply the reduction of Lemma 7 with $t = \frac{2p\log\log m}{\log c}$ for some sufficiently large constant p. Then,

$$\frac{c^t}{t^p} \geq \frac{2^{2p\log\log m}}{t^p} \geq 2^{p\log\log m}\frac{2^{p\log\log m}}{t^p} = 2^{p\log\log m}\left(\frac{\log m}{t}\right)^p > 2^{p\log\log m}$$

since $\log m > t$ for sufficiently large m.

Thus, $c^t > t^p 2^{p\log\log m} = (t\log m)^p$. Since the dimension d is $2^{O(t\log m)}$, it follows that for every constant $b \geq 1$, we can choose p large enough such that

$$c^t > (\log d)^b.$$

Observe that the reduction runs in quasi-polynomial time for our choice of t in both cases and hence the theorem follows. □

We now give the easy reduction from width to the outer k-radius that proves the main result of this section.

THEOREM 3.

1. *There exists a constant $\delta > 0$ such that the following holds for any $0 < \varepsilon < 1$: there is no quasi-polynomial time algorithm that approximates $R_k(P)$ within $(\log n)^\delta$ for all k such that $k \leq d - d^\varepsilon$ unless NP $\subseteq \tilde{P}$.*
2. *Fix any $\varepsilon > 0$. Fix any constant $c \geq 1$. Then there is no quasi-polynomial time algorithm that approximates $R_k(P)$ within $(\log d)^c$ for all k such that $k \leq d - d^\varepsilon$ unless NP $\subseteq \tilde{P}$.*

Proof. Let P be a set of n points in $\Re^d$. We map P to a set P' of n points in $\Re^{d+k-1}$ using the function that takes a point $(x_1, \ldots, x_d) \in \Re^d$ to the point $(x_1, \ldots, x_d, 0, \ldots, 0)$. It is easily checked that $R_1(P) = R_k(P')$. Theorem 3 follows

from this reduction and some simple calculations: observe that the reduction runs in polynomial time even if we set k to be $d^{1/\varepsilon} - d + 1$. With this choice, the target dimension $d' := d + k - 1$ equals $d^{1/\varepsilon}$. Thus $k = d^{1/\varepsilon} - d + 1 \geq d' - d'^{\varepsilon}$. Theorem 3(1) now follows by applying Theorem 2(1). For part (2), we apply Theorem 2(2) with $b = 2c$. Since

$$(\log d)^{2c} \geq (\varepsilon \log d')^{2c} = (\varepsilon^2 \log d')^c (\log d')^c \geq (\log d')^c$$

for sufficiently large d', Theorem 3(2) also follows. ∎

4. Conclusions. Finding efficient rounding methods for SDP relaxation plays a key role in constructing better approximation algorithms for various hard optimization problems. All of them developed to date are randomized in nature. Therefore, the mixed deterministic and randomized rounding procedure developed in this paper may have its own independent value. We expect to see more applications of the procedure in approximating various computational geometry and space embedding problems.

Acknowledgment. We wish to thank Andreas Brieden and the anonymous referees for their valuable feedback.

REFERENCES

[1] D. Achlioptas and F. McSherry, *Fast computation of low rank matrix approximations*, in Proceedings of the ACM Symposium on the Theory of Computing, 2001.

[2] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan, *Approximating extent measures of points*, J. ACM, 51 (2004), pp. 606–635.

[3] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and hardness of approximation problems*, J. ACM, 45 (1998), pp. 501–555.

[4] S. Arora and S. Safra, *Probabilistic checking of proofs: A new characterization of NP*, J. ACM, 45 (1998), pp. 70–122.

[5] Y. Azar, A. Fiat, A. Karlin, F. McSherry, and J. Saia, *Spectral analysis of data*, in Proceedings of the ACM Symposium on the Theory of Computing, 2001.

[6] G. Barequet and S. Har-Peled, *Efficiently approximating the minimum-volume bounding box of a point set in three dimensions*, J. Algorithms, 38 (2001), pp. 91–109.

[7] M. Bădoiu, S. Har-Peled, and P. Indyk, *Approximate clustering via core-sets*, in Proceedings of the ACM Symposium on the Theory of Computing, 2002.

[8] M. Bellare and P. Rogaway, *The complexity of approximating a nonlinear program*, Math. Program. B, 69 (1995), pp. 429–441.

[9] D. Bertsimas and Y. Ye, *Semidefinite Relaxations, Multivariate Normal Distributions, and Order Statistics*, Handbook Combin. Optim. 3, D.-Z. Du and P. M. Pardalos, eds., Kluwer Academic Publishers, Norwell, MA, 1998, pp. 1–19.

[10] H. L. Bodlaender, P. Gritzmann, V. Klee, and J. Van Leeuwen, *The computational complexity of norm maximization*, Combinatorica, 10 (1990), pp. 203–225.

[11] A. Brieden, P. Gritzmann, and V. Klee, *Inapproximability of some geometric and quadratic optimization problems*, in Approximation and Complexity in Numerical Optimization: Continuous and Discrete Problems, P. M. Pardalos, ed., Kluwer Academic Publishers, Norwell, MA, 2000, pp. 96–115.

[12] A. Brieden, P. Gritzmann, R. Kannan, V. Klee, L. Lovasz, and M. Simonovits, *Deterministic and randomized polynomial-time approximation of radii*, Mathematika, 48 (2001), pp. 63–105.

[13] A. Brieden, P. Gritzmann, R. Kannan, V. Klee, L. Lovasz, and M. Simonovits, *Approximation of diameters: Randomization doesn't help*, in Proceedings of the IEEE Symposium on the Foundations of Computer Science, 1998, pp. 244–251.

[14] A. Brieden, *Geometric optimization problems likely not contained in APX*, Discrete Comput. Geom., 28 (2002), pp. 201–209.

[15] U. Faigle, W. Kern, and M. Streng, *Note on the computational complexity of j-radii of polytopes in R^n*, Math. Program., 73 (1996), pp. 1–5.

[16] U. Feige, *A threshold of* $\ln n$ *for approximating set cover*, J. ACM, 45 (1998), pp. 634–652.

[17] A. Frieze, R. Kannan, and S. Vempala, *Fast Monte-Carlo algorithms for finding low rank approximations*, J. ACM, 51 (2004), pp. 1025–1041.
[18] M. X. Goemans and D. P. Williamson, *Improved approximation algorithms for maximum cut and satisfiability problems using semi-definite programming*, J. ACM, 42 (1995), pp. 1115–1145.
[19] P. Gritzmann and V. Klee, *Inner and outer j-radii of convex bodies in finite-dimensional normed spaces*, Discrete Comput. Geom., 7 (1992), pp. 255–280.
[20] P. Gritzmann and V. Klee, *Computational complexity of inner and outer j-radii of polytopes in finite-dimensional normed spaces*, Math. Program., 59 (1993), pp. 162–213.
[21] P. Gritzmann and V. Klee, *On the complexity of some basic problems in computational convexity:* I. *Containment problems*, Discrete Math., 136 (1994), pp. 129–174.
[22] S. Har-Peled and K. Varadarajan, *Projective clustering in high dimensions using core-sets*, in Proceedings of the 18th Annual Symposium on Computational Geometry, ACM Press, 2002, pp. 312–318.
[23] S. Har-Peled and K. Varadarajan, *High-dimensional shape fitting in linear time*, Discrete Comput. Geom., 32 (2004), pp. 269–288.
[24] J. Håstad, *Some optimal inapproximability results*, J. ACM, 48 (2001), pp. 798–859.
[25] N. Megiddo, *On the complexity of some geometric problems in unbounded dimension*, J. Symbolic Comput., 10 (1990), pp. 327–334.
[26] A. Nemirovski, C. Roos, and T. Terlaky, *On maximization of quadratic forms over intersection of ellipsoids with common center*, Math. Program., 86 (1999), pp. 463–473.
[27] Yu. Nesterov, *Global quadratic optimization via conic relaxation*, in Handbook of Semidefinite Programming Theory, Algorithms, and Applications, H. Wolkowicz, R. Saigal, and L. Vandenberghe, eds., Kluwer Academic Publishers, Norwell, MA, 2000.
[28] R. Raz, *A parallel repetition theorem*, SIAM J. Comput., 27 (1998), pp. 763–803.

SIAM J. COMPUT.
Vol. 36, No. 6, pp. 1777–1806

LINEAR RECURRENCES WITH POLYNOMIAL COEFFICIENTS AND APPLICATION TO INTEGER FACTORIZATION AND CARTIER–MANIN OPERATOR*

ALIN BOSTAN†, PIERRICK GAUDRY‡, AND ÉRIC SCHOST‡

Abstract. We study the complexity of computing one or several terms (not necessarily consecutive) in a recurrence with polynomial coefficients. As applications, we improve the best currently known upper bounds for factoring integers deterministically and for computing the Cartier–Manin operator of hyperelliptic curves.

Key words. linear recurrences, factorization, Cartier–Manin operator

AMS subject classifications. 11Y16, 68Q25, 11Y05

DOI. 10.1137/S0097539704443793

1. Introduction. We investigate complexity questions for linear recurrent sequences. Our main focus is on the computation of one term, or several terms not necessarily consecutive, in a recurrence with polynomial coefficients. As applications, we improve the deterministic complexity of factoring integers and of computing the Cartier–Manin operator of hyperelliptic curves.

A well-known particular case is that of linear recurrences with constant coefficients. In this case, the Nth term can be computed with a complexity logarithmic in N, using binary powering. In the general case, there is a significant gap, as no algorithm with a complexity polynomial in $(\log N)$ is known. However, Chudnovsky and Chudnovsky showed in [11] how to compute one term in such a sequence without computing all intermediate ones. This algorithm is closely related to Strassen's algorithm [48] for integer factorization; using baby steps/giant steps (BSGS) techniques, it requires a number of operations which are roughly linear in $\sqrt{N}$ to compute the Nth term.

Precisely, let R be a commutative ring with unity and let M (resp., MM) be a function $\mathbb{N} \to \mathbb{N}$ such that polynomials of degree less than d (resp., matrices of size $n \times n$) can be multiplied in $\mathsf{M}(d)$ (resp., $\mathsf{MM}(n)$) operations $(+, -, \times)$; for $x \in \mathbb{R} - \mathbb{N}$, we write $\mathsf{M}(x) = \mathsf{M}(\lceil x \rceil)$. Next let $M(X)$ be an $n \times n$ matrix with entries in $R[X]$ of degree at most 1. Given a vector of initial conditions $U_0 \in R^n$, define the sequence (U_i) of vectors in R^n by the vector recurrence

$$U_{i+1} = M(i+1)U_i \text{ for all } i \geq 0.$$

Then, assuming that $2, \ldots, \lceil \sqrt{N} \rceil$ are units in R, Chudnovsky and Chudnovsky showed that U_N can be computed using

$$O\big(\mathsf{MM}(n)\mathsf{M}(\sqrt{N}) + n^2\mathsf{M}(\sqrt{N})\log N\big)$$

*Received by the editors May 17, 2004; accepted for publication (in revised form) September 14, 2006; published electronically March 22, 2007. A preliminary version of this paper appears in [6]; with the exception of Theorem 5, all results here are new.
http://www.siam.org/journals/sicomp/36-6/44379.html

†Domaine de Voluceau, B.P. 105, E-78153 Le Chesnay Cedex, France (alin.bostan@inria.fr).

‡Laboratoire d'Informatique LIX, École Polytechnique, 91128 Palaiseau Cedex, France (gaudry@lix.polytechnique.fr, schost@lix.polytechnique.fr).

operations. Both terms in this estimate describe basic operations on polynomial matrices of degree $\sqrt{N}$ (resp., multiplication and multipoint evaluation); using FFT-based multiplication, the cost becomes linear in $\sqrt{N}$, up to polylogarithmic factors. Our goal in this paper is to improve, generalize, and give applications of this algorithm.

- We prove that the Nth term in the sequence (U_i) above can be computed in

$$O\big(\mathsf{MM}(n)\sqrt{N} + n^2\,\mathsf{M}\big(\sqrt{N}\big)\big)$$

  operations. Compared to [11], the dependence in n stays the same; however, for fixed n, we save polylogarithmic factors in N. Chudnovsky and Chudnovsky suggested a lower bound of about $\sqrt{N}$ base ring operations for this problem; thus, our improvement gets us closer to the possible optimal bound. Furthermore, in practice, saving such polylogarithmic factors is far from negligible, since in some instances of an application, as detailed below (Cartier–Manin operator computation), N may be of order 2^{32}.
- We give a generalization to the computation of *several* selected terms, which are of indices $N_1 < \cdots < N_r = N$. When the number r of terms to be computed does not exceed $\sqrt{N}$, we show that all of them can be obtained in a time complexity which is the same as above, that is, essentially linear in $\sqrt{N}$, so we are close to the optimal.
- Along the way, we consider a question of basic polynomial arithmetic: Given the values taken by a univariate polynomial P on a large enough set of points, how fast can we compute the values of P on a shift of this set of points? An obvious solution is to use fast interpolation and evaluation techniques, but we show that this can be done faster when the evaluation points form an arithmetic progression.

In all these algorithms, we will consider polynomial matrices with coefficients of degree at most 1, which is quite frequent in applications, e.g., in the two applications presented below. However, this is not a real restriction: the case of coefficients of larger degree can be handled mutatis mutandis at the cost of a more involved presentation.

A first application is the deterministic factorization of integers. To find the prime factors of an integer N, we note that Strassen's algorithm [48] has a complexity of

$$O\big(\mathsf{M}_{\mathrm{int}}(\sqrt[4]{N}\log N)\log N\big)$$

bit operations, where we denote by $\mathsf{M}_{\mathrm{int}}$ a function such that integers of bit-size d can be multiplied in $\mathsf{M}_{\mathrm{int}}(d)$ bit operations (as above, we extend this function to take arguments in $\mathbb{R}$). Chudnovsky and Chudnovsky's algorithm generalizes Strassen's; thus, our modifications apply here as well. We prove that there exists a deterministic algorithm that outputs the complete factorization of an integer N with a bit complexity in

$$O\big(\mathsf{M}_{\mathrm{int}}(\sqrt[4]{N}\log N)\big).$$

To our knowledge, this gives the fastest deterministic integer factorization algorithm. Prior to Strassen's work, the record was held by Pollard's algorithm [35]; for any $\delta > 0$, its bit complexity is in $O(\mathsf{M}_{\mathrm{int}}(\sqrt[4]{N}\log N)N^{\delta})$. Other deterministic factorization algorithms exist [36, 30]; some have a better conjectured complexity, whose validity relies on unproved number-theoretic conjectures. The fastest probabilistic algorithm for integer factorization, with a fully established complexity bound, is due to Lenstra, Jr. and Pomerance [27], with a bit complexity polynomial in

$\exp(\sqrt{\log N \log\log N})$. The number field sieve [26] has a better conjectured complexity, expected to be polynomial in $\exp(\sqrt[3]{\log N(\log\log N)^2})$.

In accordance with these estimates, the latter algorithms are better suited for practical computations than our deterministic variant, and all recent record-sized computations rely on the number field sieve. However, as already pointed out by Pollard [35], proving unconditional, deterministic upper bounds remains an important challenge.

Our second application is point-counting in cryptography, related to the computation of the Cartier–Manin operator [10, 28] of hyperelliptic curves over finite fields.

The basic ideas already appear for elliptic curves [44, Chapter V]. Suppose we are to count the number n of solutions of the equation $y^2 = f(x)$ over $\mathbb{F}_p$, where $p > 2$ is prime and f has degree 3. Let $\chi : \mathbb{F}_p \to \mathbb{F}_p$ be the map $x \mapsto x^{(p-1)/2}$. For $x \neq 0$, $\chi(x) = 1$ when x is a square, and $\chi(x) = -1$ otherwise. Hence, n equals $\sum_{x\in\mathbb{F}_p} \chi(f(x))$ modulo p. For $i \neq 0$, $\sum_{x\in\mathbb{F}_p} x^i$ equals -1 if $p-1$ divides i, and 0 otherwise; one deduces that n modulo p is the opposite of the coefficient of x^{p-1} in $f(x)^{(p-1)/2}$.

Generalizing these ideas to hyperelliptic curves leads to the notions of the Hasse–Witt matrix and Cartier–Manin operator. Using a result of Manin [28], the Hasse–Witt matrix can be used as part of a point-counting procedure. As above, for hyperelliptic curves given by an equation $y^2 = f(x)$, the entries of this matrix are coefficients of $h = f^{(p-1)/2}$.

The coefficients of h satisfy a linear recurrence with rational function coefficients. Using our results on linear recurrences, we deduce an algorithm to compute the Hasse–Witt matrix whose complexity is essentially linear in $\sqrt{p}$. For instance, in a fixed genus, for a curve defined over the finite field $\mathbb{F}_p$, the complexity of our algorithm is

$$O\big(\mathsf{M}_{\mathrm{int}}(\sqrt{p}\log p)\big)$$

bit operations. This improves the methods of [18] and [29] which have a complexity essentially linear in p. Note that when p is small enough, other methods, such as the p-adic methods used in Kedlaya's algorithm [24], also provide very efficient point-counting procedures, but their complexity is at least linear in p; see [17].

Main algorithmic ideas. We briefly recall Strassen's factorization algorithm and Chudnovsky and Chudnovsky's generalization, and describe our modifications.

To factor an integer N, trial division with all integers smaller than $\sqrt{N}$ has a cost linear in $\sqrt{N}$. To do better, Strassen proposed to group all integers smaller than $\sqrt{N}$ into c blocks of c consecutive integers, where $c \in \mathbb{N}$ is of order $\sqrt[4]{N}$. Write $f_0 = 1\cdots c \mod N$, $f_1 = (c+1)\cdots(2c) \mod N, \ldots, f_{c-1} = (c^2-c+1)\cdots(c^2) \mod N$. If the values $f_0, \ldots, f_{c-1}$ can be computed efficiently, then finding a prime factor of N becomes easy, using the gcd's of $f_0, \ldots, f_{c-1}$ with N. Thus, the main difficulty lies in computing the values f_i, whose cost will actually dominate the whole complexity.

To perform this computation, let $R = \mathbb{Z}/N\mathbb{Z}$ and let F be the polynomial $(X+1)\cdots(X+c) \in R[X]$. The "baby steps" part of the algorithm consists of computing F: using the subproduct tree algorithm [15, Chapter 10], this is done in $O(\mathsf{M}(c)\log c)$ operations in R. Then, the "giant steps" consist of evaluating F at $0, c, \ldots, (c-1)c$, since $F(ic) = f_i$. Using fast evaluation, these values can be computed in $O(\mathsf{M}(c)\log c)$ operations. Since c has order $\sqrt[4]{N}$, the whole process has a complexity of $O(\mathsf{M}(\sqrt[4]{N})\log N)$ operations in R. This is the core of Strassen's factorization algorithm; working out the complexity estimates in a boolean complexity model yields the bounds given before.

Independently of the factorization question, one sees that multiplying the values $f_0, f_1, \ldots, f_{c-1}$ yields the product $1 \cdots c^2$ modulo N. Thus, this algorithm can be used to compute factorials: the analysis above shows that in any ring R, for any $N \geq 0$, the product $1 \cdots N$ can be computed in $O(\mathsf{M}(\sqrt{N}) \log N)$ operations in R. Note the improvement obtained over the naive iterative algorithm, whose complexity is linear in N.

Now, the sequence $U_N = N!$ is a basic example of a solution of a linear recurrence with polynomial coefficients, namely $U_N = NU_{N-1}$. Chudnovsky and Chudnovsky thus generalized the above BSGS process to compute the Nth term of an $n \times n$ matrix recurrence with polynomial coefficients of degree at most 1. The main tasks are the same as above. Computing the matrix equivalent of the polynomial F can be done using $O(\mathsf{MM}(n)\mathsf{M}(\sqrt{N}))$ operations if $2, \ldots, \lceil \sqrt{N} \rceil$ are units in R, and $O(\mathsf{MM}(n)\mathsf{M}(\sqrt{N}) \log N)$ otherwise. Then, the subsequent evaluation can be done using $O(n^2 \mathsf{M}(\sqrt{N}) \log N)$ operations. This gives the complexity estimate mentioned before.

Let us now describe our approach for the factorial (the matrix case is similar). We are not interested in the coefficients of the polynomial F, but in its values on suitable points. Now, both F and the evaluation points have special structures: F is the product of $(X+1)$, $(X+2)$, $\ldots$, $(X+c)$, whereas the evaluation points form the arithmetic progression $0, c, \ldots, (c-1)c$. This enables us to reduce the cost of evaluating F from $O(\mathsf{M}(c) \log c)$ to $O(\mathsf{M}(c))$. We use a divide-and-conquer approach; the recursive step consists of evaluating a polynomial akin to F, with degree halved, on an arithmetic progression of halved size. Putting this idea into practice involves the following operation, which is central to all our algorithms: Given the values of a polynomial P on an arithmetic progression, compute the values of P on a shift of this arithmetic progression.

In the general case of an $n \times n$ matrix recurrence, our fast solution to this problem will enable us to dispense completely with polynomial matrix multiplications, and to reduce by a logarithmic factor all costs related to multipoint evaluation. However, it will impose suitable invertibility conditions in R; we will pay special attention to such conditions, since in our two applications the base ring contains zero-divisors.

Organization of the paper. Section 2 introduces notation and previous results. Section 3 gives our algorithm for shifting a polynomial given by its values on an arithmetic progression; it is used in section 4 to evaluate some polynomial matrices, with an application in section 5 to integer factorization. In section 6, we give our modification on Chudnovsky and Chudnovsky's algorithm for computing one term in a recurrence with polynomial coefficients; a generalization to several terms is given in section 7. In section 8, we apply these results to the computation of the Cartier–Manin operator.

2. Notation and basic results. We use two computational models. Our algorithms for linear recurrent sequences apply over arbitrary rings, so their complexity is expressed in an algebraic model, counting at unit cost the base ring operations. Algorithms for integer factorization and Cartier–Manin operator computation require us to count bit operations: for this purpose, we will use the multitape Turing machine model.

Our algorithms use BSGS techniques. As usual with such algorithms, the memory requirements essentially follow the time complexities (whereas naive iterative algorithms for linear recurrences run in constant memory). We will thus give memory estimates for all our algorithms.

In what follows, $\log x$ is the base-2 logarithm of x; $\lfloor x \rfloor$ and $\lceil x \rceil$ denote, respectively, the largest integer less than or equal to x, and the smallest integer larger than or equal to x.

To make some expressions below well defined, if f is defined as a map $\mathbb{N} \to \mathbb{N}$ we may implicitly extend it to a map $\mathbb{R} \to \mathbb{N}$ by setting $f(x) = f(\lceil x \rceil)$ for $x \in \mathbb{R} - \mathbb{N}$.

All our rings will be commutative and unitary. If R is such a ring, the map $\mathbb{N} \to R$ sending n to $1 + \cdots + 1$ (n times) extends to a map $\varphi : \mathbb{Z} \to R$; we will still denote by $n \in R$ the image $\varphi(n)$.

2.1. Algebraic complexity model. The algorithms of sections 3, 4, 6, and 7 apply over arbitrary rings. To give complexity estimates, we use the straight-line program model, counting at unit cost the operations $(+, -, \times)$ in the base ring; see [8]. Hence, the time complexity of an algorithm is the size of the underlying straight-line program; we will simply speak of "ring operations." Branching and divisions are not used; thus, if we need the inverses of some elements, they will be given as inputs to the algorithm.

To assign a notion of space complexity to a straight-line program, we play a pebble game on the underlying directed acyclic graph; see [3] for a description. However, we will not use such a detailed presentation: we will simply speak of the number of ring elements that have to be stored, or of "space requirements"; such quantities correspond to the number of pebbles in the underlying pebble game.

Basic operations. Let R be a ring. The following lemma (see [32] and [33, p. 66]) shows how to trade inversions (when they are possible) for multiplications.

LEMMA 1. *Let $r_0, \ldots, r_d$ be units in R. Given $(r_0 \cdots r_d)^{-1}$, one can compute $r_0^{-1}, \ldots, r_d^{-1}$ in $O(d)$ operations and space $O(d)$.*

Proof. We first compute $R_0 = r_0$, $R_1 = r_0 r_1, \ldots, R_d = r_0 r_1 \cdots r_d$ in d multiplications. The inverse of R_d is known; by d more multiplications we deduce $S_d = R_d^{-1}$, $S_{d-1} = r_d S_d, \ldots, S_0 = r_1 S_1$, so that S_i equals $(r_0 \cdots r_i)^{-1}$. We obtain the inverse s_i of r_i by computing $s_0 = S_0$, $s_1 = R_0 S_1, \ldots, s_d = R_{d-1} S_d$ for d additional operations. ∎

In what follows, we need to compute some constants in R. For $i, d \in \mathbb{N}$, and $a \in R$, set

$$\delta(i,d) = \prod_{j=0,\ j \neq i}^{d} (i-j) \quad \text{and} \quad \Delta(a,i,d) = \prod_{j=0}^{d} (a+i-j). \tag{1}$$

Then, we have the following results.

LEMMA 2. *Suppose that $2, \ldots, d$ are units in R. Given their inverses, one can compute the inverses of $\delta(0,d), \ldots, \delta(d,d)$ in $O(d)$ operations and space $O(d)$.*

Suppose that $a-d, \ldots, a-1$ are units in R. Given their inverses, one can compute $\Delta(a,0,d), \ldots, \Delta(a,d,d)$ in $O(d)$ operations and space $O(d)$.

Proof. We use the following formulas, where i ranges from 1 to d:

$$\frac{1}{\delta(0,d)} = \frac{1}{\prod_{j=1}^{d}(-j)}, \qquad \frac{1}{\delta(i,d)} = \frac{i-d-1}{i} \frac{1}{\delta(i-1,d)},$$

$$\Delta(a,0,d) = \prod_{j=0}^{d} (a-j), \qquad \Delta(a,i,d) = \frac{a+i}{a+i-d-1} \Delta(a,i-1,d). \qquad ∎$$

Algorithms for polynomials. We denote by $\mathsf{M} : \mathbb{N} - \{0\} \to \mathbb{N}$ a function such that over any ring, the product of polynomials of degree less than d can be computed in $\mathsf{M}(d)$ ring operations. Using the algorithms of [41, 39, 9], $\mathsf{M}(d)$ can be taken in $O(d \log d \log\log d)$. Following [15, Chapter 8], we suppose that for all d and d', M satisfies the inequalities

$$\frac{\mathsf{M}(d)}{d} \leq \frac{\mathsf{M}(d')}{d'} \quad \text{if } d \leq d' \quad \text{and} \quad \mathsf{M}(dd') \leq d^2\mathsf{M}(d'), \tag{2}$$

and that the product in degree less than d can be computed in space $O(d)$. These assumptions are satisfied for naive, Karatsuba, and Schönhage–Strassen multiplications. The first inequality implies that $\mathsf{M}(d) + \mathsf{M}(d') \leq \mathsf{M}(d + d')$ and that $d \leq \mathsf{M}(d)$; the second one is used to derive the inclusion $\mathsf{M}(O(d)) \subset O(\mathsf{M}(d))$.

We use the following results for arithmetic over a ring R. The earliest references we know of are [22, 31, 47, 4], and [7] gives more recent algorithms.

Evaluation. If P is in $R[X]$, of degree at most d, and $r_0, \ldots, r_d$ are in R, then $P(r_0), \ldots, P(r_d)$ can be computed in time $O(\mathsf{M}(d) \log d)$ and space $O(d \log d)$. Using the algorithm of [16, Lemma 2.1], space can be reduced to $O(d)$, but this will not be used here.

Interpolation. For simplicity, we consider only interpolation at $0, \ldots, d$. Suppose that $2, \ldots, d$ are units in R; given their inverses, from the values $P(0), \ldots, P(d)$, one can recover the coefficients of P using $O(\mathsf{M}(d) \log d)$ operations, in space $O(d \log d)$.

See the appendix for a description of the underlying algorithmic ideas.

Matrix multiplication. We denote by $\mathsf{MM} : \mathbb{N} - \{0\} \to \mathbb{N}$ a function such that the product of $n \times n$ matrices over any ring can be computed in $\mathsf{MM}(n)$ base ring operations, in space $O(n^2)$. Thus, one can take $\mathsf{MM}(n) \in O(n^3)$ using classical multiplication, and $\mathsf{MM}(n) \in O(n^{\log 7}) \subset O(n^{2.81})$ using Strassen's algorithm [46]. We do not know whether the current record estimate [13] of $O(n^{2.38})$ satisfies our requirements. Note that $n^2 \leq \mathsf{MM}(n)$; see [8, Chapter 15].

2.2. Boolean complexity model. In sections 5 and 8, we discuss the complexity of factoring integers and of computing the Cartier–Manin operator on curves over finite fields. For these applications, the proper complexity measure is bit complexity. For this purpose, our model will be the multitape Turing machine; see, for instance, [40]. We will speak of *bit operations* to estimate time complexities in this model. Storage requirements will be expressed in bits as well, taking into account input, output, and intermediate data size.

Boolean algorithms will be given through high-level descriptions, and we shall not give the details of their multitape Turing implementations. We just mention the following relevant fact: for each algorithm, there is a corresponding multitape Turing machine. Using previously designed algorithms as subroutines is then possible; each of the corresponding machines is attributed a special band that plays the role of a stack to handle subroutine calls. We refer to [40] for examples of detailed descriptions along these lines.

Integer operations. Integers are represented in base 2. The function $\mathsf{M}_{\text{int}} : \mathbb{N} \to \mathbb{N}$ is such that the product of two integers of bit-size d can be computed within $\mathsf{M}_{\text{int}}(d)$ bit operations. Hence, multiplying integers bounded by N takes at most $\mathsf{M}_{\text{int}}(\log N)$ bit operations.

We suppose that M_{int} satisfies inequalities (2), and that product in bit-size d can be done in space $O(d)$. Using the algorithm of [41], $\mathsf{M}_{\text{int}}(d)$ can be taken in

$O(d \log d \log\log d)$. Euclidean division in bit-size d can be done in time $O(\mathsf{M}_{\mathrm{int}}(d))$ and space $O(d)$; see [12]. The extended gcd of two bit-size d integers can be computed in time $O(\mathsf{M}_{\mathrm{int}}(d)\log d)$ and space $O(d)$; see [25, 38].

Effective rings. We next introduce effective rings as a way to obtain results of a general nature in the Turing model.

Let R be a finite ring, let ℓ be in $\mathbb{N}$, and consider an injection $\sigma : R \hookrightarrow \{0,1\}^\ell$. We use σ to represent the elements of R. Polynomials in $R[X]$ are represented by the sequence of the σ-values of their coefficients. Matrices over R are represented in *row-major ordering*: an $m \times n$ matrix $A = (a_{i,j})$ is represented as $\sigma(a_{1,1}), \ldots, \sigma(a_{1,n}), \ldots, \sigma(a_{m,1}), \ldots, \sigma(a_{m,n})$.

An *effective ring* is the data of such R, ℓ, and σ, together with constants $\mathsf{m}_R, \mathsf{s}_R \in \mathbb{N}$, and maps $\mathsf{M}_R, \mathsf{S}_R$ and $\mathsf{MM}_R, \mathsf{SM}_R : \mathbb{N} - \{0\} \to \mathbb{N}$ meeting the following criteria. First, through the σ representation, we ask that

- the sum and product of elements in R can be computed in time $\mathsf{m}_R \geq \ell$ and space $\mathsf{s}_R \geq \ell$;
- the product of polynomials of degree less than d in $R[X]$ can be computed in time $\mathsf{M}_R(d)$ and space $\mathsf{S}_R(d)$;
- the product of size n matrices over R can be computed in time $\mathsf{MM}_R(n)$ and space $\mathsf{SM}_R(n)$.

We ask that for all d and d', M_R satisfies the inequalities (2),

$$\frac{\mathsf{M}_R(d)}{d} \leq \frac{\mathsf{M}_R(d')}{d'} \quad \text{if } d \leq d' \quad \text{and} \quad \mathsf{M}_R(dd') \leq d^2 \mathsf{M}_R(d'),$$

as well as $d\mathsf{m}_R \leq \mathsf{M}_R(d)$. We also ask that, for all d and d', S_R satisfies

$$\mathsf{s}_R \leq \mathsf{S}_R(d) \quad \text{and} \quad \mathsf{S}_R(dd') \leq d^2 \mathsf{S}_R(d').$$

Finally, as to matrix multiplication, we require that for all n, MM_R and SM_R satisfy

$$n^2 \mathsf{m}_R \leq \mathsf{MM}_R(n) \quad \text{and} \quad \mathsf{s}_R \leq \mathsf{SM}_R(n).$$

In what follows, our results will be first given in the algebraic model, and then on an effective ring, with a bit complexity estimate; note that for both algebraic and Turing models, all constants hidden in the $O(\ )$ estimates will be independent of the base ring.

Effective rings will enable us to state bit complexity results similar to algebraic complexity ones. We have, however, no general transfer theorem from algebraic to bit complexity. First, nonarithmetic operations (loop handling, stack managing for recursive calls) are not taken into account in the former model. In most cases however, the corresponding cost is easily seen to be negligible, so we will not spend time discussing this. A more important difference is that the algebraic model does not count time to access data, that is, the number of tape movements done in the Turing model. This point will be checked for the algorithms we will discuss on Turing machines.

For concrete applications, the following lemma, proved in the appendix, gives the basic examples of effective rings. The results for matrix multiplication are not the sharpest possible, since this would take us too far afield.

LEMMA 3. *Let N be in $\mathbb{N}$, let $R_0 = \mathbb{Z}/N\mathbb{Z}$, and let P be monic of degree m in $R_0[T]$. Then $R = R_0[T]/P$ can be made an effective ring, with*

- $\ell = m\lceil \log N \rceil$,
- $\mathsf{m}_R \in O(\mathsf{M}_{\mathrm{int}}(m \log(mN)))$ *and* $\mathsf{s}_R \in O(m \log(mN))$,

- $\mathsf{M}_R(d) \in O(\mathsf{M}_{\mathrm{int}}(dm\log(dmN)))$ *and* $\mathsf{S}_R(d) \in O(dm\log(dmN))$,
- $\mathsf{MM}_R(n) \in O(n^{\log 7}\mathsf{m}_R)$ *and* $\mathsf{SM}_R(n) \in O(n^2\ell + \mathsf{s}_R)$.

Finally, the results given before in the algebraic model have the following counterpart in the Turing model, using again the notation $\delta(i,d)$ and $\Delta(a,i,d)$ introduced in (1). The proofs are given in the appendix.

LEMMA 4. *Let R be an effective ring. Then the following hold:*

1. *Suppose that $r_0,\dots,r_d$ are units in R. Given $r_0,\dots,r_d,(r_0\cdots r_d)^{-1}$, one can compute $r_0^{-1},\dots,r_d^{-1}$ in time $O(d\mathsf{m}_R)$ and space $O(d\ell+\mathsf{s}_R)$.*
2. *Suppose that $2,\dots,d$ are units in R. Given their inverses, one can compute the inverses of $\delta(0,d),\dots,\delta(d,d)$ in time $O(d\mathsf{m}_R)$ and space $O(d\ell+\mathsf{s}_R)$.*
3. *Suppose that $a-d,\dots,a-1$ are units in R. Given their inverses, one can compute $\Delta(a,0,d),\dots,\Delta(a,d,d)$ in time $O(d\mathsf{m}_R)$ and space $O(d\ell+\mathsf{s}_R)$.*
4. *Let $P=\sum_{i=0}^d p_iX^i$ be in $R[X]$. Given $p_0,\dots,p_d$ and elements $r_0,\dots,r_d$ in R, $P(r_0),\dots,P(r_d)$ can be computed in time $O(\mathsf{M}_R(d)\log d)$ and space $O(\ell d\log d+\mathsf{S}_R(d))$.*
5. *Suppose that $2,\dots,d$ are units in R. If $P\in R[X]$ has degree at most d, then given $P(0),\dots,P(d)$ and the inverses of $2,\dots,d$, one can compute the coefficients of P in time $O(\mathsf{M}_R(d)\log d)$ and space $O(\ell d\log d+\mathsf{S}_R(d))$.*

3. Shifting evaluation values. We now address a special case of the question of *shifting evaluation values* of a polynomial. Let R be a ring, let P be of degree d in $R[X]$, and let a and $r_0,\dots,r_d$ be in R. Given $P(r_0),\dots,P(r_d)$, how fast can we compute $P(r_0+a),\dots,P(r_d+a)$? We stress the fact that the coefficients of P are *not* part of the input.

Suppose that all differences r_i-r_j, $i\neq j$ are units in R. Then using fast interpolation and evaluation, the problem can be solved using $O(\mathsf{M}(d)\log d)$ operations in R. We propose an improved solution, in the special case when $r_0,\dots,r_d$ form an arithmetic progression; its complexity is in $O(\mathsf{M}(d))$, so we gain a logarithmic factor.

Our algorithm imposes invertibility conditions on the sample points slightly more general than those above. Given α,β in R and d in $\mathbb{N}$, we define the following property: $\mathsf{h}(\alpha,\beta,d)$**:** β, $2,\dots,d$ and $\alpha-d\beta$, $\alpha-(d-1)\beta,\dots,\alpha+(d-1)\beta$, $\alpha+d\beta$ are units. We then define $\mathsf{d}(\alpha,\beta,d)=\beta\cdot 2\cdots d\cdot(\alpha-d\beta)\cdots(\alpha+d\beta)\in R$. Assumption $\mathsf{h}(\alpha,\beta,d)$ holds if and only if $\mathsf{d}(\alpha,\beta,d)$ is a unit.

THEOREM 5. *Let α,β be in R and d be in $\mathbb{N}$ such that $\mathsf{h}(\alpha,\beta,d)$ holds, and suppose that the inverse of $\mathsf{d}(\alpha,\beta,d)$ is known. Let F be in $R[X]$ of degree at most d and $r\in R$. Given*

$$F(r),F(r+\beta),\dots,F(r+d\beta),$$

one can compute

$$F(r+\alpha),F(r+\alpha+\beta),\dots,F(r+\alpha+d\beta),$$

in time $2\mathsf{M}(d)+O(d)$ and space $O(d)$, in the algebraic model. If R is effective, then the bit complexity is $2\mathsf{M}_R(d)+O(d\mathsf{m}_R)$ and the space complexity is $O(\mathsf{S}_R(d))$ bits.

Proof. Our algorithm reduces to the multiplication of two suitable polynomials of degrees at most d and $2d$; $O(d)$ additional operations come from pre- and post-processing operations. All operations below on integer values take place in R.

First, we perform a change of variables. Define $P(X)=F(\beta X+r)$; then our assumption is that the values $P(0),P(1),\dots,P(d)$ are known. Let us write $a=\alpha/\beta$;

our objective is then to determine the values $P(a), \dots, P(a+d)$. To this effect, assumption $\mathsf{h}(\alpha, \beta, d)$ enables us to write the Lagrange interpolation formula:

$$P = \sum_{i=0}^{d} P(i) \frac{\prod_{j=0, j\neq i}^{d} (X-j)}{\prod_{j=0, j\neq i}^{d} (i-j)} = \sum_{i=0}^{d} \widetilde{P}_i \prod_{j=0, j\neq i}^{d} (X-j),$$

with $\widetilde{P}_i = P(i)/\delta(i,d)$, where $\delta(i,d)$ is defined in (1). For k in $0, \dots, d$, let us evaluate P at $a+k$:

$$P(a+k) = \sum_{i=0}^{d} \widetilde{P}_i \prod_{j=0, j\neq i}^{d} (a+k-j).$$

Assumption $\mathsf{h}(\alpha, \beta, d)$ implies that $a-d, \dots, a+d$ are units. We can thus complete each product by the missing factor $a+k-i$:

$$P(a+k) = \sum_{i=0}^{d} \widetilde{P}_i \frac{\prod_{j=0}^{d} (a+k-j)}{a+k-i} = \left(\prod_{j=0}^{d} (a+k-j) \right) \cdot \left(\sum_{i=0}^{d} \widetilde{P}_i \frac{1}{a+k-i} \right). \tag{3}$$

We now use the sequence $\Delta(a,k,d)$ introduced in (1) and define $Q_k = P(a+k)/\Delta(a,k,d)$. Using these values, (3) reads

$$Q_k = \sum_{i=0}^{d} \widetilde{P}_i \frac{1}{a+k-i}. \tag{4}$$

Let $\widetilde{P}$ and S be the polynomials

$$\widetilde{P} = \sum_{i=0}^{d} \widetilde{P}_i X^i, \quad S = \sum_{i=0}^{2d} \frac{1}{a+i-d} X^i;$$

then by (4), for $k = 0, \dots, d$, Q_k is the coefficient of degree $k+d$ in the product $\widetilde{P}S$. From the knowledge of Q_k, we easily deduce $P(a+k)$.

Let us analyze the complexity of this algorithm, first in the algebraic model. Using Lemma 1, from the inverse of $\mathsf{d}(\alpha, \beta, d)$, we obtain those of β, $2, \dots, d$ and $\alpha - d\beta, \dots, \alpha + d\beta$ in $O(d)$ operations. Using the equality $(a+id)^{-1} = \beta(\alpha + id\beta)^{-1}$, we obtain the inverses of $a-d, \dots, a+d$ in $O(d)$ further operations. Lemma 2 then gives all $\Delta(a,i,d)$ and the inverses of all $\delta(i,d)$ for $O(d)$ operations as well. The sequence $\widetilde{P}_i$ is deduced for $O(d)$ operations.

The coefficients Q_i are then obtained by a polynomial multiplication in degrees d and $2d$; this can be reduced to two polynomial multiplications in degrees less than d, and $O(d)$ additional operations, for a complexity of $2\mathsf{M}(d) + O(d)$. Given $Q_0, \dots, Q_d$, we deduce $P(a), \dots, P(a+d)$ by multiplications with the coefficients $\Delta(a,i,d)$; this requires $O(d)$ ring operations. This concludes the algebraic complexity estimates, since space requirements are easily seen to be in $O(d)$.

When R is effective, we have to implement this algorithm on a multitape Turing machine. For this simple algorithm, there is no difficulty; we give details to show the manipulations that need to be made, making no effort to minimize the number of tapes. For the next algorithms, we will be more sketchy and concentrate on difficult points.

Initially, $P(0), \ldots, P(d)$ and the inverse of $\mathsf{d}(\alpha, \beta, d)$ are contiguous blocks of ℓ bits on the input tape. First, we produce on an auxiliary tape T_1 all elements whose inverses will be used, *in a suitable order*, namely β, $2, \ldots, d, \alpha - d\beta, \ldots, \alpha + d\beta$. Then the inverse of $\mathsf{d}(\alpha, \beta, d)$ is appended to these elements; using Lemma 4, we obtain β^{-1}, $2^{-1}, \ldots, d^{-1}$ and $(\alpha - d\beta)^{-1}, \ldots, (\alpha + d\beta)^{-1}$ on a tape T_2. As before, we deduce $(a-d)^{-1}, \ldots, (a+d)^{-1}$; this is done is a single sweep of T_2, and the results are stored on a tape T_3. Then, using Lemma 4, all $\delta(i, d)^{-1}$ are computed and stored on a tape T_4, and all $\Delta(a, i, d)$ on a tape T_5. The whole cost up to now is $O(d\mathsf{m}_R)$, the cost of the tape movements being $O(d\ell)$. The space complexity is in $O(d\ell + \mathsf{s}_R)$.

The coefficients of S are copied from T_3 to a tape T_6, and those of $\widetilde{P}$ are computed and stored on a tape T_7; the cost is $O(d\mathsf{m}_R)$, since the data are well organized on tapes. We then compute the product of S and $\widetilde{P}$. The result is the list of coefficients Q_k, stored on a tape T_8 after a time $2\mathsf{M}_R(d) + O(d\mathsf{m}_R)$. Finally the target values $P(a+k)$ are computed at an additional cost of $O(d\mathsf{m}_R)$, since again everything is well organized. This concludes the time analysis. The space complexity is easily seen to fit the required bound. □

Remark 1. In [20], the operation called *middle product* is defined: Given a ring R, and A, B in $R[X]$ of respective degrees at most d and $2d$, write $AB = C_0 + C_1 X^{d+1} + C_2 X^{2d+2}$, with all C_i of degree at most d; then the middle product of A and B is the polynomial C_1. This is precisely what is needed in the algorithm above.

Up to considering the reciprocal polynomial of A, the middle product by A can be seen as the transpose of the map of multiplication by A. General program transformation techniques then show that it can be computed in time $\mathsf{M}(d) + O(d)$ (but with a possible loss in space complexity): this is the *transposition principle* for linear algorithms, which is an analogue of results initiated by Tellegen [49] and Bordewijk [2] in circuit theory. Thus, the time complexity of the algorithm above can be reduced to $\mathsf{M}(d) + O(d)$ ring operations, but possibly with an increased space complexity. We refer to [23, Problem 6] for a longer discussion and [8, Theorem 13.20] for a proof; see also [20] for the independent discovery of the middle product, and [7] for additional applications.

Remark 2. Using the notation of the proof above, we mention an alternative $O(\mathsf{M}(d))$ algorithm which does *not* require any invertibility assumption, in the special case when $a = d + 1$. The key fact is that for any polynomial P of degree d, the sequence $P(0), P(1), \ldots$ is linearly recurrent, of characteristic polynomial $Q(X) = (1 - X)^{d+1}$. Thus, if the first terms $P(0), \ldots, P(d)$ are known, the next $d + 1$ terms $P(d + 1), \ldots, P(2d + 1)$ can be recovered in $O(\mathsf{M}(d))$ using the algorithm in [43, Theorem 3.1].

4. Algorithms for polynomial matrix evaluation. In Strassen's algorithm sketched in the introduction, an important part of the effort lies in evaluating polynomials on points that form an arithmetic progression. A generalization of this question appears in Chudnovsky and Chudnovsky's algorithm for matrix recurrences, where one has to evaluate a polynomial matrix at points in an arithmetic progression. We now present such an evaluation algorithm, in the special case when the polynomial matrix has the form

$$M_k(X) = M(X + \alpha k) \cdots M(X + \alpha),$$

where $M(X)$ is a given $n \times n$ polynomial matrix with entries of degree at most 1: this is enough to handle both Strassen's and Chudnovsky and Chudnovsky's algorithms. Using the result of the previous section, we propose a divide-and-conquer approach,

which, for fixed n, saves a logarithmic factor in k compared to classical multipoint evaluation techniques.

Let R be a ring. We will need several invertibility assumptions in R, in order to apply Theorem 5 along all recursive steps of the algorithm; we discuss this first. With a positive integer k, we associate the sequence $k_0, \dots, k_{\lfloor \log k \rfloor}$ defined by $k_0 = k$ and $k_{i+1} = \lfloor k_i/2 \rfloor$, so that $k_{\lfloor \log k \rfloor} = 1$.

Then, given α, β in R and k in $\mathbb{N}$, we say that assumption $\mathsf{H}(\alpha, \beta, k)$ holds if assumptions $\mathsf{h}(\beta(k_i+1), \beta, k_i)$ and $\mathsf{h}(\alpha k_i, \beta, k_i)$ of the previous section hold for $i = 1, \dots, \lfloor \log k \rfloor$: this is what we need for the algorithm below. We write $\mathsf{D}(\alpha, \beta, k)$ for the product

$$\prod_{i=1}^{\lfloor \log k \rfloor} \mathsf{d}(\beta(k_i+1), \beta, k_i)\, \mathsf{d}(\alpha k_i, \beta, k_i);$$

note that $\mathsf{H}(\alpha, \beta, k)$ holds if and only if $\mathsf{D}(\alpha, \beta, k)$ is a unit in R. We mention a few basic results related to this definition; the straightforward proofs are left to the reader.

LEMMA 6. *Given α, β, and k, $\mathsf{D}(\alpha, \beta, k)$ can be computed in time and space $O(k)$, in the algebraic model. If R is effective, this can be done in time $O(k\mathsf{m}_R)$ and space $O(k\ell + \mathsf{s}_R)$.*

Condition $\mathsf{H}(\alpha, \beta, k)$ asserts that $O(k)$ elements are units in R. It is easy, but cumbersome, to give the list of these elements. It will be enough to note the following particular cases.

LEMMA 7.

- $\mathsf{H}(k, 1, k)$ *holds if and only if* $2, \dots, 2k_i+1$ *and* $kk_i - k_i, \dots, kk_i + k_i$ *are units in R, for* $i = 1, \dots, \lfloor \log k \rfloor$.
- $\mathsf{H}(1, 2^s, 2^s)$ *holds if and only if* $2, \dots, 2^s + 1$ *are units in R.*

We can now state the main result of this section.

THEOREM 8. *Suppose that $\mathsf{H}(\alpha, \beta, k)$ holds and that the inverse of $\mathsf{D}(\alpha, \beta, k)$ is known. Then the scalar matrices $M_k(0), M_k(\beta), \dots, M_k(k\beta)$ can be computed in*

$$O(\mathsf{MM}(n)k + n^2\mathsf{M}(k))$$

ring operations, in space $O(n^2k)$. If R is effective, then the bit complexity is

$$O\big(\mathsf{MM}_R(n)k + n^2\mathsf{M}_R(k) + n^2\ell k \min(\log k, \log n)\big),$$

and the space complexity is $O(n^2k\ell + \mathsf{S}_R(k) + \mathsf{SM}_R(n))$ bits.

Proof. We first deal with inverses. Let $k_0, \dots, k_{\lfloor \log k \rfloor}$ be defined as above. In the following we need the inverses of all

$$\mathsf{d}(\alpha k_i, \beta, k_i) \quad \text{and} \quad \mathsf{d}(\beta(k_i+1), \beta, k_i) \quad \text{for } i = 1, \dots, \lfloor \log k \rfloor.$$

For any i, both $\mathsf{d}(\alpha k_i, \beta, k_i)$ and $\mathsf{d}(\beta(k_i+1), \beta, k_i)$ can be computed in $O(k_i)$ ring operations; hence, all of them can be computed in $O(k)$ operations. Using Lemma 1, their inverses can be deduced from that of $\mathsf{D}(\alpha, \beta, k)$ for $O(\log k)$ products.

We will then give an estimate on the complexity of computing the values of $M_{k_i}(X)$ on $0, \beta, \dots, k_i\beta$, for decreasing values of i. The case $i = \lfloor \log k \rfloor$ is obvious, since then $k_i = 1$ and $M_{k_i}(X) = M(X + \alpha)$, which can be evaluated at 0 and β in $O(n^2)$ operations.

Then, for some $i = \lfloor \log k \rfloor, \dots, 1$, suppose that the values of $M_{k_i}(X)$ are known on $0, \beta, \dots, k_i\beta$. We now show how to deduce the values of $M_{k_{i-1}}(X)$ on $0, \beta, \dots, k_{i-1}\beta$.

To this effect, we will use Theorem 5, using the fact that all entries of $M_{k_i}(X)$ have degree at most k_i. To keep control on the $O(\ )$ constants, we let C be a constant such that the complexity estimate in Theorem 5 is upper-bounded by $2\mathsf{M}(d) + \mathsf{C}d$.

- Applying Theorem 5 to each entry of $M_{k_i}(X)$ to perform a shift by $(k_i+1)\beta$, we see that the values of $M_{k_i}(X)$ on $(k_i+1)\beta, \dots, (2k_i+1)\beta$ can be computed for $n^2(2\mathsf{M}(k_i)+\mathsf{C}k_i)$ ring operations, since $\mathsf{d}(\beta(k_i+1), \beta, k_i)^{-1}$ is known. We then have at our disposal the values of $M_{k_i}(X)$ at $0, \beta, \dots, (2k_i+1)\beta$.
- Applying Theorem 5 to each entry of $M_{k_i}(X)$ to perform shifts by $k_i\alpha$ and then by $(k_i+1)\beta$, we see that the values of $M_{k_i}(X+k_i\alpha)$ on $0, \beta, \dots, (2k_i+1)\beta$ can be computed for $2n^2(2\mathsf{M}(k_i) + \mathsf{C}k_i)$ ring operations. For the first shift we need $\mathsf{d}(\alpha k_i, \beta, k_i)^{-1}$ and for the second we need $\mathsf{d}(\beta(k_i+1), \beta, k_i)^{-1}$; they have both been precomputed in the preamble.

From these values, it is easy to deduce the values of $M_{k_{i-1}}(X)$ at $0, \beta, \dots, k_{i-1}\beta$. We distinguish two cases, according to the parity of k_{i-1}.

- Suppose first that k_{i-1} is even, so that $k_{i-1} = 2k_i$. Using the equality
$$M_{k_{i-1}}(X) = M_{k_i}(X + \alpha k_i) \cdot M_{k_i}(X),$$
we obtain the values of $M_{k_{i-1}}(X)$ at $0, \beta, \dots, k_{i-1}\beta$ by multiplying the known values of $M_{k_i}(X + \alpha k_i)$ and $M_{k_i}(X)$ at $0, \beta, \dots, k_{i-1}\beta$. This takes $(k_{i-1} + 1)\mathsf{MM}(n)$ operations.
- Suppose now that k_{i-1} is odd, so that $k_{i-1} = 2k_i + 1$. Using the equality
$$M_{k_{i-1}}(X) = M(X + \alpha k_{i-1}) \cdot M_{k_i}(X + \alpha k_i) \cdot M_{k_i}(X),$$
we obtain the values of $M_{k_{i-1}}(X)$ at $0, \beta, \dots, k_{i-1}\beta$ as follows. We first multiply the values of $M_{k_i}(X + \alpha k_i)$ and $M_{k_i}(X)$ at $0, \beta, \dots, k_{i-1}\beta$, for $(k_{i-1}+1)\mathsf{MM}(n)$ operations. Then we evaluate $M(X+\alpha k_{i-1})$ at these points for $2n^2(k_{i-1} + 1)$ operations, from which we deduce the requested values of $M_{2k_i+1}(X)$ for $(k_{i-1} + 1)\mathsf{MM}(n)$ operations.

Let us denote by $T(k_{i-1})$ the cost of these operations. From the analysis above, we deduce the inequality

$$T(k_{i-1}) \le T(k_i) + 2(k_{i-1} + 1)\mathsf{MM}(n) + 6n^2\mathsf{M}(k_i) + n^2\mathsf{C}'k_i$$

for a constant C' that can be taken to be $\mathsf{C}' = 3\mathsf{C} + 4$. Using the definition of the sequence k_i and our assumptions on the function M, we deduce the estimate $T(k) = T(k_0) \in O(\mathsf{MM}(n)k + n^2\mathsf{M}(k))$. The space complexity estimate is easily dealt with, since all computations at step i can be done in space $O(n^2k_i)$.

We now come to the last statement, where R is effective. In the Turing context, we have to pay attention to the way matrices and vectors are represented. The preamble, where inverses are precomputed, poses no problem; it suffices to organize the elements to invert in the correct order. However, at the heart of the procedure, we have to switch between two representations of matrices of vectors over R. This is free in an algebraic model but has to be justified to have negligible cost in a Turing model.

At the input of the inner loop on i, we have on a tape the values of $M_{k_i}(X)$ at $0, \beta, \dots, k_i\beta$. They are stored as a sequence of matrices over R, each in row-major representation. In order to apply Theorem 5, we need its input to be contiguous on a tape. Hence, we must first reorganize the data, switching to a representation as a matrix of vectors, with vectors of size $k_i + 1$: Corollary 19 in the appendix shows how to do this in cost $O(\ell n^2 k_i \min(\log k_i, \log n))$. After the applications of Theorem 5, we

have at our disposal the values of $M_{k_i}(X)$ and of $M_{k_i}(X+k_i\alpha)$ at $0, \beta, \dots, (2k_i+1)\beta$, organized as matrices of vectors. We switch back to their representation as a sequence of matrices over R to perform the matrix multiplications. This is the converse problem as before, and it admits the same $O(\ell n^2 k_i \min(\log k_i, \log n))$ solution. The matrix products can then be done at the expected cost, since the input data is contiguous, and finally we are ready to enter again the loop with the next value of i.

Putting together the costs of these operations, the additional cost compared to the algebraic model is $O(\ell n^2 k \min(\log k, \log n))$. The memory requirements consist of $O(n^2 k\ell)$ for storing all intermediate matrices and polynomials, plus a term in $O(\mathsf{S}_R(k))$ coming from Theorem 5, and the term $\mathsf{SM}_R(n)$ for matrix multiplication. □

The case $\alpha = \beta = 1$ is not covered in the last theorem. However, this case is easier to handle (note also that there are no invertibility conditions).

PROPOSITION 9. *Suppose that $\alpha = 1$, that is, $M_k(X) = M(X+k)\cdots M(X+1)$. Then the scalar matrices $M_k(0), M_k(1), \dots, M_k(k)$ can be computed using $O(\mathsf{MM}(n)k)$ ring operations, in space $O(n^2k)$. If R is effective, the bit complexity is $O(\mathsf{MM}_R(n)k)$ and the space requirement is $O(n^2k\ell + \mathsf{SM}_R(n))$ bits.*

Proof. We first evaluate all matrices $M(1), \dots, M(2k)$; this requires $O(n^2k)$ operations and takes space $O(n^2k)$. The conclusion is now similar to the proof of Lemma 1. Denoting by I the $n \times n$ identity matrix, we first compute the products

$$R_k = I, \quad R_{k-1} = R_k M(k), \quad R_{k-2} = R_{k-1}M(k-1), \dots, R_0 = R_1 M(1)$$

and

$$L_0 = I, \quad L_1 = M(k+1)L_0, \quad L_2 = M(k+2)L_1, \dots, L_k = M(2k)L_{k-1}.$$

This takes $O(\mathsf{MM}(n)k)$ ring operations, and $O(n^2k)$ space. We conclude by computing the matrices $M_k(i) = L_i R_i$ for $0 \le i \le k$. This also takes $O(\mathsf{MM}(n)k)$ ring operations and $O(n^2k)$ space. The estimates in the Turing model come similarly. □

In section 7, we have to deal with a similar evaluation problem, but with arbitrary sample points. The following corollary of Proposition 9 will then be useful; compared to the particular case of Theorem 8, we lose a logarithmic factor in the evaluation process.

COROLLARY 10. *Let notation be as in Proposition 9 and assume that $2, \dots, k$ are units in R, and that their inverses are known. Then for any set of $k+1$ elements β_i of R, the matrices $M_k(\beta_0), M_k(\beta_1), \dots, M_k(\beta_k)$ can be computed in*

$$O(\mathsf{MM}(n)k + n^2\mathsf{M}(k)\log k)$$

operations, in space $O(kn^2 + k\log k)$. If R is effective, the bit complexity is

$$O(\mathsf{MM}_R(n)k + n^2\mathsf{M}_R(k)\log k),$$

and the space requirement is $O(k\ell n^2 + \mathsf{SM}_R(n) + \ell k \log k + \mathsf{S}_R(k))$ bits.

Proof. We start by evaluating $M_k(X)$ on $0, \dots, k$; by Proposition 9, this takes $O(\mathsf{MM}(n)k)$ operations and space $O(n^2k)$. Using the inverses of $2, \dots, k$, it is then possible to interpolate all entries of $M_k(X)$ in $O(n^2\mathsf{M}(k)\log k)$ operations, in space $O(k\log k)$. Then, we can evaluate all entries of this matrix at the points $\beta_0, \beta_1, \dots, \beta_k$, with the same cost.

In the Turing model, we face the same problem as in Theorem 8. The output of Proposition 9 is given as a sequence of scalar matrices, so we switch to a matrix of

vectors to apply interpolation and evaluation, and convert it back to a sequence of matrices. Proposition 9 gives the cost for evaluation at $0, \dots, k$; Lemma 4 gives that for interpolation at these points, and for evaluation at $\beta_0, \dots, \beta_k$; Corollary 19 gives the cost for changing the data's organization. The conclusion follows. ∎

5. Application to integer factorization. In this section, we apply the algorithm of Theorem 8 to reduce by a logarithmic factor the best upper bound for deterministic integer factorization, due to Strassen [48]. Strassen's result is that a positive integer N can be completely factored using at most $O\big(\mathsf{M}_{\mathrm{int}}(\sqrt[4]{N}\log N)\log N\big)$ bit operations. The following theorem improves this result.

THEOREM 11. *There exists a deterministic algorithm that outputs the complete factorization of any positive integer N using at most $O(\mathsf{M}_{\mathrm{int}}(\sqrt[4]{N}\log N))$ bit operations. The space complexity is $O(\sqrt[4]{N}\log N)$ bits.*

Our proof closely follows that of Strassen, in the presentation of [15], the main ingredient being now Theorem 8; some additional complications arise due to the nontrivial invertibility conditions required by that theorem. First, applying Lemma 3 (with $m = 1$) gives the data describing $\mathbb{Z}/N\mathbb{Z}$ as an effective ring:

- $\ell = \lceil \log N \rceil$,
- $\mathsf{m}_R \in O(\mathsf{M}_{\mathrm{int}}(\log N))$ and $\mathsf{s}_R \in O(\log N)$,
- $\mathsf{M}_R(d) \in O(\mathsf{M}_{\mathrm{int}}(d\log(dN)))$ and $\mathsf{S}_R(d) \in O(d\log(dN))$.

The data for matrix multiplication is not required here, since all matrices have size 1.

LEMMA 12. *Let $f_0, \dots, f_{k-1}$ be in $\mathbb{Z}/N\mathbb{Z}$. Then one can decide whether all f_i are invertible modulo N and, if not, find a noninvertible f_i in time*

$$O\big(k\,\mathsf{M}_{\mathrm{int}}(\log N) + \log k\,\mathsf{M}_{\mathrm{int}}(\log N)\log\log N\big)$$

and space $O(k\log N)$ bits.

Proof. For $i \le k-1$, we will denote by F_i the canonical preimage of f_i in $[0, \dots, N-1]$. Hence, our goal is to find one F_i such that $\gcd(F_i, N) > 1$.

We first form the "subproduct tree" associated with the values F_i. By completing with enough 1's, we can assume that k is a power of 2, i.e., that $k = 2^m$. First, we define $F_{i,m} = F_i$ for $i = 0, \dots, k-1$; then iteratively we let $F_{i,j-1} = F_{2i,j}F_{2i+1,j} \bmod N$, for $j = m, \dots, 1$ and $i = 0, \dots, 2^{j-1}-1$. These numbers are organized in a tree similar to the one described in the appendix for evaluation and interpolation; see also [15, Chapter 10]. The number of multiplications to perform in $\mathbb{Z}/N\mathbb{Z}$ is at most k; hence, their total cost is $O(k\,\mathsf{M}_{\mathrm{int}}(\log N))$, the number of tape movements being a negligible $O(k\log N)$. The space complexity is $O(k\log N)$ bits as well.

By computing $\gcd(F_{0,0}, N)$, we can decide whether all of $F_0, \dots, F_{k-1}$ are invertible modulo N. If this is not the case, finding one i for which $\gcd(F_i, N) > 1$ amounts to going down the tree from the root to one leaf. Suppose indeed that we have determined that $\gcd(F_{i,j}, N) > 1$ for some $j < m$. Computing $\gcd(F_{2i,j+1}, N)$ enables us to determine one of $F_{2i,j+1}$ or $F_{2i+1,j+1}$ which has a nontrivial gcd with N.

The cost of a gcd is $O(\mathsf{M}_{\mathrm{int}}(\log N)\log\log N)$. Since the tree has depth $\log k$, we compute only $\log k$ gcd's. Again, since the tree is well organized on the tape, going down the tree is done in a one pass process; so our claim on the time complexity is proved. The space complexity does not increase, concluding the proof. ∎

Our second intermediate result is the cornerstone of the algorithm; we improve the estimate of [15, Theorem 19.3] using Theorem 8.

LEMMA 13. *Let b and N be positive integers with $2 \le b < N$. Then one can compute a prime divisor of N bounded by b, or prove that no such divisor exists, in*

$$O\big(\mathsf{M}_{\mathrm{int}}(\sqrt{b}\log N) + \log b\,\mathsf{M}_{\mathrm{int}}(\log N)\log\log N\big)$$

bit operations. The space complexity is $O(\sqrt{b}\log N)$ bits.

Proof. Let $c = \lfloor\sqrt{b}\rfloor$ and let us consider the polynomials of $\mathbb{Z}/N\mathbb{Z}[X]$ given by $f(X) = X + 1 - c$ and

$$F(X) = \prod_{k=0}^{c-1}(X + ck + 1) = f(X + c^2)\cdots f(X + 2c)f(X + c).$$

Our first goal is to compute the values

$$F(0) = \prod_{k=0}^{c-1}(ck + 1) \bmod N, \ldots, F(c-1) = \prod_{k=0}^{c-1}(ck + c) \bmod N.$$

To this effect, we want to apply Theorem 8 with $n = 1$, $\alpha = c$, $\beta = 1$, and $k = c$. This can be done under suitable invertibility conditions: by Lemma 7, $O(\sqrt{b})$ integers bounded by $c^2 \le b$ must be invertible modulo N. All these integers are easily computable and less than N. Using Lemma 12, we can test whether one of these integers is not invertible modulo N and, if this is the case, find one such integer in

$$O\big(\sqrt{b}\,\mathsf{M}_{\mathrm{int}}(\log N) + \log b\,\mathsf{M}_{\mathrm{int}}(\log N)\log\log N\big)$$

bit operations. Then, by trial division, we can find a prime factor of N bounded by b in $O(\sqrt{b}\,\mathsf{M}_{\mathrm{int}}(\log N))$ bit operations; in this case, the result is proved. Thus, we can now assume that all invertibility conditions are satisfied.

By Lemma 6, computing $\mathsf{D}(c, 1, c) \in \mathbb{Z}/N\mathbb{Z}$ has a cost of $O(\sqrt{b}\,\mathsf{M}_{\mathrm{int}}(\log N))$; computing its inverse has a cost in $O(\mathsf{M}_{\mathrm{int}}(\log N)\log\log N)$. Applying Theorem 8, we deduce that all the values $F(i)$, for $i = 0, \ldots, c-1$, can be computed in time $O(\mathsf{M}_{\mathrm{int}}(\sqrt{b}\log(bN)))$; since $b < N$, this is in $O(\mathsf{M}_{\mathrm{int}}(\sqrt{b}\log N))$.

Suppose that N admits a prime factor bounded by c^2; then, some $F(i)$ is not invertible modulo N. By Lemma 12, such an i can be found in time

$$O\big(\sqrt{b}\,\mathsf{M}_{\mathrm{int}}(\log N) + \log b\,\mathsf{M}_{\mathrm{int}}(\log N)\log\log N\big).$$

Since $F(i)$ is not invertible, $\gcd(ck+i+1, N)$ is nontrivial for some $k \le c-1$. Applying Lemma 12 again, the element $ck + i + 1$ can be found in time

$$O\big(\sqrt{b}\,\mathsf{M}_{\mathrm{int}}(\log N) + \log b\,\mathsf{M}_{\mathrm{int}}(\log N)\log\log N\big).$$

By definition, $ck + i + 1 \le b$, so by trial division, we can then find a prime factor of N bounded by b in $O(\sqrt{b}\,\mathsf{M}_{\mathrm{int}}(\log N))$ bit operations.

At this point, if we have not found any prime divisor, then we have certified that N has no prime divisor in $2, \ldots, c^2$, so we finally inspect the range $c^2 + 1, \ldots, b$. Since $b - c^2 \in O(\sqrt{b})$, and since all these numbers are in $O(b)$, we can finish using trial division with the same time complexity bounds as above. The space complexity is dominated by those of Theorem 8 and Lemma 12. ∎

The proof of Theorem 11 follows from successive applications of the lemma above with increasing values of b. Starting with $b = 2$, the algorithm of Lemma 13 is run

with the same value of b until no more factors smaller than b remain in N; then the value of b is doubled. The algorithm stops as soon as $b \geq \sqrt{N}$.

The space complexity estimate comes immediately. However, analyzing the time complexity requires more work. Indeed, since there are at most $(\log N)$ prime factors of N, a rough upper bound on the runtime of the whole factorization would be $(\log N)$ times the runtime of Lemma 13 with $b = \sqrt{N}$, which is too high for our purposes. We show now that this $(\log N)$ factor can in fact be avoided, thus proving Theorem 11.

When Lemma 13 is run with a given parameter b, all prime divisors of N less than $b/2$ have already been found and divided out. Therefore the primes that can be detected are greater than $b/2$; since their product is bounded by N, we deduce that the number of runs of Lemma 13 for the parameter b is upper-bounded by $O(\log N/\log b)$.

In the complexity estimate of Lemma 13, the sum of all $\mathsf{M}_{\mathrm{int}}(\log N)\log b \log\log N$ terms is bounded by a polynomial in $(\log N)$, so its contribution in the total runtime is negligible. We are thus left with the problem of evaluating the following quantity:

$$\sum_{i=1}^{\lceil(\log N)/2\rceil} \frac{\log N}{\log(2^i)} \mathsf{M}_{\mathrm{int}}(2^{i/2}\log N) \leq \mathsf{M}_{\mathrm{int}}\left(\log N \sum_{i=1}^{\lceil(\log N)/2\rceil} \left\lceil \frac{\log N}{i} \right\rceil 2^{i/2}\right),$$

where the upper bound follows from the first assumption of (2). Then, the sum is upper-bounded by a constant times $N^{1/4}$, giving the runtime of Theorem 11.

6. Computing one term in a linear recurrence. We now address the following problem: given an $n \times n$ matrix $M(X)$ with entries in $R[X]$, all of them of degree at most 1, and a vector of initial conditions U_0, define the sequence (U_i) of elements in R^n by the linear recurrence

$$U_{i+1} = M(i+1)U_i \text{ for all } i \geq 0.$$

Our question is to compute the vector U_N for some $N \in \mathbb{N}$. In the introduction, we gave the complexity of Chudnovsky and Chudnovsky's algorithm for this task. Our modification is similar to the one for integer factorization in the previous section.

THEOREM 14. *Let N be a positive integer and $s = \lfloor \log_4 N \rfloor$, such that $2, \ldots, 2^s+1$ are units in R. Given the inverses of $\mathsf{D}(1, 2^t, 2^t)$, $t \leq s$, U_N can be computed in*

$$O\big(\mathsf{MM}(n)\sqrt{N} + n^2\,\mathsf{M}(\sqrt{N})\big)$$

operations, in space $O(n^2\sqrt{N})$. If R is effective, then the bit complexity is

$$O\big(\mathsf{MM}_R(n)\sqrt{N} + n^2\mathsf{M}_R(\sqrt{N}) + n^2\ell\sqrt{N}\min(\log N, \log n)\big),$$

using $O(n^2\ell\sqrt{N} + \mathsf{SM}_R(n) + \mathsf{S}_R(\sqrt{N}))$ bits.

Proof. The proof is divided into two steps. We begin by proving the assertion in the particular case when N is a power of 4; then we treat the general case. Dealing with powers of 4 makes it possible to control the list of elements that need to be units.

- *The case when N is a power of 4.* Let us suppose that $k = 2^s$ and $N = k^2$, so that $N = 4^s$. Let $M_k(X)$ be the $n \times n$ matrix over $R[X]$ defined by

  $$M_k(X) = M(X+k)\cdots M(X+1);$$

  then, the requested output U_N can be obtained by the equation

  $$U_N = M_k(k(k-1))\cdots M_k(k)\,M_k(0)\,U_0. \tag{5}$$

Taking $\alpha = 1$ and $\beta = k = 2^s$, Lemma 7 shows that condition $\mathsf{H}(1, 2^s, 2^s)$ of Theorem 8 is satisfied. Thus, $M_k(0), M_k(k), \ldots, M_k((k-1)k)$ can be computed within the required time and space complexities. Then, by formula (5), the result is obtained by performing $\sqrt{N}$ successive matrix-vector products, which has a cost in both time and space of $O(n^2\sqrt{N})$. There is no additional complication in the Turing model, since in the row-major representation of matrices, matrix-vector products do not involve expensive tape movements.

- *The general case.* Let $N = \sum_{i=0}^{s} N_i 4^i$ be the 4-adic expansion of N, with $N_i \in \{0, 1, 2, 3\}$ for all i. Given any $t \geq 0$, we will denote by $\lceil N \rceil^t$ the integer $\sum_{i=0}^{t-1} 4^i N_i$. Using this notation, we define a sequence $(V_t)_{0 \leq t \leq s}$ as follows: We let $V_0 = U_0$ and, for $0 \leq t \leq s$ we set

$$\begin{aligned} &V_{t+1} = M(\lceil N \rceil^t + 4^t N_t) \cdots M(\lceil N \rceil^t + 1) V_t \quad \text{if } N_t \in \{1, 2, 3\}, \\ &V_{t+1} = V_t \quad \text{if } N_t = 0. \end{aligned} \tag{6}$$

One checks that $V_t = U_{\lceil N \rceil^t}$ for all t, and in particular that $V_{s+1} = U_N$. We will compute all V_t successively. Suppose thus that the term V_t is known. If N_t is zero, we have nothing to do. Otherwise, we let $V_{t+1}^{(0)} = V_t$, and, for $1 \leq j \leq N_t$, we let

$$M^{(j)}(X) = M\left(X + \lceil N \rceil^t + 4^t(j-1)\right).$$

Then we define $V_{t+1}^{(j)}$ by

$$V_{t+1}^{(j)} = M^{(j)}(4^t) \cdots M^{(j)}(1) V_{t+1}^{(j-1)}, \quad j = 1, \ldots, N_t.$$

By (6), we have $V_{t+1}^{(N_t)} = V_{t+1}$. Thus, passing from V_t to V_{t+1} amounts to computing N_t selected terms of a linear recurrence of the special form treated in the previous case, for indices of the form $4^t \leq 4^s$. With all necessary assumptions being satisfied, using the complexity result therein and the fact that all N_t are bounded by 3 we see that the total cost of the general case is thus

$$O\left(\sum_{t=0}^{s} \left(\mathsf{MM}(n) 2^t + n^2\, \mathsf{M}(2^t)\right)\right) = O\left(\mathsf{MM}(n) 2^s + n^2 \left(\sum_{t=0}^{s} \mathsf{M}(2^t)\right)\right).$$

Using the fact that $2^s \leq \sqrt{N} \leq 2^{s+1}$ and the assumptions on the function M, we easily deduce that the whole complexity fits into the bound $O\big(\mathsf{MM}(n)\sqrt{N} + n^2\, \mathsf{M}\big(\sqrt{N}\big)\big)$, and the bound concerning the memory requirements follows. As before, porting this algorithm on a Turing machine does not raise any difficulty. □

7. Computing several terms in a linear recurrence. We now study the case when several terms in a linear recurrence are questioned; this will be applied in the next section for the computation of the Cartier–Manin operator. We use the same notation as before: $M(X)$ is an $n \times n$ matrix whose entries are degree 1 polynomials over a ring R; we consider the sequence (U_i) defined for all $i \geq 0$ by $U_{i+1} = M(i+1)U_i$, and let U_0 be a vector in R^n. Given r indices $N_1 < N_2 < \cdots < N_r$, we want to compute all the values $U_{N_i}, 1 \leq i \leq r$.

An obvious solution is to repeatedly apply Theorem 14 to each interval $[N_i, N_{i-1}]$, leading to a time complexity of

$$O\left(\mathsf{MM}(n)\sum_{i=1}^{r}\sqrt{N_i - N_{i-1}} + n^2\sum_{i=1}^{r}\mathsf{M}\left(\sqrt{N_i - N_{i-1}}\right)\right)$$

and with a memory requirement of $O(n^2 \max_{i=1}^{r}\sqrt{N_i - N_{i-1}})$, where we have set $N_0 = 0$. In special cases, this might be close to optimal, for instance if $N_1, \ldots, N_{r-1}$ are small compared to N_r. However, in most cases it is possible to improve on this: we now present an alternative algorithm, which improves on the time complexity, at the cost, however, of increased memory requirements.

THEOREM 15. *Let $N_1 < N_2 < \cdots < N_r = N$ be positive integers. Suppose that $2, \ldots, 2^s + 1$ are units in R, where $s = \lfloor \log_4 N \rfloor$, and that the inverse of $\mathsf{D}(1, 2^s, 2^s)$ is known. Suppose also that $r < N^{\frac{1}{2}-\varepsilon}$, with $0 < \varepsilon < \frac{1}{2}$. Then $U_{N_1}, \ldots, U_{N_r}$ can be computed within*

$$O\big(\mathsf{MM}(n)\sqrt{N} + n^2\mathsf{M}(\sqrt{N})\big)$$

ring operations in space $O(n^2\sqrt{N})$. If R is effective, the bit complexity is in

$$O\big(\mathsf{MM}_R(n)\sqrt{N} + n^2\mathsf{M}_R(\sqrt{N}) + n^2\ell\sqrt{N}\min(\log N, \log n)\big),$$

and the space complexity is $O(n^2\ell\sqrt{N} + \mathsf{SM}_R(n) + \mathsf{S}_R(\sqrt{N}))$ bits.

In other words, the complexity of computing several terms is essentially the same as that of computing the one of largest index, as long as the total number of terms to compute is not too large. If the N_i form an arithmetic progression, and if the multiplication function M is essentially linear, we gain a factor of $\sqrt{r}$ compared to the naive approach. In the limiting case, where $r = \sqrt{N}$, and for fixed size n, our algorithm is optimal up to logarithmic factors, as it takes a time essentially linear in the size of the output.

Proof. The algorithm proceeds as follows: we start by applying Theorem 8 with $k \approx \sqrt{N}$, so as to compute k terms in the sequence with indices separated by intervals of size about $\sqrt{N}$. If all indices N_i have been reached, then we are done. Otherwise, we have reached r indices that are within a distance of about $\sqrt{N}$ of $N_1, \ldots, N_r$. A recursive refining procedure is then done simultaneously for all these r indices. When we get close enough to the wanted indices, we finish with a divide-and-conquer method. The refining and the final steps make use of Corollary 10.

We now describe more precisely the first step (named Step 0), the ith refining step, and the final step. In what follows, given $k \in \mathbb{N}$, we denote by $M_k(X)$ the polynomial matrix $M_k(X) = M(X + k) \cdots M(X + 1)$.

Step 0. Let $k_0 = 2^s$. As a preliminary, from the inverse of $\mathsf{D}(1, 2^s, 2^s)$, we deduce using Lemma 1 the inverses of all integers $1, \ldots, k_0$ in R. This will be used later on.

Define $N_j^{(0)} = k_0\lfloor \frac{N_j}{k_0} \rfloor$ for $j \le r$, so that $N_j^{(0)} \le N_j < N_j^{(0)} + k_0$. Step 0 consists of computing $U_{N_1^{(0)}}, \ldots, U_{N_r^{(0)}}$. This is done by computing all vectors $U_{k_0}, U_{2k_0}, \ldots, U_{4k_0^2}$; this sequence contains all requested vectors, since all wanted indices are multiples of k_0 and upper-bounded by $4k_0^2$. Lemma 7 shows that the assumptions of Theorem 8 with $\alpha = 1$ and $\beta = k = k_0$ are satisfied. We use it to compute the matrices

$$M_{k_0}(0), M_{k_0}(k_0), \ldots, M_{k_0}((k_0 - 1)k_0);$$

the vector U_0 is then successively multiplied by these matrices, yielding the vectors $U_{k_0}, U_{2k_0}, \dots, U_{k_0^2}$. To complete the sequence, we repeat three times the same strategy, starting with $V_0 = U_{k_0^2}$ and shifting the matrix $M(X)$ accordingly. Among all the resulting vectors, we collect the requested values $U_{N_1^{(0)}}, \dots, U_{N_r^{(0)}}$.

Step i. We assume that after step $(i-1)$, we are given an integer k_{i-1} and indices $N_j^{(i-1)}$, where for all $j \le r$, we have $N_j^{(i-1)} \le N_j < N_j^{(i-1)} + k_{i-1}$. We also suppose that $U_{N_1^{(i-1)}}, \dots, U_{N_r^{(i-1)}}$ are known. In other words, we know r vectors whose indices are within a distance of k_{i-1} of the wanted ones. Then, the ith refining step is as follows. Set

$$k_i = \left\lceil \sqrt{r k_{i-1}} \right\rceil;$$

then the new terms that we want to compute correspond to the indices

$$N_1^{(i)} = N_1^{(i-1)} + k_i \left\lfloor \frac{N_1 - N_1^{(i-1)}}{k_i} \right\rfloor, \dots, N_r^{(i)} = N_r^{(i-1)} + k_i \left\lfloor \frac{N_r - N_r^{(i-1)}}{k_i} \right\rfloor,$$

which satisfies the induction assumption for entering step $(i+1)$. To compute these values, we evaluate the new polynomial matrix $M_{k_i}(X)$ at the points

$$\begin{aligned}
\mathbf{N}_1^{(i)} &= \left\{ N_1^{(i-1)},\ N_1^{(i-1)} + k_i, \dots, N_1^{(i-1)} + \left(\left\lfloor \frac{k_{i-1}}{k_i} \right\rfloor - 1 \right) k_i \right\} \\
&\ \ \vdots \qquad\qquad\qquad\qquad \vdots \\
\mathbf{N}_r^{(i)} &= \left\{ N_r^{(i-1)},\ N_r^{(i-1)} + k_i, \dots, N_r^{(i-1)} + \left(\left\lfloor \frac{k_{i-1}}{k_i} \right\rfloor - 1 \right) k_i \right\}.
\end{aligned}$$

There are $r\lfloor \frac{k_{i-1}}{k_i} \rfloor \le k_i$ points of evaluation. We have already computed the inverses of $1, \dots, k_0$ in R; in particular, we know the inverses of $1, \dots, k_i$, so Corollary 10 can be applied to perform the evaluation. Then, for all $j \le r$, we successively multiply $U_{N_j^{(i-1)}}$ by the values taken by $M_{k_i}(X)$ at all indices in $\mathbf{N}_j^{(i)}$. By construction, $N_j^{(i)} - k_i$ belongs to $\mathbf{N}_j^{(i)}$, so we obtain in particular the requested value $U_{N_j^{(i)}}$.

Final step. The refining process stops when k_i is close to r, namely $k_i \le 2r$. In this situation, we have at our disposal r indices $N_1', \dots, N_r'$ such that $N_j' \le N_j < N_j' + 2r$ for all j, and such that all values $U_{N_1'}, \dots, U_{N_r'}$ are known. Then another recursive algorithm is used: $M_r(X)$ is computed, evaluated at all N_j' using Corollary 10, and used to reduce all gaps that were of size between r and $2r$; again, the invertibility conditions cause no problem. As a result, all the gaps are now of size at most r. Then $M_{\frac{r}{2}}(X)$ is computed and used to reduce the gaps to at most $\frac{r}{2}$, and so on until we get the result.

It remains to perform the complexity analysis. The cost of Step 0 is dominated by that of evaluating the matrix $M_{k_0}(X)$; by Theorem 8, it can be done in

$$O\big(\mathsf{MM}(n)\sqrt{N} + n^2 \mathsf{M}(\sqrt{N})\big)$$

operations in R, and space $O(n^2\sqrt{N})$. By Corollary 10, the ith refining step has a cost of

$$O\left(\mathsf{MM}(n) k_i + n^2 \mathsf{M}(k_i) \log k_i\right)$$

operations, and requires $O(n^2 k_i + k_i \log k_i)$ temporary memory allocation; the total cost of this phase is thus

$$O\left(\sum_{i=1}^{i_{\max}} \mathsf{MM}(n)k_i + n^2\mathsf{M}(k_i)\log k_i\right),$$

where $i_{\max}$ is the number of refining steps. Our hypotheses on the function M show that this fits into the bound

$$O\left(\mathsf{MM}(n)\sum_{i=1}^{i_{\max}} k_i + n^2\mathsf{M}\left(\sum_{i=1}^{i_{\max}} k_i\right)\log\left(\sum_{i=1}^{i_{\max}} k_i\right)\right).$$

An easy induction shows that k_i is at most $(4r)^{\frac{2^i-1}{2^i}} N^{\frac{1}{2^{i+1}}}$, whence $i_{\max} = O(\log\log N)$. Furthermore, using $r < N^{\frac{1}{2}-\varepsilon}$, we get

$$k_i \le 4N^{\frac{1}{2}}\left(N^{-\varepsilon}\right)^{\frac{2^i-1}{2^i}}.$$

For any $0 < \ell \le 1$ we have $\sum_{i=1}^{i_{\max}} \ell^{\frac{2^i-1}{2^i}} \le i_{\max}\ell^{\frac{1}{2}}$; therefore we deduce

$$\sum_{i=1}^{i_{\max}} k_i = O\big(N^{\frac{1-\varepsilon}{2}}\log\log N\big).$$

Using the second assumption in (2), the cost of the refining steps is negligible compared with that of the first step, since the $N^{-\varepsilon/2}$ compensates for all logarithmic factors. The memory requirement also is negligible compared with that of Step 0.

Using Corollary 10, the cost of the first reduction in the final step is

$$O\left(\mathsf{MM}(n)r + n^2\mathsf{M}(r)\log r\right)$$

ring operations and $O(n^2 r + r\log r)$ temporary space. Due to our hypotheses on the function M, the second reduction costs at most half as much, the third reduction costs at most $\frac{1}{4}$ of it, etc. Summing up, we see that the whole cost of the final step is bounded by twice that of the first reduction, which is itself less than the cost of Step 0.

There is no complication in the Turing model. Indeed, between the calls to Theorem 8, to Corollary 10, and the successive matrix-vector products, there is no need to reorganize data, so the tape movements' cost is negligible. Furthermore, our assumptions on M_R and S_R imply that, as in the arithmetic model, the time and space costs of Step 0 are predominant. □

8. Application to the Cartier–Manin operator. Let $\mathcal{C}$ be a hyperelliptic curve of genus g defined over the finite field $\mathbb{F}_{p^d}$ with p^d elements, where $p > 2$ is prime. We suppose that the equation of $\mathcal{C}$ is of the form $y^2 = f(x)$, where f is monic and square free, of degree $2g+1$. The generalization to hyperelliptic curves of the Hasse invariant for elliptic curves is the Hasse–Witt matrix [21]: Let h_k be the coefficient of degree x^k in the polynomial $f^{(p-1)/2}$. The Hasse–Witt matrix is the $g \times g$ matrix with coefficients in $\mathbb{F}_{p^d}$ given by $H = (h_{ip-j})_{1\le i,j\le g}$. It represents, in a suitable basis, the operator on differential forms introduced by Cartier [10]; Manin [28] showed that this matrix is strongly related to the action of the Frobenius endomorphism on

the p-torsion part of the Jacobian of $\mathcal{C}$. The article [50] provides a complete survey about these facts; they are summarized in the following theorem.

THEOREM 16 (Manin). *Let $H_\pi = HH^{(p)} \cdots H^{(p^{d-1})}$, where the notation $H^{(q)}$ means elementwise raising to the power q. Let $\kappa(t)$ be the characteristic polynomial of the matrix H_π and $\chi(t)$ the characteristic polynomial of the Frobenius endomorphism of the Jacobian of $\mathcal{C}$. Then $\chi(t) \equiv (-1)^g t^g \kappa(t) \bmod p$.*

This result provides a method to compute the characteristic polynomial of the Frobenius endomorphism. As such, it can be used in a point-counting algorithm, which is the main application we have in mind; see the end of this section for more comments.

To compute the entries of the Hasse–Witt matrix, the obvious solution consists of expanding the product $f^{(p-1)/2}$. Using binary powering, this can be done in $O(\mathsf{M}(gp))$ base ring operations, whence a time complexity that is essentially linear in p, if g is kept constant. In what follows, we show how to obtain a complexity essentially linear in $\sqrt{p}$ using the results of the previous sections. We will make the additional assumption that the constant term of f is not zero; otherwise, the problem is actually simpler.

Introduction of a linear recurrent sequence. In [14], Flajolet and Salvy already treated the question of computing a selected coefficient in a high power of some given polynomial, as an answer to a SIGSAM challenge. The key point of their approach is that $h = f^{(p-1)/2}$ satisfies the following first-order linear differential equation

$$fh' - \frac{p-1}{2} f'h = 0.$$

This shows that coefficients of h satisfy a linear recurrence of order $2g+1$, with polynomial coefficients of degree 1. Explicitly, denote by h_k the coefficient of degree k of h, and for convenience, set $h_k = 0$ for $k < 0$. Similarly, the coefficient of degree k of f is denoted by f_k. Then the differential equation above implies that, for all k in $\mathbb{Z}$,

$$(k+1)f_0 h_{k+1} + \left(k - \frac{p-1}{2}\right) f_1 h_k + \cdots + \left(k - 2g - \frac{(2g+1)(p-1)}{2}\right) f_{2g+1} h_{k-2g} = 0.$$

We set $U_k = [h_{k-2g}, h_{k-2g+1}, \ldots, h_k]^t$, and let $A(k)$ be the companion matrix:

$$A(k) = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & 1 \\ \frac{f_{2g+1}((2g+1)(p-1)/2-(k-2g-1))}{f_0 k} & \cdots & \cdots & \cdots & \frac{f_1((p-1)/2-k+1)}{f_0 k} \end{pmatrix}.$$

The initial vector $U_0 = [0, \ldots, 0, f_0^{(p-1)/2}]^t$ can be computed using binary powering techniques in $O(\log p)$ operations; then for $k \geq 0$, we have $U_{k+1} = A(k+1)U_k$. Thus, to answer our specific question, it suffices to note that the vector U_{ip-1} gives the coefficients h_{ip-j} for $j = 1, \ldots, g$ that form the ith row of the Hasse–Witt matrix of $\mathcal{C}$.

Theorems 14 and 15 cannot be directly applied to this sequence, because $A(k)$ has entries that are rational functions, not polynomials. Though the algorithm could be adapted to handle the case of rational functions, we rather use the very specific form of the matrix $A(k)$, so only a small modification is necessary. Let us define a new

sequence V_k by $V_k = f_0^k k! U_k$. Then, this sequence is linearly generated and we have $V_{k+1} = B(k+1)V_k$, where $B(k) = f_0 k A(k)$. Therefore, the entries of the matrix $B(k)$ are polynomials of degree at most 1. Note also that the denominators $f_0^k k!$ satisfy the recurrence relation $f_0^{k+1}(k+1)! = (f_0(k+1)) \cdot (f_0^k k!)$.

We can apply the results of the previous sections to compute separately the required vectors V_k, as well as the corresponding denominators: specifically, we will compute $V_{p-1}, V_{2p-1}, \ldots, V_{gp-1}$ as well as $f_0^{p-1}(p-1)!, \ldots, f_0^{gp-1}(gp-1)!$. Then the vectors $U_{p-1}, U_{2p-1}, \ldots, U_{gp-1}$ are deduced using the relation $f_0^k k!\, U_k = V_k$.

Lifting to characteristic zero. A difficulty arises from the fact that the characteristic is too small compared to the degrees we are aiming for, so $p!$ is zero in $\mathbb{F}_{p^d}$. The workaround is to do computations in the unramified extension K of $\mathbb{Q}_p$ of degree d, whose residual field is $\mathbb{F}_{p^d}$. The ring of integers of K will be denoted by $\mathcal{O}_K$, so that any element of $\mathcal{O}_K$ can be reduced modulo p to give an element of $\mathbb{F}_{p^d}$. On the other hand, K has characteristic 0, so p is invertible in K.

We consider an arbitrary lift of f to $\mathcal{O}_K[X]$. The reformulation in terms of linear recurrent sequence made in the paragraph above can be performed over K; the coefficients of $f^{(p-1)/2}$ are computed as elements of K and then projected back onto $\mathbb{F}_{p^d}$. This is possible, as they all belong to $\mathcal{O}_K$. We separately compute the values in K of the vectors V_{ip-1} and the denominators $f_0^{ip-1}(ip-1)!$, for $i = 1, \ldots, g$. To this effect, we can apply any of the strategies mentioned in section 7.

The first one is the plain iteration based on Theorem 14; iterating g times, it performs $O\left(\mathsf{MM}(g) g\sqrt{p} + g^3 \mathsf{M}(\sqrt{p})\right)$ operations in K and requires storing $O(g^2\sqrt{p})$ elements. The second strategy is to apply Theorem 15. In this case, we need to have $g \leq (gp)^{1/2-\varepsilon'}$, for some $0 < \varepsilon' < \frac{1}{2}$; this is equivalent to imposing that $g \leq p^{1-\varepsilon}$ for some $0 < \varepsilon < 1$. Then, with increased memory requirements of $O(g^2\sqrt{gp})$, the number of operations in K reduces to $O\left(\mathsf{MM}(g)\sqrt{gp} + g^2 \mathsf{M}(\sqrt{gp})\right)$.

Computing at fixed precision. We do not want to compute in K at arbitrary precision, since this is not an effective ring (even in a much weaker sense than the one we defined). For our purposes, it suffices to truncate all computations modulo a suitable power of p. To evaluate the required precision of the computation, we need to check when the algorithm operates a division by p.

To compute the vectors V_{ip-1} and the denominators $f_0^{ip-1}(ip-1)!$, for $i = 1, \ldots, g$, we use either Theorem 14 or Theorem 15. In the worst case, it might be required to invert all integers up to $\sqrt{gp}$. With the condition $g \leq p^{1-\varepsilon}$, these numbers are strictly smaller than p, and they are thus units in $\mathcal{O}_K$. Hence no division by p occurs in this first phase. Then, for all $i = 1, \ldots, g$, to deduce U_{ip-1} from V_{ip-1}, we need to divide by $f_0^{ip-1}(ip-1)!$. The element f_0 is a unit in $\mathcal{O}_K$, so the only problem comes from the factorial. If $i < p$, then the p-adic valuation of $(ip-1)!$ is exactly $i-1$. Therefore the worst case is $i = g$, for which we have to divide by p^{g-1}. Hence computing the vectors V_{ip-1} modulo p^g is enough to know the vectors U_{ip-1} modulo p, and then to deduce the Hasse–Witt matrix.

Overall complexity. Since the ring $R = \mathcal{O}_K/p^g\mathcal{O}_K$ is isomorphic to $(\mathbb{Z}/p^g\mathbb{Z})[X]/P$ for some monic polynomial P of degree d, Lemma 3 shows that R is an effective ring with

- $\ell = d\lceil g \log p \rceil$,
- $\mathsf{m}_R \in O(\mathsf{M}_{\mathrm{int}}(dg \log(dp)))$ and $\mathsf{s}_R \in O(dg \log(dp))$,
- $\mathsf{M}_R(k) \in O(\mathsf{M}_{\mathrm{int}}(dkg \log(dkp)))$ and $\mathsf{S}_R(k) \in O(dkg \log(dkp))$,
- $\mathsf{MM}_R(n) \in O(n^{\log 7} \mathsf{m}_R)$ and $\mathsf{SM}_R(n) \in O(n^2\ell + \mathsf{s}_R)$.

From the results of sections 6 and 7, we deduce the following theorem on the complexity of computing the Hasse–Witt matrix. We give two variants that follow the two strategies described above. The first one follows from applying g times Theorem 14 with $n = 2g+1, N = p, \ell = d\lceil g \log p \rceil$; the second one come from applying Theorem 15 with $n = 2g+1, N = gp, \ell = d\lceil g \log p \rceil$. Both theorems require us to compute the inverses of some integers in R as prerequisites; the cost of their computation is negligible.

THEOREM 17. *Let $p > 2$ be a prime, $d \geq 0$, and $\mathcal{C}$ be a hyperelliptic curve defined over $\mathbb{F}_{p^d}$ by the equation $y^2 = f(x)$, with f of degree $2g+1$. Assuming $g < p^{1-\varepsilon}$ for some $0 < \varepsilon < 1$, one can compute the Hasse–Witt matrix of $\mathcal{C}$ using one of the two following strategies:*

1. *A memory-efficient strategy gives a complexity of*

$$O\left(g^{1+\log 7} p^{\frac{1}{2}} \mathsf{M}_{\mathrm{int}}(dg \log(dp)) + g^3 \mathsf{M}_{\mathrm{int}}(dgp^{\frac{1}{2}} \log(dp)) + dg^4 p^{\frac{1}{2}} \log g \log p\right)$$

   bit operations and $O(dg^3 p^{\frac{1}{2}} \log p + dgp^{\frac{1}{2}} \log d)$ storage.
2. *A time-efficient strategy gives a complexity of*

$$O\left(g^{\frac{1}{2}+\log 7} p^{\frac{1}{2}} \mathsf{M}_{\mathrm{int}}(dg \log(dp)) + g^2 \mathsf{M}_{\mathrm{int}}(dg^{\frac{3}{2}} p^{\frac{1}{2}} \log(dgp)) + dg^{\frac{7}{2}} p^{\frac{1}{2}} \log g \log p\right)$$

   bit operations, with $O(dg^{\frac{7}{2}} p^{\frac{1}{2}} \log p + dg^{\frac{3}{2}} p^{\frac{1}{2}} \log d)$ storage.

The matrix H already gives information on the curve $\mathcal{C}$: for instance, H is invertible if and only if the Jacobian of $\mathcal{C}$ is ordinary [50, Corollary 2.3]. However, as stated in Theorem 16, the matrix H_π, and in particular its characteristic polynomial κ, tells much more and is required if the final goal is point-counting.

From now on, all operations are done in the effective ring $R' = \mathbb{F}_{p^d}$; hence the cost of the basic operations becomes $\mathsf{m}_{R'} \in O(\mathsf{M}_{\mathrm{int}}(d \log(dp)))$ and $\mathsf{s}_{R'} \in O(d \log(dp))$. The matrix H_π is the "norm" of H and as such can be computed with a binary powering algorithm. For simplicity, we assume that d is a power of 2; then, denoting

$$H_{\pi,i} = H H^{(p)} \cdots H^{(p^{2^i-1})},$$

we have

$$H_{\pi,i+1} = H_{\pi,i} \cdot (H_{\pi,i})^{(p^{2^i})}.$$

Hence computing $H_{\pi,i+1}$ from $H_{\pi,i}$ costs one matrix multiplication and 2^i matrix conjugations. A matrix conjugation consists of raising all the entries to the power p; therefore it costs $O(g^2 \mathsf{M}_{\mathrm{int}}(d \log(dp)) \log p)$ bit operations. The matrix we need to compute is $H_\pi = H_{\pi, \log d}$. Hence the cost of computing H_π is

$$O\left(\left(dg^2 \log p + g^{\log 7} \log d\right) \mathsf{M}_{\mathrm{int}}(d \log(dp))\right)$$

bit operations. The general case, where d is not a power of 2, is handled by adjusting the recursive step according to the binary expansion of d and yields the same complexity up to a constant factor.

The cost of the characteristic polynomial computation of an $n \times n$ matrix defined over an effective ring can be bounded by $O(n^4)$ operations in the ring using a sequential version of Berkowitz's algorithm [1]. This adds a negligible $O(g^4 \mathsf{M}_{\mathrm{int}}(d \log(dp)))$ contribution to the complexity.

If we are interested only in the complexity in p and d, i.e., if we assume that the genus is fixed, then the two strategies of Theorem 17 become equivalent, up to constant factors. Then, to summarize, the reduction modulo p of the characteristic polynomial χ of the Frobenius endomorphism can be computed in time

$$O\big(\mathsf{M}_{\mathrm{int}}(dp^{\frac{1}{2}}\log(dp)) + d\log p\,\mathsf{M}_{\mathrm{int}}(d\log(dp))\big)$$

bit operations and $O(dp^{\frac{1}{2}}\log(dp))$ storage.

Case of large genus. If $g \geq p$, then our analysis is no longer valid. In this paragraph, we assume that the function $\mathsf{M}_{\mathrm{int}}$ is essentially linear, i.e., we do not count logarithmic factors. Then the cost of strategy 1 of Theorem 17 is of order $dg^4p^{\frac{1}{2}}$ bit operations, the cost of strategy 2 is of order $dg^{\frac{7}{2}}p^{\frac{1}{2}}$, and that of the naive algorithm is in $O(dgp)$.

It turns out that for $p^{1/6} < g < p^{1-\varepsilon}$, the algorithms of Theorems 14 and 15 are not the fastest. Assume that $g > p^{1/6}$. Then it follows that $g^4p^{\frac{1}{2}} > gp$, and therefore the naive algorithm is faster than strategy 1 of Theorem 17. If further $g > p^{1/5}$, then $g^{\frac{7}{2}}p^{\frac{1}{2}} > gp$, and the naive algorithm is also faster than strategy 2. Thus, whatever the strategy used in Theorem 17, the parameter range for which our algorithms are interesting is far from the limit induced by the technical condition $g < p^{1-\varepsilon}$.

Combination with other point-counting algorithms. Computing the characteristic polynomial of H_π is not enough to deduce the group order of the Jacobian of $\mathcal{C}$, since only $\chi \bmod p$ is computed. We now survey different ways to complete the computation; we give rough complexity estimates, neglecting the logarithmic factors.

If p is small compared to g or d, p-adic algorithms [24, 37] have the best asymptotic complexity. These algorithms compute χ modulo high powers of p, so they necessarily recompute the information that has been obtained via the Cartier–Manin operator. Hence, our approach is of no interest here.

Consider next the extensions of Schoof's algorithm [34]. These algorithms have a complexity that is polynomial in $d\log p$ and exponential in g. For fixed g, our algorithm will be faster only if p is small compared to d, so that the power of d in the complexity of Schoof's algorithm can compensate the $\sqrt{p}$ complexity of our method. But in that case, our algorithm gives only very small information, and therefore the overall complexity of point counting is unchanged.

The combination with approaches based on the baby steps/giant steps algorithm (or low memory variants) is more fruitful, and can be of practical interest. Indeed, as far as we know, there is no implementation of any Schoof-like approach for genus greater than 2, and even for genus 2, the current record computations [19, 29] are obtained by combining many methods, including the baby steps/giant steps approach. Here is thus a short description of known approaches using BSGS ideas:

1. *BSGS method*: This is the generic method for finding the order of a group. If the order is known to be in an interval of width w, then the complexity is in $O(\sqrt{w})$. In the case of the Jacobian of $\mathcal{C}$, Hasse–Weil bounds give $w = O(p^{d(g-\frac{1}{2})})$, so the complexity is in $O(p^{d(\frac{g}{2}-\frac{1}{4})})$.
2. Computing the number of points of $\mathcal{C}$ in small degree extensions of $\mathbb{F}_{p^d}$ reduces the width of the search interval. Counting (naively) the points of $\mathcal{C}$ up to extension degree k costs $O(p^{dk})$, and the cost of the BSGS algorithm becomes $O(q^{d(\frac{g}{2}-\frac{k+1}{4})})$. This method (and additional practical improvements) is from [45]. We call it "approximation method" below.
3. When χ is known modulo some integer M, the group order is also known modulo M and therefore the BSGS method can be sped up by a factor of

$\sqrt{M}$. In [29] it is shown that in some cases a full factor M can be gained by doing a BSGS search on the coefficients of χ instead of just the group order. We abbreviate this method as MCT (from the names of the authors).

Thus, in genus 2, the complexity of the BSGS algorithm is in $O(p^{\frac{3d}{4}})$. For prime fields, our $O(p^{\frac{1}{2}})$ method is faster and gives essentially the complete information. For extension fields, our method gives the characteristic polynomial modulo p at a cost of $O(p^{\frac{1}{2}})$, from which we can recover the whole characteristic polynomial using the MCT algorithm at a cost of $O(p^{\frac{3d}{4}-1})$. Thus, for $d = 2$, the complexity is improved from $O(p^{\frac{3}{2}})$ to $O(p^{\frac{1}{2}})$, and for $d = 3$, from $O(p^{\frac{9}{4}})$ to $O(p^{\frac{5}{4}})$.

In genus 3, using the approximation method with $k = 1$ yields a complexity in $O(p^d)$. For prime fields, our method yields most of the information, the remaining part being computable using BSGS in time $O(p^{\frac{1}{4}})$. Hence, the cost drops from $O(p)$ to $O(p^{\frac{1}{2}})$; this is of practical interest, since the $O(p)$ algorithm is currently used for genus 3 point-counting over prime fields. For extension fields, the complexity drops from $O(p^d)$ to $O(p^{d-\frac{1}{2}})$.

The complexities for small degrees and genera are summarized in the following table. For each parameter set (g, d), there are two columns: the left-hand column describes the previously best known combination of methods; the right-hand one gives the new best combination with our algorithm (written "CM"). In each column we put an X in front of the algorithms that are used in the combination, and at the bottom list the total complexity.

	$g=2$						$g=3$					
	$d=1$		$d=2$		$d=3$		$d=1$		$d=2$		$d=3$	
BSGS	X		X		X		X	X	X	X	X	X
Approx.							X		X	X	X	X
MCT				X		X						
CM		X		X		X		X		X		X
Cplx.	$p^{3/4}$	$p^{1/2}$	$p^{3/2}$	$p^{1/2}$	$p^{9/4}$	$p^{5/4}$	p	$p^{1/2}$	p^2	$p^{3/2}$	p^3	$p^{5/2}$

Computer experiments. We have implemented our algorithm using Shoup's NTL C++ library [42]. NTL does not provide any arithmetic of local fields or rings, but allows one to work in finite extensions of rings of the form $\mathbb{Z}/p^g\mathbb{Z}$, as long as no divisions by p occur; the divisions by p are well isolated in the algorithm, so we could handle them separately. Furthermore, NTL multiplies polynomials defined over this kind of structure using an asymptotically fast FFT-based algorithm.

To illustrate that our method can be used as a tool in point-counting algorithms, we have computed the Zeta function of a (randomly chosen) genus 2 curve defined over $\mathbb{F}_{p^3}$, with $p = 2^{32} - 5$. Such a Jacobian has therefore about 2^{192} elements and should be suitable for cryptographic use if the group order has a large prime factor. Note that previous computations were limited to p of order 2^{23} [29].

The characteristic polynomial χ of the Frobenius endomorphism was computed modulo p in 3 hours and 41 minutes, using 1 GB of memory, on an AMD Athlon MP 2200+. Then we used the Schoof-like algorithms of [19] to compute χ modulo $128 \times 9 \times 5 \times 7$, and finally we used the modified BSGS algorithm of [29] to finish the computation. These other parts were implemented in Magma [5] and were performed in about 15 days of computation on an Alpha EV67 at 667 MHz. This computation was meant as an illustration of the possible use of our method, so little time was spent optimizing our code. In particular, the Schoof-like part and the final BSGS computations are done using a generic code that is not optimized for extension fields. Still, to our knowledge, on the same computers, such a computation would not have been possible with previous algorithms.

Appendix. Computations in the Turing model. In this appendix we discuss basic complexity results for polynomials and matrices over effective rings, in the multitape Turing machine model. We do not consider the operations used to control the computations, like incrementing an index in a loop: this is done on separate tapes, and the corresponding cost is negligible.

Proof of Lemma 3. Let $N \in \mathbb{N}$, $R_0 = \mathbb{Z}/N\mathbb{Z}$, and $R = R_0[T]/P$, with $P \in R_0[T]$ monic of degree m. We show here how to make R an effective ring. Elements of R_0 will be represented as integers in $0, \ldots, N-1$, and elements of R as sequences of m elements of R_0; representing such an element requires $\ell = m\lceil \log N \rceil$ bits.

Polynomials in $R_0[T]$ are multiplied as polynomials in $\mathbb{Z}[T]$; then their coefficients are reduced modulo N. Using Kronecker's substitution [15, Corollary 8.27], the multiplication in degree d is done in time $\mathsf{M}_{\mathrm{int}}(d\log(dN))$ and space $O(d\log(dN))$; the subsequent reduction is done by fast integer Euclidean division, using Cook's algorithm [12], which adds a negligible cost. Using Cook's algorithm again, Euclidean division in degree d in $R_0[T]$ can be done in time $O(\mathsf{M}_{\mathrm{int}}(d\log(dN)))$ and space $O(d\log(dN))$. In particular, taking $d = m$, this establishes the bounds on m_R and s_R given in the lemma.

Polynomials in $R[X]$ are multiplied as polynomials in $\mathbb{Z}[T, X]$, and then reduced modulo N and P, where the product in $\mathbb{Z}[T, X]$ is reduced to an integer product by bivariate Kronecker's substitution. In degree d, this yields time and space complexities M_R and S_R of, respectively, $O(\mathsf{M}_{\mathrm{int}}(dm\log(dmN)))$ and $O(dm\log(dmN))$.

We finally discuss matrix multiplication, contenting ourselves with the description of Strassen's algorithm [46, 15] for matrices of size $n = 2^k$ (which is enough to establish our claim). Each step of the algorithm requires us to compute 14 linear combinations of the 4 quadrants of the input matrices before entering recursive calls; 4 linear combinations of the 7 subproducts are performed after the recursive calls.

At each step in the recursion, the data has to be reorganized. The row-major representation of each input matrix is replaced with the consecutive row-major representations of its four quadrants, from which the linear combinations can be performed; a similar unfolding is done after the recursive calls. Taking into account the cost of this reorganization does not alter the complexity of this algorithm. This yields estimates for MM_R and SM_R, respectively, in $O(n^{\log 7}\mathsf{m}_R)$ and $O(n^{\log 7}\ell + \mathsf{s}_R)$, the term s_R standing for temporary memory used for scalar multiplications.

Finally, checking all required conditions on m_R, s_R, M_R, S_R, MM_R, and SM_R is straightforward.

Proof of Lemma 4. Let now R be an effective ring, with elements represented on ℓ bits. We prove here the assertions in Lemma 4.

- Trading inverses for multiplications: proof of Lemma 4, item 1. We use the notation of the proof of Lemma 1. Looking at the proof, one sees that all quantities R_i can be computed and stored on a tape T_1 in a single forward sweep of the input $r_0, \ldots, r_d$; reading the input backward, we compute all S_i and store them on a tape T_2. Finally, the output values s_i are computed by a single forward sweep of T_1 and T_2. The time complexity is $O(d\mathsf{m}_R)$ for multiplications, plus $O(d\ell)$ for tape movements; hence it fits in $O(d\mathsf{m}_R)$. The space complexity is $O(d\ell)$ bits for storage, plus s_R temporary bits for multiplications.
- Computing constants: proof of Lemma 4, items 2 and 3. We apply the same formulas as in the proof of Lemma 2. The cost of all operations is in $O(d\mathsf{m}_R)$; it is easy to check that the tape movements contribute with a negligible $O(d\ell)$ cost. As above, the space complexity is $O(d\ell)$ bits for storage,

plus s_R temporary bits for multiplications.

- Evaluation and interpolation: proof of Lemma 4, items 4 and 5. Let $r_0, \dots, r_d$ be in R. For simplicity, we suppose that the number of points is a power of 2, that is, $d+1 = 2^k$; the general case is handled similarly and presents only notational difficulties. All algorithms below are classical [15, Chapter 10]; our focus is on their adaptation in the Turing model.

For $i \le d$, set $A_{i,k} = X - r_i \in R[X]$; then, for $0 \le j \le k-1$ and $0 \le i \le 2^j - 1$, set $A_{i,j} = A_{2i,j+1}A_{2i+1,j+1}$. These polynomials will be arranged in a "subproduct tree," where $A_{2i,j+1}$ and $A_{2i+1,j+1}$ are the children of $A_{i,j}$. We now show how to compute this tree, writing A_j for the sequence $A_{0,j}, \dots, A_{2^j-1,j}$. Note that the sum of the degrees of the polynomials in A_j is $d+1$.

Given the sequence A_j, one can compute the extended sequence A_j, A_{j-1} in $O(\mathsf{M}_R(d))$ bit operations and space $O(d\ell + \mathsf{S}_R(2^{k-j}))$. It suffices to read the input once and to compute on the fly the products $A_{2i,j}A_{2i+1,j}$, storing them on an auxiliary tape, before appending all results to the input; the cost estimate follows from the superadditivity of M_R. Applying this $k = \log d$ times, one can compute the sequences $A_k, \dots, A_0$ in time $O(\mathsf{M}_R(d)\log d)$ and space $O(\ell d \log d + \mathsf{S}_R(d))$.

The evaluation algorithm uses the subproduct tree as follows. Let $P = P_{0,0}$ be of degree at most d, and set $P_{2i,j+1} = P_{i,j} \bmod A_{2i,j+1}$ and $P_{2i+1,j+1} = P_{i,j} \bmod A_{2i+1,j+1}$, for $0 \le i \le 2^j - 1$ and $0 \le j \le k$. We write P_j for the sequence $P_{0,j}, \dots, P_{2^j-1,j}$.

On input the sequences P_j and A_{j+1}, given on two distinct tapes, one can compute P_j, P_{j+1} in $O(\mathsf{M}_R(d))$ bit operations and space $O(\ell d + \mathsf{S}_R(2^{k-j}))$: we read once the input sequences and compute on the fly the remainders $P_{2i,j+1}$ and $P_{2i+1,j+1}$, storing them on an auxiliary tape; then they are appended to the sequence P_j. The estimates for Euclidean division and the superadditivity property then give the complexity estimate. Applying this $k = \log d$ times, given P and the sequence $A_k, \dots, A_0$, one can compute all $P_{i,k} = P(r_i)$ in time $O(\mathsf{M}_R(d)\log d)$ and space $O(\ell d \log d + \mathsf{S}_R(d))$.

It remains to deal with interpolation. Difficulties come from the inversion of quantities associated with the sample points. We thus suppose that $a_i = i$ for all $i \le d$ (this is what is used in this article), that $2, \dots, d$ are units in R, and that their inverses are known. Interpolating a polynomial P at $0, \dots, d$ is done by computing $\sum_{i \le d} P_i \prod_{j \ne i}(X - j)$, where $P_i = P(i)/\delta(i,d)$. The inverses of all $\delta(i,d)$ can be computed in time $O(d\mathsf{m}_R)$, by Lemma 4, item 2. Then, from [15, Chapter 10], the sum can be computed "going up" the subproduct tree, just as evaluation amounts to "going down" the tree. One checks that as above, it can be performed in time $O(\mathsf{M}_R(d)\log d)$ and space $O(\ell d \log d + \mathsf{S}_R(d))$.

Matrix of vectors and vectors of matrices. Let R be an effective ring, with elements represented using ℓ bits. Two representations for matrices with vector entries are used in this paper:

1. the row-major representation, where each entry is a vector over R, say of size k;
2. the vector representation, through a sequence of k scalar matrices, each in row-major representation.

In the Turing model, we must take care of data contiguity. We now give an algorithm that converts efficiently from one representation to the other; we start with a lemma on matrix transposition.

LEMMA 18. *In row-major representation, the transpose of an $m \times n$ matrix A can be computed in bit complexity $O(\ell mn \min(\log m, \log n))$ and space $O(\ell mn)$.*

Proof. Suppose that $n \leq m$. We first copy A from the input tape to an auxiliary tape, and pad on the fly the end of each line with an arbitrary symbol to make the column dimension equal to a power of 2. Thus we obtain an $m \times n'$ matrix A', where n' is a power of 2; the cost is in $O(mn\ell)$. We describe now a recursive algorithm that transposes A'; the transpose of A can be deduced as the top-left $n \times m$ submatrix of the transpose of A', and it can be copied on the output tape at a cost of $O(mn\ell)$.

We are thus reduced to transposing an $m \times n$ matrix A with n a power of 2. First, note that if $n = 1$, then the representations of A and of its transpose are the same. For $n \geq 2$ we proceed as follows. Let A_1 be the submatrix of A formed of the $n/2$ first coefficients of each row and A_2 the submatrix of A formed of their $n/2$ last coefficients. Then, the row-major representation of the transpose of A is the row-major representation of the transpose of A_1 followed by the row-major representation of the transpose of A_2. Hence computing the transpose of A amounts to the following operations:

- Uninterleaving: put A_1 followed by A_2 on a tape in place of the original A, using a temporary auxiliary tape.
- Recursively call to replace A_1 by its transpose at the same place, using a temporary auxiliary tape, and do the same with A_2.

The number $T(m, n)$ of tape movements verifies an equation of the form

$$T(m, n) \leq \lambda \ell mn + 2T(m, n/2),$$

for some constant λ. Therefore the overall cost is $O(\ell mn \log n)$ and the number of cells visited on each tape is at most ℓmn. This concludes the proof in the case $n \leq m$.

In the case $m \leq n$, we use essentially the same recursive algorithm but with the matrix split in two blocks of complete rows. Hence the algorithm for size m decomposes in two recursive calls at size $m/2$ and one subsequent step of interleaving the resulting matrices. In this way the $\log n$ factor is replaced with $\log m$. ∎

COROLLARY 19. *Let M be an $n \times n$ matrix, with entries in R^k. Switching between the two possible representations of M has bit complexity in $O(\ell n^2 k \min(\log n, \log k))$ and space complexity in $O(\ell n^2 k)$.*

Proof. Let M be represented on tape as a matrix of vectors. We can see the data of this tape as the row-major representation of an $n^2 \times k$ matrix over R. Let us compute the transpose of this matrix using the algorithm of Lemma 18. We obtain the representation of a $k \times n^2$ matrix over R; for $i \leq k$, its ith entry is the row-major representation of the $n \times n$ matrix made of the ith entries of M. ∎

Acknowledgments. We thank Bruno Salvy for his comments on this paper, Joachim von zur Gathen and Jürgen Gerhard for answering our questions on the complexity of integer factorization, and the referees for their helpful comments.

REFERENCES

[1] J. BERKOWITZ, *On computing the determinant in small parallel time using a small number of processors*, Inform. Process. Lett., 18 (1984), pp. 147–150.

[2] J. L. BORDEWIJK, *Inter-reciprocity applied to electrical networks*, Appl. Sci. Res. B., 6 (1956), pp. 1–74.

[3] A. BORODIN, *Time space tradeoffs (getting closer to the barrier?)*, in Proceedings of the 4th International Symposium on Algorithms and Computation, Lecture Notes in Comput. Sci. 762, Springer-Verlag, London, 1993, pp. 209–220.

[4] A. BORODIN AND R. T. MOENCK, *Fast modular transforms*, Comput. Systems Sci., 8 (1974), pp. 366–386.

[5] W. Bosma, J. Cannon, and C. Playoust, *The Magma algebra system.* I. *The user language*, J. Symbolic Comput., 24 (1997), pp. 235–265. See also http://www.maths.usyd.edu.au.

[6] A. Bostan, P. Gaudry, and É. Schost, *Linear recurrences with polynomial coefficients and computation of the Cartier-Manin operator on hyperelliptic curves*, in Proceedings of the International Conference on Finite Fields and Applications (Toulouse, 2003), Lecture Notes in Comput. Sci. 2948, Springer, Berlin, 2004, pp. 40–58.

[7] A. Bostan, G. Lecerf, and É. Schost, *Tellegen's principle into practice*, in Proceedings of the International Conference on Symbolic and Algebraic Computation, ACM Press, New York, 2003, pp. 37–44.

[8] P. Bürgisser, M. Clausen, and M. A. Shokrollahi, *Algebraic Complexity Theory*, Grundlehren der Math. Wiss. 315, Springer-Verlag, Berlin, 1997.

[9] D. G. Cantor and E. Kaltofen, *On fast multiplication of polynomials over arbitrary algebras*, Acta Inform., 28 (1991), pp. 693–701.

[10] P. Cartier, *Une nouvelle opération sur les formes différentielles*, C. R. Acad. Sci. Paris, 244 (1957), pp. 426–428.

[11] D. V. Chudnovsky and G. V. Chudnovsky, *Approximations and complex multiplication according to Ramanujan*, in Ramanujan Revisited (Urbana-Champaign, IL, 1987), Academic Press, Boston, MA, 1988, pp. 375–472.

[12] S. Cook, *On the Minimum Computation Time of Functions*, Ph.D. thesis, Harvard University, Cambridge, MA, 1966.

[13] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, J. Symbolic Comput., 9 (1990), pp. 251–280.

[14] P. Flajolet and B. Salvy, *The SIGSAM challenges: Symbolic asymptotics in practice*, SIGSAM Bull., 31 (1997), pp. 36–47.

[15] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, Cambridge University Press, Cambridge, UK, 1999.

[16] J. von zur Gathen and V. Shoup, *Computing Frobenius maps and factoring polynomials*, Comput. Complexity, 2 (1992), pp. 187–224.

[17] P. Gaudry and N. Gürel, *Counting points in medium characteristic using Kedlaya's algorithm*, Experiment. Math., 12 (2003), pp. 395–402.

[18] P. Gaudry and R. Harley, *Counting points on hyperelliptic curves over finite fields*, in Algorithmic Number Theory (ANTS-IV), Lecture Notes in Comput. Sci. 1838, Springer, Berlin, 2000, pp. 313–332.

[19] P. Gaudry and É. Schost, *Construction of secure random curves of genus* 2 *over prime fields*, in Advances in Cryptology (EUROCRYPT 2004), C. Cachin and J. Camenisch, eds., Lecture Notes in Comput. Sci. 3027, Springer, Berlin, 2004, pp. 239–256.

[20] G. Hanrot, M. Quercia, and P. Zimmermann, *The middle product algorithm,* I. *Speeding up the division and square root of power series*, Appl. Algebra Engrg. Comm. Comput., 14 (2004), pp. 415–438.

[21] H. Hasse and E. Witt, *Zyklische unverzweigte Erweiterungskörper vom Primzahlgrade p über einem algebraischen Funktionenkörper der Charakteristik p*, Monatsch. Math. Phys., 43 (1936), pp. 477–492.

[22] E. Horowitz, *A fast method for interpolation using preconditioning*, Inform. Process. Lett., 1 (1972), pp. 157–163.

[23] E. Kaltofen, R. M. Corless, and D. J. Jeffrey, *Challenges of symbolic computation: My favorite open problems*, J. Symbolic Comput., 29 (2000), pp. 891–919.

[24] K. S. Kedlaya, *Counting points on hyperelliptic curves using Monsky-Washnitzer cohomology*, J. Ramanujan Math. Soc., 16 (2001), pp. 323–338.

[25] D. E. Knuth, *The analysis of algorithms*, in Actes du Congrès International des Mathématiciens (Nice, 1970), Tome 3, Gauthier-Villars, Paris, 1971, pp. 269–274.

[26] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard, *The number field sieve*, in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, ACM, New York, 1990, pp. 564–572.

[27] H. W. Lenstra, Jr., and C. Pomerance, *A rigorous time bound for factoring integers*, J. Amer. Math. Soc., 5 (1992), pp. 483–516.

[28] J. I. Manin, *The Hasse-Witt matrix of an algebraic curve*, Trans. Amer. Math. Soc., 45 (1965), pp. 245–264.

[29] K. Matsuo, J. Chao, and S. Tsujii, *An improved baby step giant step algorithm for point counting of hyperelliptic curves over finite fields*, in Algorithmic Number Theory (ANTS-V), Lecture Notes in Comput. Sci. 2369, Springer, Berlin, 2002, pp. 461–474.

[30] J. McKee and R. Pinch, *Old and new deterministic factoring algorithms*, in Algorithmic Number Theory (Talence, 1996), Lecture Notes in Comput. Sci. 1122, Springer, Berlin, 1996, pp. 217–224.

[31] R. T. Moenck and A. Borodin, *Fast modular transforms via division*, in Proceedings of the Thirteenth Annual IEEE Symposium on Switching and Automata Theory (University of Maryland, College Park, MD), 1972, pp. 90–96.
[32] P. L. Montgomery, *Speeding the Pollard and elliptic curve methods of factorization*, Math. Comp., 48 (1987), pp. 243–264.
[33] P. L. Montgomery, *An FFT Extension of the Elliptic Curve Method of Factorization*, Ph.D. thesis, University of California, Los Angeles CA, 1992.
[34] J. Pila, *Frobenius maps of abelian varieties and finding roots of unity in finite fields*, Math. Comp., 55 (1990), pp. 745–763.
[35] J. M. Pollard, *Theorems on factorization and primality testing*, Proc. Cambridge Philos. Soc., 76 (1974), pp. 521–528.
[36] C. Pomerance, *Analysis and comparison of some integer factoring algorithms*, in Computational Methods in Number Theory, Part I, Math. Centre Tracts 154, Math. Centrum, Amsterdam, 1982, pp. 89–139.
[37] T. Satoh, *The canonical lift of an ordinary elliptic curve over a finite field and its point counting*, J. Ramanujan Math. Soc., 15 (2000), pp. 247–270.
[38] A. Schönhage, *Schnelle Berechnung von Kettenbruchentwicklungen*, Acta Inform., 1 (1971), pp. 139–144.
[39] A. Schönhage, *Schnelle Multiplikation von Polynomen über Körpern der Charakteristik* 2, Acta Inform., 7 (1977), pp. 395–398.
[40] A. Schönhage, A. F. W. Grotefeld, and E. Vetter, *Fast Algorithms*, Bibliographisches Institut, Mannheim, 1994.
[41] A. Schönhage and V. Strassen, *Schnelle Multiplikation großer Zahlen*, Computing, 7 (1971), pp. 281–292.
[42] V. Shoup, *NTL: A library for doing number theory*. http://www.shoup.net/ntl (2005).
[43] V. Shoup, *A fast deterministic algorithm for factoring polynomials over finite fields of small characteristic*, in Proceedings of the International Conference on Symbolic and Algebraic Computation, ACM Press, New York, 1991, pp. 14–21.
[44] J. H. Silverman, *The Arithmetic of Elliptic Curves*, Graduate Texts in Math. 106, Springer-Verlag, New York, 1996.
[45] A. Stein and H. Williams, *Some methods for evaluating the regulator of a real quadratic function field*, Experiment. Math., 8 (1999), pp. 119–133.
[46] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.
[47] V. Strassen, *Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten*, Numer. Math., 20 (1972/73), pp. 238–251.
[48] V. Strassen, *Einige Resultate über Berechnungskomplexität*, Jber. Deutsch. Math.-Verein., 78 (1976/77), pp. 1–8.
[49] B. Tellegen, *A general network theorem, with applications*, Philips Res. Rep., 7 (1952), pp. 259–269.
[50] N. Yui, *On the Jacobian varieties of hyperelliptic curves over fields of characteristic* $p > 2$, J. Algebra, 52 (1978), pp. 378–410.